University of Pisa

Master Thesis

A fast optimization algorithm for Moving Horizon Estimation

Candidate:
Bruno Morabito

Supervisors:
Prof. Ing. Gabriele Pannocchia
Prof. Ing. Rolf Findeisen
Prof. Ing. Claudio Scali

May 2015

Master's Degree in Chemical Engineering

Department of Civil and Industrial Engineering

Declaration of Authorship

I, Bruno MORABITO, declare that this thesis titled, 'A fast optimization algorithm for Moving Horizon Estimation' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Continue to act thus, my dear Lucilius, set yourself free for your own sake; gather and save your time, which till lately has been forced from you, or filched away, or has merely slipped from your hands. Make yourself believe the truth of my words, that certain moments are torn from us, that some are gently removed, and that others glide beyond our reach. The most disgraceful kind of loss, however, is that due to carelessness. Furthermore, if you will pay close heed to the problem, you will find that the largest portion of our life passes while we are doing ill, a goodly share while we are doing nothing, and the whole while we are doing that which is not to the purpose. What man can you show me who places any value on his time, who reckons the worth of each day, who understands that he is dying daily? For we are mistaken when we look forward to death; the major portion of death has already passed. Whatever years be behind us are in death's hands.

Therefore, Lucilius, do as you write me that you are doing: hold every hour in your grasp. Lay hold of today's task, and you will not need to depend so much upon tomorrow's. While we are postponing, life speeds by. Nothing, Lucilius, is ours, except time. We were entrusted by nature with the ownership of this single thing, so fleeting and slippery that anyone who will can oust us from possession. What fools these mortals be! They allow the cheapest and most useless things, which can easily be replaced, to be charged in the reckoning, after they have acquired them; but they never regard themselves as in debt when they have received some of that precious commodity, time! And yet time is the one loan which even a grateful recipient cannot repay.

You may desire to know how I, who preach to you so freely, am practising. I confess frankly: my expense account balances, as you would expect from one who is free-handed but careful. I cannot boast that I waste nothing, but I can at least tell you what I am wasting, and the cause and manner of the loss; I can give you the reasons why I am a poor man. My situation, however, is the same as that of many who are reduced to slender means through no fault of their own: every one forgives them, but no one comes to their rescue.

What is the state of things, then? It is this: I do not regard a man as poor, if the little which remains is enough for him. I advise you, however, to keep what is really yours; and you cannot begin too early. For, as our ancestors believed, it is too late to spare when you reach the dregs of the cask. Of that which remains at the bottom, the amount is slight, and the quality is vile. Farewell."

Letter from Seneca to his friend Lucilius

UNIVERSITY OF PISA

Abstract

Chemical Engineering
Department of Civil and Industrial Engineering

Master Thesis in Chemical Engineering

A fast optimization algorithm for Moving Horizon Estimation

by Bruno MORABITO

The Moving Horizon Estimation (MHE) is a technique that allows to estimate the states of a system considering constraints, either when they are affected by noise or are not measured. This method can be associated with control techniques such as Model Predictive Control (MPC).

The core of the mathematics formulation of MHE consists of an optimization problem that can easily become huge as the horizon and the number of states of the system increase. This leads inevitably to a large computational time that makes difficult the implementation of the algorithm for on-line purpose. In this work we develop a fast and simple algorithm that solves the MHE problem using the Nesterov's Fast Gradient Method that, under certain assumption, solves the optimization problem faster than using the standard optimization algorithms such as Interior Point Method or Active Set Method. Contrary to the MPC problem, the Hessian resulting from the MHE is time-varying. This poses a problem on the calculation of the eigenvalues which are required by the Fast Gradient method, therefore we implement also a fast algorithm for obtaining a upper and lower bound on the eigenvalues of the Hessian.

The algorithm has been validated through several simulations on linear random systems and then applied on two practical examples. The first consists in applying the MHE together with MPC in a anaerobic digestor for methane production. Anaerobic digestion plants often suffer from a slow and unreliable on-line measurement systems therefore the automatic control is often difficult, for this reason we aim to solve this problem estimating most of the states instead of measuring them so as to apply the control action with the estimated information. The second is a chemical plant section consisting of two CSTR reactors and a nonadiabatic flash separator where we suppose some states are not measurable.

Acknowledgements

I would like to express my sincere thanks to Prof. Gabriele Pannocchia and Prof. Rolf Findeisen who made my awesome experience at the Otto von Guericke University possible, who gave me advice, guidance, help and great opportunities for my career and life.

I am thankful to all the members of the Findeisen research group, in particular to Makus Kögel who showed a remarkable patience with my numerous questions and concerns, to Eric Bullinger who gave me lots of useful tips and suggestions and to Ulrike Thürmer for her kindness on helping me with bureaucratic issues.

Furthermore I would like to show my gratitude to my amazing friends of Pisa with whom I have spend the last five years. All of them gave me an incredible support, help and inspiration and have been my second family for all that time. A particular appreciation goes to Giuseppe Forte, Marina Polignano and Giacomo Bartali with whom I have shared great part of the most intense moments of a student's life and my flatmate Raffaele Zaccara who has been standing me for the last four years.

To my family, of course, goes all my gratitude for their support and teaching.

Bruno Morabito
Pisa, May 2015

Contents

List of Figures

List of Tables

Abbreviations

AS	**A**ctive **S**et
EKF	**E**stended **K**alman **F**ilter
FG	**F**ast **G**radient
FIE	**F**ull **I**noformation **E**stimation
II	**I**nverse **I**iteration
KF	**K**alman **F**ilter
LP	**L**inear **P**rogramming
LS	**L**inear **S**istem
MHE	**M**oving **H**orizon **E**stimation
MPC	**M**odel **P**redictive **C**ontrol
NLS	**N**on **L**inear **S**istem
NMPC	**N**onlinear **M**odel **P**redictive **C**ontrol
UASB	**U**pflow **A**naerobic **S**ludge **B**lanket reactor (UASB)
UKF	**U**nscented **K**alman **F**ilter

Symbols

$\mathcal{R}(A)$	Range of matrix A.
$\mathcal{N}(A)$	Kernel of matrix A.
$\mathbb{R}$	Set of real numbers.
$\mathbb{N}$	Set of natural numbers.
$\mathbb{Z}$	Set of relative numbers.
$\mathbb{R}^n$	Euclidean space of dimension n.
$\mathbb{R}^{n \times n}$	Matrix of dimension $n \times n$.
$\mathcal{X}$	Set of constraints on the states.
$\mathcal{V}$	Set of constraints on the measurement error.

x	Vector of system states.
u	Vector of system inputs.
y	Vector of system outputs.
w	Disturbance on the states.
v	Disturbance on the measurements.
f	Generic function.
r	Rayleigh Quotient.
$g(\cdot)$	Equality constrains vector.
$h(\cdot)$	Inequality constraints vector.
$\hat{\mathbf{x}}$	Stacked vector of optimization variables. $\hat{\mathbf{x}} = (\hat{x}_k^T, \hat{x}_{k+1}^T, ..., \hat{x}_{k+j}^T)$
$\tilde{x}_0$	Best a priori information about the states.
A	Matrix of the states.
B	Matrix of the inputs.
C	Matrix of the outputs.
J	Cost function of MHE/MPC problem.

G	Lagrange dual function.
H	Matrix of the Quadratic Programming problem.
L	Luenberger Observer matrix.
L	Lagrange function (Chapter 2).
L	Largest matrix eigenvalue (from Chapter 4).
N	Horizon length (number of steps).
R	Variance matrix of the measurements disturbances.
Q	Variance matrix of the states disturbances.
dom	Domain.
cond	Condition number.
relint	Relative interior.
rank	Rank of a matrix.
$\min_{(\cdot)}(f(\cdot))$	Minimum of function f.
$\|x\|_2$	Euclidean norm ($\sqrt{x^T x}$).
$\|x\|_A$	Weighted Euclidean norm ($\sqrt{x^T A x}$).
$\nabla(\cdot)$	Gradient.
$\hat{(\cdot)}$	Optimization variable.
$(\cdot)^T$	Matrix or vector transposition.
$(\cdot)_k$	Discrete variable at time k.
$(\cdot)^+$	Discrete variable one step ahead in time.
$(\cdot)_{j\|k}$	Value of the variable at time j calculated at time k.
$(\cdot)^-$	A priori variable (not update with the last measurements).
$(\cdot)^*$	Optimal point.
$P_{j\|k}$	Prior weighting Matrix.
Γ	Controllability matrix.
Ω	Observability matrix.
λ, μ	Lagrange multipliers.
μ	Smallest eigenvalue (from Chapter 4).
ϵ	Tollerance.

To my family.

Chapter 1

Introduction

1.1 Motivation

State estimation on dynamical systems plays an important role in feedback control, states monitoring, fault detection and system optimization. The problem of state estimation arises either when we the state is not physically measurable or is disturbed by an noise that we want to filter. For example in the field optimal control, where a control action is applied to a system (*e.g.*: a chemical plant, a robot, a vehicle etc...) in order to lead a *cost function* to a minimum, we need to know the states of the system in order to compute the most appropriate control action. These states are often given by a *state estimator*. Various state estimator algorithms have been implemented and important results have been carried out in recent years.

Maybe the most famous and widespread state estimation technique is the *Kalman Filter* [1], used, for example, in objects tracking (e.g., missiles, faces, heads, hands), economics, navigation and in many computer vision applications (like stabilizing depth measurements, feature tracking, cluster tracking, fusing data from radar etc.). The main advantage of this method is its *recursiveness*, namely it does not require to store past data in order to estimate the *most likely states* at the next time step. This property has revealed to be of huge importance for obtaining fast solution and has allowed the on-line application of this method. Nevertheless, Kalman filter has some disadvantages: firstly it does not take into account *constraints on the variables* [2], secondly the solution of the problem loses its simplicity if we cannot assume gaussian distributed noises or the system model is not linear. This forces us to previously linearise the system, in this case we talk about *Extended Kalman Filter* [3]. The *Unscented Kalman Filter* (UKF) [4] is applicable directly on nonlinear problem but does not allow to introduce constraints.

Considering state constraints can significantly improve the estimation performance [5] (usually they are *physical* bound, such as the maximum level of liquid in a tank) and

dealing directly with nonlinear problem would ease the burden of the linearisation step. In order to solve those limitations an approach that has recently received much interest is the Moving Horizon Estimation (MHE). This method constructs an *weight function* which has a minimum when the values of its variables are (approximately) near the the values of real system states. Intuitively we can thing of the MHE approach like a way that, given the outputs of a system from a certain time in the past up to the present moment, and given an estimation of the noise variance that effects the system, gives us the *most likely* system states that are causing those specific outputs.

The main problem with the MHE problem is the computational effort required for solving the algorithm that precludes us to use it when the computing power is limited and/or the data sampling rates are excessive. As we are going to show the bottleneck of this approach is usually the *optimization algorithm*, namely the algorithms that, given the cost function, founds its minimum, usually represented by a particular set of states at time k. A review on the optimization algorithm for MHE and Nonlinear Model Predictive Control (NMPC) can be consulted in [6]. In order to solve this bottleneck we are going to apply the Fast Gradient Method of Nesterov (FG) [7] and we are going to show that under the assumption of *box constraints* (as well as other simple sets of constrains) this method easily fits our problem and gives a faster solution than other standard optimization techniques.

1.2 Thesis overview

This thesis consists of 5 chapters :

- In the present chapter, we discuss about the motivation and the goal of this work, about the mathematical notation used along it and a literature review that covers the state of the art of MHE. Respect to the huge number of publications regarding the MPC, the MHE is still a new argument in the scientific community, this explains the little amount of literature available.
- In Chapter 2 we report the theory concerning optimal control (in particular MPC) and MHE. Starting from the general case in discrete-time systems we make our way to the MHE formulation used in this work, which represent, together with the Fast Gradient method, an unusual approach to the MHE problem.
- In Chapter 3 we are going to have a look at the theoretical background that we need for a clear understanding of the problem. In particular we are going to give some fundamental concepts on the optimization problem, convex optimization, and the conditions that hold in the optimal point. We point out that different formulation

lead to different problem structure that can be useful for a particular type of optimization algorithm and can reduce the condition number of the problem.

- In Chapter 4 we are going to resume the most important optimization algorithms and techniques, such as *Descend method, Newton's Method, Interior Point method* and finally the Nesterov's Fast Gradient method.

- In Chapter 5 we show the results that led us to the final version of the algorithm. We deal discuss about the class of system that we have studied, our approach to the estimation and control problems, how we have dealt with constraints, algorithms that have been used for the simulations, stopping criteria and eigenvalues computation. We are going to see that the Fast Gradient together with the *Inverse iteration method* and *Cholesky decomposition* necessary for computing the eigenvalues, we can solve the problem faster than the other standard optimization techniques, and furthermore the time required grows linearly with the matrix dimension.

- In Chapter 6 we apply our algorithm in two examples: an Upflow Anaerobic Sludge Blanket (UASB) reactor and a plant section composed by two CSTR reactors with a flash separator. In the UASB case we aim to control the reactor with a MPC based controller, so we need to estimate some states that cannot be reliably measured on-line. In the other example we do not control the system (which is stable) but we only estimate two states that we assume not measurable. Furthermore we compare the speed of the algorithm with the Matlab solver *quadprog* and we point out that the FG algorithm is faster than its built-in methods.

- In Chapter 7 we resume briefly the results and we point out some other possible enhancement of the algorithm as well as other possible future research directions.

1.3 Literature review

The literature on MHE in linear systems appears scarce. Abrol et al [8] use MHE based on *In Situ Adaptive Tabulation* (ISAT) [9] in nonlinear discrete-time and continuous-time systems. ISAT can be used to store solution to the optimal estimation problem at every time step by running the MHE off-line and then using the ISAT stored model trajectories to retrieve the closest solution. The limitation of this method is of course the large storage requirement, but the authors claim that nowadays this rarely becomes a problem in storage and retrieval techniques. In this study they have carried simulations on a two-state CSTR reactor more than 300 times faster than standard MHE with good estimation accuracy.

Darby et al.[10] have implemented a lookup table and function evaluation for real-time implementation of MHE and have tested it on a linear system. However, the number of polytopes generated in the approach tends to grow combinatorially with the number of constraints, which limits the size of the problem that can be handled.

In [11] a primal barrier Interior-Point method algorithm has been applied on a linear system exploiting the structure of the KKT system which has been shown to be symmetric indefinite. The computational time in this approach grows linearly with the horizon length.

1.3.1 Contribution of this work

We propose for the MHE in constrained, linear time-discrete systems a simple, tailored algorithm based of Nesterov's fast gradient method which has been successfully applied in the *Model Predictive Control* (MPC) problem in [12], [13], [14], [15], [16]. Since the MHE and MPC formulation have several common characteristics, the idea is to use this method to the estimation problem. Furthermore we propose to eliminate the noise from the optimization variables in such a way that we obtain a sparse formulation with only the states as optimization variables. In contrast to MPC, in MHE the Hessian matrix and consequently its eigenvalues are time-varying due to the arrival cost. Since the considered gradient method requires the largest and smallest eigenvalues of the Hessian, we compute them at each step using the Inverse Iteration method[17] enhanced with the Cholesky factorization.

1.4 Notation

1.4.1 System notation

In the continuous-time case and for a generic nonlinear system, we write the system model with the following system of differential equations:

$$\frac{dx}{dt} = f(x, u, t) \tag{1.1}$$

$$y = h(x, u, t) \tag{1.2}$$

$$x(t_0) = x_0 \tag{1.3}$$

where $x \in \mathbb{R}^n$ is the vector of the states, $u \in \mathbb{R}^m$ is the vector of the inputs, $y \in \mathbb{R}^p$ is the vector of the outputs and $t \in \mathbb{R}$ is the time.

We are going to refer to a linear continuous- time time-invariant model using the following notation:

$$\begin{aligned} \frac{dx}{dt} &= Ax + Bu + Gw \\ y &= Cx + Du + v \\ x(0) &= x_0 \end{aligned} \tag{1.4}$$

in which $A \in \mathbb{R}^{nxn}$ is the *state transition matrix*, $B \in \mathbb{R}^{nxm}$ is the *input matrix*, $C \in \mathbb{R}^{pxn}$ is the *output matrix* and $D \in \mathbb{R}^{pxm}$ allows a direct coupling between u and y, but in many cases, like in this work, is not considered, namely $D = 0$.

In the majority of the applications a *discrete time model* is used, since in practice the measurements of states or output are taken with discrete sampling. In this case we refer to our system with the notation:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \\ x(0) &= x_0 \end{aligned} \tag{1.5}$$

in which $k \in \mathbb{N}$ is a nonnegative integer denoting the sample number, and is connected to time by $t = k\Delta$, where Δ is the sampling time.

More simply we can write:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \\ x_0 &\text{ given} \end{aligned} \tag{1.6}$$

or to reduce the notation further:

$$\begin{aligned} x^+ &= Ax + Bu \\ y &= Cx + Du \\ x(0) &= x_0 \end{aligned} \tag{1.7}$$

in which the $^+$ means the state at the next sample time.

1.4.2 Optimization notation

Let $J(\hat{\mathbf{z}}) \in \mathbb{R}^{Nn \times Nn} \to \mathbb{R}$ be a convex function. We define the constrained minimization problem as:

$$\min_{\hat{\mathbf{z}}} J(\hat{\mathbf{z}}) \tag{1.8a}$$

such that:

$$g_i(\hat{\mathbf{z}}) = 0 \quad k = 0, ..., N-1, \tag{1.8b}$$

$$h_i(\hat{\mathbf{z}}) \leq 0 \quad k = 0, ..., N-1, \tag{1.8c}$$

in which g_i represent the $i-th$ equality constraint and h_i the $i-th$ inequality constraint. In the optimization context, the notation $\hat{(\cdot)}$ represents the optimization variable, usually made of a vector of states, noises or inputs at a certain time k. For matter of simplicity we stack all the variable at different time in one vector, *e.g*:

$$\hat{\mathbf{z}} = (\hat{z}_k^T, \hat{z}_{k+1}^T, ..., \hat{z}_{k+j}^T)^T \quad k, j \in \mathbb{Z}$$

Chapter 2

Optimal control and estimation

The problem of optimal control and estimation are often correlated due to the fact that we need to know the states' evolution along the time in order to compute an optimal control action. An control action is *optimal* when it minimizes a given *cost function* that is designed usually by the engineers depending on the cost items of the system. For example, we want to travel from a city to another with our car as fast as possible but, at the same time, we want to spare our petrol. The engineer's job is to *weight* those two conflicting requirements (the time and the consumption of fuel) in a unique cost function that, through the optimization problem, will give us a optimal control action (how much we have to push the accelerator).

Optimal control and estimation not only work together but are often considered *dual* in the sense of their similar formulation [6] [18]. Indeed, both try to minimize the cost function (also called *objective function*) which variables are usually constrained, along a *time horizon* which can be finite or infinite. In the estimation case, solving the objective function means find the *most likely* states sequence that give the output we can measure.

We can obtain a simple solution of the optimization problem in the case of *infinite* horizon for unconstrained and linear problem, obtaining the Linear Quadratic Regulator (LQR) control law in the control side and the Kalman Filter in the estimation side. In case of nonlinear systems or/and constrained systems the solution becomes *non recursive* thus, while in the Kaman Filter or LQR controller we need to store only the data at the previous time step, in those cases we should store all the past data, that of course is not possible with an infinite horizon. That is why the Model Predictive Control (MPC) (control problem) and the Moving Horizon Estimation (estimation problem) use a *finite horizon* and summarize the "ignored" information in a single term.

Roughly we can say that the MPC looks at the future, predicting the behaviour of the system along a certain horizon and choosing consequently the best, or "cheapest" control action. The MHE instead looks at the *past* and tries to predict the state of the system with the information that it has collected along a certain horizon.

2.1 Optimal control

2.1.1 Model Predictive Control

In the most general case we deal with nonlinear systems, so we talk about Nonlinear Model Predictive Control (NMPC) [18] [19], furthermore we only consider discrete time optimization since it is more likely the case in real applications. The NMPC formulation is the following:

$$J(\hat{\mathbf{x}}, \hat{\mathbf{u}}) = \sum_{k=0}^{N_{MPC}-1} [L_k(\hat{x}_k, \hat{u}_k)] + E(\hat{x}_{N_{MPC}}) \tag{2.1a}$$

$$\min_{\hat{\mathbf{x}}, \hat{\mathbf{u}}} J(\hat{\mathbf{x}}, \hat{\mathbf{u}}) \tag{2.1b}$$

$$\text{such that:} \quad \hat{x}_0 - \bar{x}_0 = 0, \tag{2.1c}$$

$$\hat{x}_{k+1} - f_k(\hat{x}_k, \hat{u}_k) = 0 \quad k = 0, ..., N_{MPC} - 1, \tag{2.1d}$$

$$g_i(\hat{x}_k, \hat{u}_k) = 0 \quad k = 0, ..., N_{MPC} - 1, \tag{2.1e}$$

$$h_i(\hat{x}_k, \hat{u}_k) \leq 0 \quad k = 0, ..., N_{MPC} - 1, \tag{2.1f}$$

$$r(\hat{x}_{N_{MPC}}) \leq 0. \tag{2.1g}$$

L_k and E are general weight functions while N_{MPC} is the horizon *length*, that is the number of time step in the future that we consider in the optimization (for other notation see § 1.4.2). This is an example of *Nonlinear programming* (NLP) [20] problem and, in general, only a *numerical* solution exists, even for linear systems, and it requires the application of optimization algorithms that can deal with constraints *e.g.* Interior Point method, Active Set etc.. The solution consists of $\{u_j\}$ control inputs with $j = 1, ...N_{MPC} - 1$ and $\{x_l\}$ states with $j = 1, ...N_{MPC}$, the control system takes only the first optimized input u_0 and applies it to the plant and then, at the next time step, the optimization is carried out again obtaining new $\{u_j\}$ and $\{x_l\}$.

If we consider only linear systems, do not consider the constraints and choose quadratic weighting costs it is possible to find an analytical and quite simple solution. This solution gives us a control law $u_k = -Kx_k$ that minimize the cost function. In this case we talk about *Linear Quadratic Regulator*.

2.1.2 Linear Quadratic Regulator

In the regulation problem, we want to drive state and input to zero by solving an optimization problem that found the minimum of this objective function:

$$J(x(0), \mathbf{u}) = \frac{1}{2} \sum_{k=0}^{N-1} \left[x_k^T Q x_k + u_k^T R u_k \right] + \frac{1}{2} x_N^T P_f x_N \tag{2.2}$$

subject to $x^* = Ax + Bu$, where $\mathbf{u} = (u_0^T, ..., u_i^T ..., u_{N-1}^T)$ represent the control input at different time steps. Q, P_f and R are chosen to be real diagonal matrices (therefore symmetric), furthermore Q and P_f are *positive semidefinite* and R is *positive definite*. These assumption guarantee that the solution to the control problem *exist and is unique*. The solution to this problem will be an *optimal control policy* $u(\cdot)$ that minimize the cost function $J(\cdot)$. Furthermore they have to be chosen in order to weight more the control action or the state that we want to drive to zero as soon as possible since, generally, it is the most expensive in terms of energy, fuel consumption etc. Thanks to the quadratic form and the absence of constraints, we can solve the LQR problem that consist of minimizing the Eq. 2.2 using the principle of *dynamic programming* of Bellman [21]. The solution is recursive, requires low computational effort and give us a constant gain K and therefore a control law $u = -Kx$ that minimizes the objective function.

Anyway the problem 2.2, does not ensure a stable control law. If we want to ensure stability of the system (it is always the case for continuous processes) we have to make some further assumptions.

2.1.3 Controllability

Definition 2.1 (Completely controllable system)**.** The system is *completely controllable* (CC) if it is possible to go from *any* initial state to *any* final state in a finite time.

Let us translate the Definition 2.1 in algebraic formulae. If from state x_0 we want to go to $\tilde{x}$ in N steps, that means:

$$\begin{aligned}
x_1 &= Ax_0 + Bu_0 \\
x_2 &= Ax_1 + Bu_1 = A^2x_0 + ABu_0 + Bu_1 \\
x_3 &= Ax_2 + Bu_2 = A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2 \\
&\vdots \\
\tilde{x} &= A^Nx_0 + A^{N-1}Bu_0 + \ldots + Bu_{N-1}
\end{aligned}$$

that is equivalent to look for the existence of solution to the following linear system *for an arbitrary right-hand side*:

$$\begin{bmatrix} B & AB & \cdots & A^{N-1} & B \end{bmatrix} \begin{bmatrix} u_{N-1} \\ u_{N-2} \\ \vdots \\ u_0 \end{bmatrix} = z - A^Nx \tag{2.3}$$

Remember that $A \in \mathbb{R}^{n\times n}$. It is possible to demonstrate that the matrix power A^h with $h > n$ can be written as linear combination of A^g where rank $(A) = g$. Therefore the the rank of C with for a certain $N > n$ is always, at maximum, n. This means that if we cannot reach the $\tilde{x}$ in n steps, we cannot reach $\tilde{x}$ in *any number of steps*. We define $\Gamma = \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix}$ *controllability matrix*.

Theorem 2.2 (Complete controllability (CC))**.** *A linear, time-invariant system of the form*

$$x_{k+1} = Ax_k + Bu_k \tag{2.4}$$

$$y_k = Cx_k \tag{2.5}$$

where $A \in \mathbb{R}^{n\times n}$ *is* completely controllable *if and only if*

$$rank\ (\Gamma) = n. \tag{2.6}$$

in this case we say that (A, B) *is completely controllable.*

Now if we let N go to infinity we obtain the problem:

$$J(x(0), \mathbf{u}) = \frac{1}{2}\sum_{k=0}^{\infty}\left[x_k^T Q x_k + u_k^T R u_k\right] \tag{2.7}$$

it is possible to demonstrate [5] that the solution to this problem is obtained using the optimal control policy:

$$u^0(x) = Kx$$

where

$$K = -(B'\Pi B + R)^{-1}B'\Pi A \tag{2.8}$$

$$\Pi = Q + A'\Pi A - A'\Pi B(B'\Pi B + R)^{-1}B'\Pi A \tag{2.9}$$

in which the Eq. 2.9 is the *Riccati Algebraic Equation* (RAE), that has solution if $Q, R > 0$ and (A, B) is controllable. This leads to an optimal constant gain K. We call this type of regulator *Linear Quadratic Regulator* (LQR).

Lemma 2.3 (LQR convergence)**.** *Given a linear time-invariant system of the form 2.4, with* (A, B) *controllable and* $Q, R > 0$ *the LQR gives a convergent closed-loop system.*

Proof. With the assumption of (A, B)*controllability*, *positive definiteness* of Q and R the function $J(x_0, \mathbf{u})$ is a *Lyapunov function* (see Appendix B). □

The systems chosen in this work are all controllable and Q and R are positive definite. Therefore the LQR controller provides a convergent closed-loop system.

2.2 Estimation

In this section we go quickly over the most widespread estimation techniques, finishing with a more complete dissertation on the MHE.

2.2.1 Luenberger Observer

This observer is formed by using a model in parallel to the real process [22]. In order to take into account mismatch between real process and model the Luenberger Observer adds a feedback of the value $y_k - \hat{y}_k$ where y_k is the real output and $\hat{y}_k$ is the model output. The model is therefore extended with this feedback:

$$\hat{x}_{k+1} = A\hat{x}_k + Bu + u_B \tag{2.10}$$

$$\hat{y} = C\hat{x}_k \tag{2.11}$$

where $u_B = L(y - \hat{y})$. The error between real and model state at time k is $e_k = x_k - \bar{x}_k$ and at time $k+1$ is $e_{k+1} = x_{k+1} - \hat{x}_{k+1}$. Subtracting these two terms we obtain:

$$e_{k+1} = (A - LC)e_k$$

which means that the error goes to zero if the eigenvalues of $(A - LC)$ are smaller than one. The design phase consists of choosing the right poles of the closed loop observer.

2.2.2 Kalman Filter

Introduced by Kalman [1] has been used in several engineering contexts. It estimates the system states by minimizing the average of the squared error. The system has the form:

$$x_{k+1} = Ax_k + Bu_k + w_k$$

$$y_k = Cx_k + v_k -$$

The filter estimates two types of states: the *a priori* estimate $x_{k|k-1}$ and the *a posteriori* estimate $x_{k|k}$, therefore the method can be summarized in two parts, one of *time updating* where we predict the state and one of measurement updating for estimate the state. The aim is to calculate an a posteriori estimate as a linear combination of the a priori estimate and a weighted error between the measurements and the predicted measurements. This difference between measurement and predicted measurement is called *innovation* or *residual*.

$$x_{k|k} = x_{k|k-1} + K_k(y_k - Cx_{k|k-1})$$

The a priori estimate can be calculated with the model equations. The weighting matrix K_k regulates the influence of the error between the measurement and the predicted measurement and minimizes the a posteriori error covariance.

$$\text{time update} \quad \begin{cases} \hat{x}_{k|k-1} &= A\hat{x}_{k-1|k-1} + Bu_{k-1} \\ P_k^- &= AP_{k-1}A^T + Q \end{cases}$$

$$\text{measurement update} \quad \begin{cases} K_k &= P_k^- C^T (CP_{k-1}C^T + R)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}) \\ P_k &= (I - K_k C)P_k^- \end{cases}$$

Q and R are symmetric positive semi-definite matrices representing the covariances of the normally distributed random noises w and v. $P_k^-$is the covariance of the estimated state $\hat{x}_{k|k-1}$ and P_k the covariance of the estimated state $\hat{x}_{k|k}$. The measurement update equations will start at $k = 0$, while the time update equations are valid for $k = 1, 2, 3, ...$ The initial prediction values can be defined for example as

$$\hat{x}_{0|-1} = \bar{x}_0$$
$$P_{0|-1} = Q_0$$

with $\bar{x}_0$ being an independent normally distributed random with covariance Q_0.

2.2.3 Full Information Estimation

The Full Information Estimation (FIE) permits to consider constraints in the estimation problem. It carries out a constrained optimization in an horizon that increases with the time. It has shown to have the best theoretical properties in terms of stability and optimality[18], but it becomes computational intractable except some simple cases. The full information objective function is:

$$J(\hat{x}_0, \hat{\mathbf{w}}) = \Gamma_0(\hat{x}_0 - \bar{x}_0) + \sum_{i=k-N}^{k-1} l_i(w_k, v_k) \tag{2.12}$$
$$x_{k+1} = f(x_k, w_k)$$
$$y_k = h(x_k) + v_k$$

where l_i and l_x are generic stage costs. The FIE is solved by minimizing the problem 2.12.

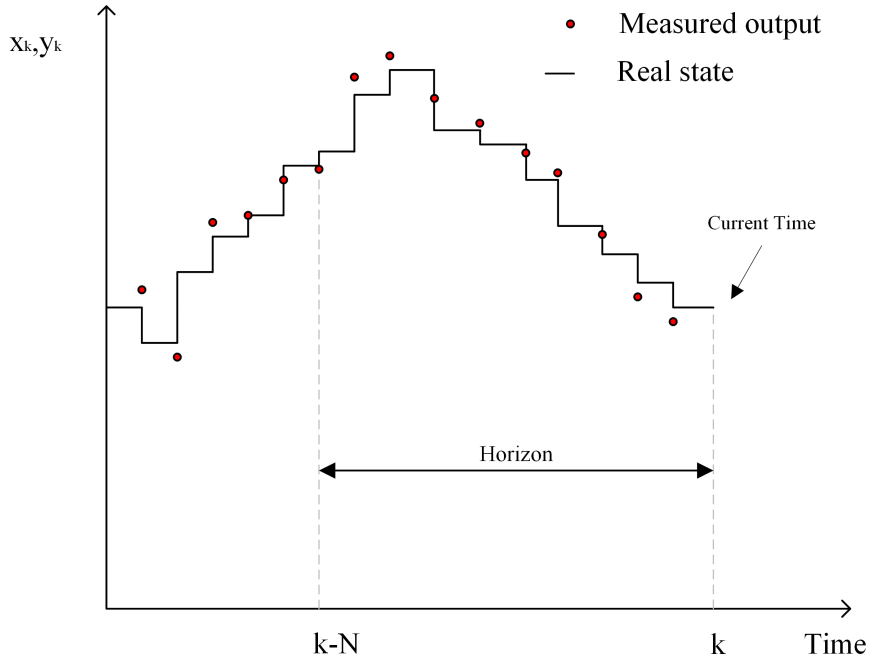


FIGURE 2.1: Graphical representation of the moving horizon.

2.2.4 Moving Horizon Estimation

Since the FIE is not practically usable due to its computational complexity, the MHE considers only a few steps in the past. Moving Horizon Estimation is a quite new technique [23] (a complete dissertation can be found in [18]). For the MHE we usually use a convex function to penalize the mismatch between the real measurement and the model prediction represented by v_k , similarly for the disturbances w_k, usually introduced to account for plant-model mismatch. Furthermore we weight quadratically the difference between the state at the beginning of the horizon $\hat{x}_{k-N}$ and our *best initial information* $\tilde{x}_{k-N}$ that we have of it. Notice that the control action is fixed and time-variant in the MHE problem, thus we cannot change it. Therefore the *standard* objective function looks like:

$$J(\hat{\mathbf{w}}, \hat{\mathbf{x}}) = \frac{1}{2}\|\tilde{x}_{k-N} - \hat{x}_{k-N}\|^2_{P^{-1}_{k-N}} + \sum_{i=k-N}^{k-1} \frac{1}{2}\|\hat{w}_i\|^2_{Q^{-1}} + \sum_{i=k-N}^{k} \frac{1}{2}\|v_k\|^2_{R^{-1}}. \tag{2.13}$$

The weighting matrices Q and R are chosen as close as possible to the noises covariance, that is because if, say, w has a broader noise distribution (large covariance) than v, we are going to *trust* more the difference $y - Cx$, forcing it to be smaller with a larger weight. The covariance indeed matches this need: if Q or R are small Q^{-1} or R^{-1} are large and vice versa.

The matrix P^{-1}_{k-N} represent the *prior weighting matrix* and, conversely to R^{-1} and Q^{-1}, it is time variant, thus is updated each time step with new information available in

order to have a *stable* estimation. For linear systems, quadratic objective function and convergence disturbances it is possible to demonstrate the we obtain a stable estimator if we update the matrix P_{k-N}^{-1} with Riccati iteration.

The parameter modifiable by the engineers are the weighting matrices P_0^{-1},Q^{-1},R^{-1} (P_0^{-1} should be close to the covariance of the state distribution at the beginning of the horizon) the first estimation $\tilde{x}_0$ and the horizon length N.

We need to update $\tilde{x}_{k-N}$ in the prior weighting. There are mainly two method for it the *filtering update* and the *smoothing update*. The filtering update uses the value $\tilde{x}_{k-N|k-N}$ and therefore needs to store the solution of the last $k-N$ MHE problems, the smoothing update instead uses the $\tilde{x}_{k-N|k}$. In our case we decided to use the smoothing update.

2.2.4.1 Observability

In general we cannot be sure that the estimator converges to the real system states. Let us define the *Robust Global Asymptotic Stability* (RGAS) of the estimator:

Definition 2.4 (Robust Global Asymptotic Stability for MHE)**.** The estimate based on noisy measurement is RGAS if for all $\hat{x}_0$ and $\bar{x}_0$ and convergent w, v the following hold:

- The estimate converges to the state as $k \to \infty$

$$\hat{x}_k \to x_k$$

- For every $\epsilon > 0$ there exists $\delta > 0$ such that

$$\frac{1}{2}\|\tilde{x}_0 - \hat{x}_0\|^2_{P_0^{-1}} + \sum_{i=0}^{\infty} \frac{1}{2}\|\hat{w}_i\|^2_{Q^{-1}} + \sum_{i=0}^{\infty} \frac{1}{2}\|v_k\|^2_{R^{-1}} \leq \delta$$

  this implies $|x_k - \hat{x}_k| \leq \epsilon$ for all k.

The first question that one can ask is: are the measurement "rich" enough in order to be able to estimate some of the states that we cannot measure directly? Intuitively we can understand that, in the linear case, it depends on two matrices: the matrix A of the system that express how the states are *connected* between each other, and the matrix C that express which states we are measuring. If we are measuring the a right number and/or the right states, we can obtain indirectly information about the other states. This property is called *observability*.

Theorem 2.5 (Observability for linear systems)**.** *A linear time-invariant time-discrete system*

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k$$

where $x_k \in \mathbb{R}^n$ *is* observable *if the rank of the* observability matrix $\mathcal{O}$

$$\Omega = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

is equal to n*, that is* Ω *is full rank. In this case we say also that* (A, C) *is observable.*

For optimal estimators, like the Kalman filter, the conditions for a convergent estimation to the states $\hat{x}_k \to x_k$ are that the system (A, C) has to be observable, $Q, R > 0$ and the errors have to be convergent to zero.

Notice that in some cases the observability requirement is too strict and it can be weakened into the *detectability*.

Definition 2.6 (Detectability)**.** A system is *detectable* if the unobservable states are asymptotically stable.

With those definition we can now define when a constrained MHE for linear systems is robust GAS:

Theorem 2.7 (Robust GAS of constrained MHE in linear systems)**.** *Consider a detectable linear system, with convergent constrained disturbances. The constrained MHE estimator using the prior weighting an optimal unconstrained estimator (like the Kalman Filter) is robustly GAS.*

2.3 Different MHE formulations

During this section we consider only linear systems:

$$x^+ = Ax + Bu + w$$
$$y = Cx + v$$
$$x(0) = x_0$$

and we show how the choice of the optimization variable plays an important role in the problem structure. A study on the effect of these structure has been carried out in the thesis [24]. Remember that :

$$x, w \in \mathbb{R}^n, \quad u \in \mathbb{R}^m \quad , y, v \in \mathbb{R}^p, \quad A \in \mathbb{R}^{n \times n}, \quad B \in \mathbb{R}^{m \times m}, \quad C \in \mathbb{R}^{p \times n},$$

If we choose a quadratic objective function like 5.5 we obtain in general a Quadratic Programming (QP) problem, which has the following form:

$$\min_{\hat{\mathbf{z}}} \frac{1}{2}\hat{\mathbf{z}}^T H \hat{\mathbf{z}} + \hat{\mathbf{z}}^T f \qquad \text{such that:} \quad \begin{cases} C\hat{\mathbf{z}} & \leq d \\ \mathcal{A}\hat{\mathbf{z}} & = b \\ l_b \leq & \hat{\mathbf{z}} \leq u_b \end{cases} \tag{2.14}$$

where $H \in \mathbb{R}^{Nn \times Nn}$ (called *Hessian matrix*), $\hat{\mathbf{z}} \in \mathbb{R}^{Nn}$, N is the horizon length.

2.3.1 Noises and states as optimization variables

The "classic" way to express the problem is to leave as optimization variables the states and the disturbances w, namely:

$$\hat{\mathbf{z}} = (\hat{x}_{k-N}^T, \hat{w}_{k-N}^T, \hat{x}_{k-N+1}^T, \hat{w}_{k-N+1}^T, ..., \hat{x}_{k-1}^T, \hat{w}_{k-1}^T)^T.$$

In this case we have that the MHE problem is:

$$J(\hat{\mathbf{w}}, \hat{\mathbf{x}}) = \frac{1}{2}\|\tilde{x}_{k-N} - \hat{x}_{k-N}\|^2_{P^{-1}_{k-N}} + \sum_{i=k-N}^{k-1} \frac{1}{2}\|\hat{w}_i\|^2_{Q^{-1}} + \sum_{i=k-N}^{k} \frac{1}{2}\|y_i - C\hat{x}_i\|^2_{R^{-1}} \tag{2.15}$$

with the constraints

$$\begin{aligned} x_{k+1} &= Ax_k + B_u k \\ x_k &\in \mathbb{X}, \quad \mathbb{X} = [x_{lb}, x_{ub}]; \\ w_k &\in \mathbb{W}, \quad \mathbb{W} = [w_{lb}, w_{ub}]; \\ y_k - Cx_k &\in \mathbb{V}, \quad \mathbb{V} = [v_{lb}, v_{ub}]; \end{aligned}$$

After a little algebra we obtain the H matrix in the form:

$$H = \begin{bmatrix} C^T R^{-1} C + P^{-1} & 0 & 0 & 0 & 0 & \dots \\ 0 & Q^{-1} & 0 & 0 & 0 & \dots \\ 0 & 0 & C^T R^{-1} C & 0 & 0 & \dots \\ 0 & 0 & 0 & Q^{-1} & 0 & \dots \\ 0 & 0 & 0 & 0 & C^T R^{-1} C & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.16}$$

$$\mathcal{A} = \begin{bmatrix} -A & -I & I & 0 & 0 & & & \\ 0 & 0 & -A & -I & I & & & \\ & & & & \ddots & & & \\ & & & & -A & -I & I & 0 \end{bmatrix}$$

$$\mathcal{C} = \begin{bmatrix} -C & 0 & 0 & 0 & & & \\ C & 0 & 0 & 0 & & & \\ 0 & 0 & -C & 0 & & & \\ 0 & 0 & C & 0 & & & \\ & & & \ddots & & & \\ & & & & 0 & -C & 0 \\ & & & & 0 & C & 0 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} Bu_{k-N} \\ Bu_{k-N+1} \\ \vdots \\ Bu_{k-1} \end{bmatrix} \qquad \mathbf{d} = \begin{bmatrix} v_{ub} - y_{k-N} \\ -v_{lb} + y_{k-N} \\ v_{ub} - y_{k-N+1} \\ -v_{lb} + y_{k-N+1} \\ \vdots \\ v_{ub} - y_{k-1} \\ -v_{lb} + y_{k-1} \end{bmatrix}$$

2.3.2 Only noises as optimization variables

Another way is to express the problem only as a function of the first state estimate $\hat{x}_{k-N}$ and the noises $\mathbf{w}$:

$$\hat{\mathbf{z}} = (\hat{x}_{k-N}, \hat{w}_{k-N}, \hat{w}_{k-N+1}, ..., \hat{w}_{k-1}).$$

The matrices construction in this case is more complex so we introduce some transformation matrices like in [24]. In order to use only the noises we take advantage of the

system state space equation, therefore we do not have equality constraints:

$$\min_{\hat{\mathbf{z}}_c} \frac{1}{2}\hat{\mathbf{z}}_c^T H \hat{\mathbf{z}}_c + \hat{\mathbf{z}_c}^T f \qquad \text{such that:} \quad C_c\hat{\mathbf{z}}_c \leq b_c \tag{2.17}$$

in which $\tilde{\mathbf{z}} = (\hat{\mathbf{z}}_c; \hat{\mathbf{z}}_{rem})$ where $\hat{\mathbf{z}}_{rem}$ contains the "discarded" states variables. We define the transformation matrix Π:

$$\tilde{\mathbf{z}} = \Pi\hat{\mathbf{z}} = \begin{bmatrix}\Pi_1 \\ \Pi_2\end{bmatrix}$$
$$\hat{\mathbf{z}}_c = \Pi\hat{\mathbf{z}}, \hat{\mathbf{z}}_{rem} = \Pi\hat{\mathbf{z}}.$$

Let H^* be equal to 2.16, we can transform the problem:

$$\min_{\hat{\mathbf{z}}_c} \frac{1}{2}\left(\Pi^{-1}\begin{bmatrix}\hat{\mathbf{z}}_c \\ \hat{\mathbf{z}}_{rem}\end{bmatrix}\right)^T H^* \left(\Pi^{-1}\begin{bmatrix}\hat{\mathbf{z}}_c \\ \hat{\mathbf{z}}_{rem}\end{bmatrix}\right) + \left[\left(\Pi^{-1}\begin{bmatrix}\hat{\mathbf{z}}_c \\ \hat{\mathbf{z}}_{rem}\end{bmatrix}\right)^T f.\right] \tag{2.18}$$

Let us define:

$$(\Pi^{-1})^T H^* \Pi^{-1} = \begin{bmatrix}H_{11} & H_{12} \\ H_{21} & H_{22}\end{bmatrix}$$

and

$$(\Pi^{-1})^T f = \begin{bmatrix}f_1 \\ f_2\end{bmatrix}$$

After a little algebra we obtain:

$$H_c = H_{11} + M_1^T H_{22} M_1 + H_{12}M_1 + M_1^T H_{12}^T \tag{2.19}$$
$$f_c = M_1^T H_{12} m_1 + H_{12} m_1 + f_1 + M_1^T f_2. \tag{2.20}$$

Let be $\mathcal{C}^*$ like in the problem 2.16 so if we define $\mathcal{C}^*\Pi = [A_1 A_2]$ we can transform the inequality constraints like:

$$C_c = A_1 + A_2 M1$$
$$b_c = b_{in} - A_2 m_1$$

in which:

$$M_1 = \begin{bmatrix} A & I & 0 & 0 & \dots \\ A^2 & A & I & 0 & \dots \\ A^3 & A^2 & A & I & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ A^N & A^{N-1} & \dots & & \end{bmatrix}$$

$$m_1 = \begin{bmatrix} B & 0 & \dots \\ AB & B & \dots \\ A^2B & AB & \dots \\ \vdots & \vdots & \ddots \\ A^{N-1}B & A^{N-2}B & \end{bmatrix} \begin{bmatrix} u_{k-N} \\ u_{k-N+1} \\ u_{k-N+2} \\ \vdots \\ u_{k-1} \end{bmatrix}$$

$$\Pi_1 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & I & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & I & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & I & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad \Pi_2 = \begin{bmatrix} 0 & 0 & I & 0 & 0 & 0 & 0\dots \\ 0 & 0 & 0 & 0 & I & 0 & 0\dots \\ 0 & 0 & 0 & 0 & 0 & 0 & I\dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

2.3.3 Only states as optimization variables

A different way to approach the problem is tu use the system state space equation in order to hide the errors on the state so as to have only states as optimization variables:

$$J = \sum_{i=k-N}^{k-1} \frac{1}{2}\|\hat{x}_{i+1} - A\hat{x}_i - Bu_i\|^2_{Q^{-1}} + \sum_{i=k-N}^{k} \frac{1}{2}\|y_i - C\hat{x}_i\|^2_{R^{-1}} + \frac{1}{2}\|\tilde{x}_{k-N} - \hat{x}_{k-N}\|^2_{P^{-1}_{k-N}} \tag{2.21}$$

where we denoted

$$\hat{\mathbf{x}} = (\hat{x}^T_{k-N|k}, ..., \hat{x}^T_{k|k})^T \tag{2.22}$$

and the notation $\hat{(\cdot)}_{k-N|k}$ indicates the state estimation at the time step $k-N$ computed at the time step k.

Since we used this approach in our algorithm we now go into the details on how to construct the matrix H.

First term

Let us have a closer look at how to construct the problem 5.7 working out the first term of Eq.5.6 for the first time step ($i = 1$). For matter of simplicity $\hat{x}_i = x_i$:

$$\begin{aligned} \|\hat{x}_1 - A\hat{x}_0 - Bu_0\|^2_{Q^{-1}} &= x_1^T Q^{-1} x_1 + (Ax_0 + Bu_0)^T Q^{-1}(Ax_0 + Bu_0) + \\ &- (Ax_0 + Bu_0)^T Q^{-1} x_1 - x_1^T Q^{-1}(Ax_0 + Bu_0) \\ &= x_1^T Q^{-1} x_1 + x_0^T A^T Q^{-1} A x_0 + u_0^T B^T Q^{-1} B u_0 + \\ &+ u_0^T B^T Q^{-1} A x_0 + x_0^T A^T Q^{-1} B u_0 - x_0^T A^T Q^{-1} x_1 + \\ &- u_0^T B^T Q^{-1} x_1 - x_1^T Q^{-1} A x_0 - x_1^T Q^{-1} B u_0 \end{aligned} \tag{2.23}$$

Now let us collect the all the quadratic parts in x_i of this term in a matrix M_1, we obtain:

$$M_1^{(1)} = \begin{bmatrix} A^TQ^{-1}A & -A^TQ^{-1} \\ -Q^{-1}A & Q^{-1} \end{bmatrix}.$$

Where the (1) indicates the first horizon step. For the second horizon step, we proceed like in 2.23 and update the matrix M_1 as follows:

$$M_1^{(2)} = \begin{bmatrix} A^TQ^{-1}A & -A^TQ^{-1} & 0 \\ -Q^{-1}A & Q^{-1} + A^TQ^{-1}A & -A^TQ^{-1} \\ 0 & -Q^{-1}A & Q^{-1} \end{bmatrix}$$

Notice that the block in position $(2, 2)$ has changed. Proceeding with the $i-th$ horizon step:

$$M_1^{(i)} = \begin{bmatrix} A^TQ^{-1}A & -A^TQ^{-1} & 0 & \dots & 0 \\ -Q^{-1}A & Q^{-1} + A^TQ^{-1}A & -A^TQ^{-1} & \dots & 0 \\ 0 & -Q^{-1}A & Q^{-1} + A^TQ^{-1}A & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & -A^TQ^{-1} \\ 0 & 0 & 0 & -Q^{-1}A & Q^{-1} \end{bmatrix}$$

Second term

Now we solve the second term of Eq.5.7 for the $i-th$

$$\begin{aligned} \|y_i - C\hat{x}_i\|^2_{R^{-1}} &= y_i^TR^{-1}y_i + x_i^TC^TR^{-1}Cx_i - y_i^TR^{-1}Cx_i - x_i^TC^TR^{-1}y_i \\ &= x_i^TC^TR^{-1}Cx_i - 2x_i^TC^TR^{-1}y_i \end{aligned} \tag{2.24}$$

and we put the quadratic parts in x_i in the matrix $M_2^{(i)}$

$$M_2^{(i)} = \begin{bmatrix} C^TR^{-1}C & 0 & \dots & 0 & 0 \\ 0 & C^TR^{-1}C & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & 0 & C^TR^{-1}C & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Third term

Finally

$$\|\tilde{x}_i - \hat{x}_i\|^2_{P_i^{-1}} = \tilde{x}_i^TP_0^{-1}\tilde{x}_i + x_i^TP_i^{-1}x_i - 2\tilde{x}_i^TP_i^{-1}x_i \tag{2.25}$$

from which we form the M_3^i matrix collecting its quadratic parts on x_i

$$M_3^{(i)} = \begin{bmatrix} P_0^{-1} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Now we are ready to obtain the H matrix of Eq.5.7 for the $i-th$ horizon step summing $M_1^{(i)}, M_2^{(i)}$ and $M_3^{(i)}$. Notice that H is symmetric definite positive.

$$H = \begin{bmatrix} A^TQ^{-1}A + C^TRC + P_{k-N}^{-1} & -A^TQ^{-1} & 0 & \dots & 0 \\ -Q^{-1}A & Q^{-1} + A^TQ^{-1}A + C^TRC & -A^TQ^{-1} & \dots & 0 \\ 0 & -Q^{-1}A & Q^{-1} + A^TQ^{-1}A + C^TRC & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & -A^TQ^{-1} \\ 0 & 0 & 0 & 0 & Q^{-1} \end{bmatrix} \tag{2.26}$$

Now we carry on solving only the linear terms in x so to get the $f(\tilde{x}_{k-N}, \mathbf{y}, u)$ of Eq. 5.7. Conversely we do not need the constant term $g(\tilde{x}_{k-N}, \mathbf{y}, u)$ since we are interested only on the optimal point.

First let us collect all the terms depending on the input and state, notice that all of them come from Eq.2.23. For the first horizon step they are: $u_0^TB^TQ^{-1}Ax_0$, $-x_1^TQ^{-1}Bu_0$, as well for the $i-th$: $u_i^TB^TQ^{-1}Ax_i$, $-x_{i+1}^TQ^{-1}Bu_i$, therefore we obtain the matrix:

$$F^{(i)} = \begin{bmatrix} A^TQ^{-1}B & \dots & 0 \\ -Q^{-1}B & \ddots & 0 \\ \vdots & \ddots & A^TQ^{-1}B \\ 0 & 0 & -Q^{-1}B \end{bmatrix}$$

Then we collect the terms coming from Eq.2.23 and Eq.2.25 that depend on the state only and we obtain, for the $i-thstep$

$$\tilde{f}^{(i)} = \begin{bmatrix} -C^TR^{-1}y_0 - P_0^{-1}\tilde{x}_{k-N} \\ -C^TR^{-1}y_1 \\ \vdots \\ -C^TR^{-1}y_i \\ \mathbf{0} \end{bmatrix}$$

where $\mathbf{0}$ is a vector of zero elements in $\mathbb{R}^n$. We obtain $f^{(i)}(\tilde{x}_{k-N}, \mathbf{y}, u) = \tilde{f}^{(i)} + F^{(i)}u^{(i)}$, where $u^{(i)}$ is the vector obtained stacking along a column the input vectors

$\{u_1, u_2, ..., u_i\}$. The same results can be obtained with the same matrix transformation as in Eq. 2.19 where:

$$M_1 = \begin{bmatrix} -A & I & 0 & 0 & \dots \\ 0 & -A & I & 0 & \dots \\ & & \ddots & \ddots & \end{bmatrix}$$

$$m_1 = \begin{bmatrix} -B & 0 & 0 & \dots \\ 0 & -B & 0 & \dots \\ & & \ddots & \end{bmatrix} \cdot \begin{bmatrix} u_{k-N} \\ u_{k-N+1} \\ \vdots \\ u_{k-1} \end{bmatrix}$$

$$\Pi_1 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & \\ 0 & 0 & I & 0 & 0 & \\ 0 & 0 & 0 & 0 & I & \\ & & & & & \ddots \end{bmatrix} \Pi_2 = \begin{bmatrix} 0 & I & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & I & 0 & 0 & \\ 0 & 0 & 0 & 0 & 0 & I & \\ & & & & & & \ddots \end{bmatrix}.$$

Chapter 3

Theoretical background on optimization problems

The main issue that obstructs the application of the MHE for closed loop controlled systems is the time required for solving the optimization problem. In order to understand the principles on which most of the standard optimization algorithms are grounded, we want to discuss about the theory behind the concept of optimization.

3.1 Optimization problem

The concept of optimization has been revealed to be a tool of huge importance and great impact in the engineering field, as well as in other discipline, but it was also demonstrated to be an essential factor in nature. Several semi-heuristic algorithms inspired to some nature systems have been recently developed, like the *Ant colony optimization* [25], or *Artificial bee colony algorithm* [26].

Optimization founds significant applications on one of the most important topic of the recent years: energy saving or more generally *green engineering*. We can optimize the consumption of fuel in vehicles, the power supplied to the heating system of an apartment, the usage of some dangerous reactants in a chemical reaction or the formation of a toxic by-product. These are only few examples of the thousand other applications of optimization strategies.

In the MHE problem, the optimization problem comes as a way to obtain the most probable system states once we know the output of the system, a probable initial point and an idea of noises magnitude that effect the system. In the next sections we are going to have a closer look at the mathematics that is behind the optimization algorithms.

3.2 Optimization sets

The most general formulation of an optimization problem is the following:

$$\min_x f(x) \qquad s.t. \begin{cases} h(x) \leq 0 \\ g(x) = 0 \end{cases} \tag{3.1}$$

in which $f \in \mathbb{R}^n \to \mathbb{R}$ is called *objective function*, $x \in \mathbb{R}^n$ represents the optimization variable, $h(x) \in \mathbb{R}^d$ is the vector of inequality constrains and $g(x) \in \mathbb{R}^l$ is the vector of equality constraints.

The first question that arises is: *can we found a solution to this problem and, if yes, is that solution unique?* In the successive sections we are going to review the properties that a well-posed problem must have, in order to be able to compute a solution, and in which case the solution is unique.

3.2.1 Affine and convex set

Definition 3.1 (Affine Set)**.** A set $C \subseteq \mathbb{R}^n$ is *affine* if the line through any two distinct points in C *lies* in C. Namely, if for any $x_1, x_2 \in C$ and $\theta \in \mathbb{R}$ we have $\theta x_1 + (1-\theta)x_2 \in C$ in which $\theta \in [0, 1]$. We can generalize this idea for more than two points and obtain the *affine combination*, that is $\sum \theta_i x_i$ where $\sum \theta_i = 1$.

Definition 3.2 (Convex Set)**.** A set $C \subseteq \mathbb{R}^n$ is *convex*, if every *convex combination* of points $x_i \in C$, defined as $\sum \theta_i x_i$ is contained in C, where $\theta_i \geq 0$ and $\sum \theta_i = 0$.

Notice that the only difference respect to an affine set, is that the convex set requires that $\theta_i \geq 0$. We can easily see that an affine set is also a convex set, but in general the converse is not true. For example:

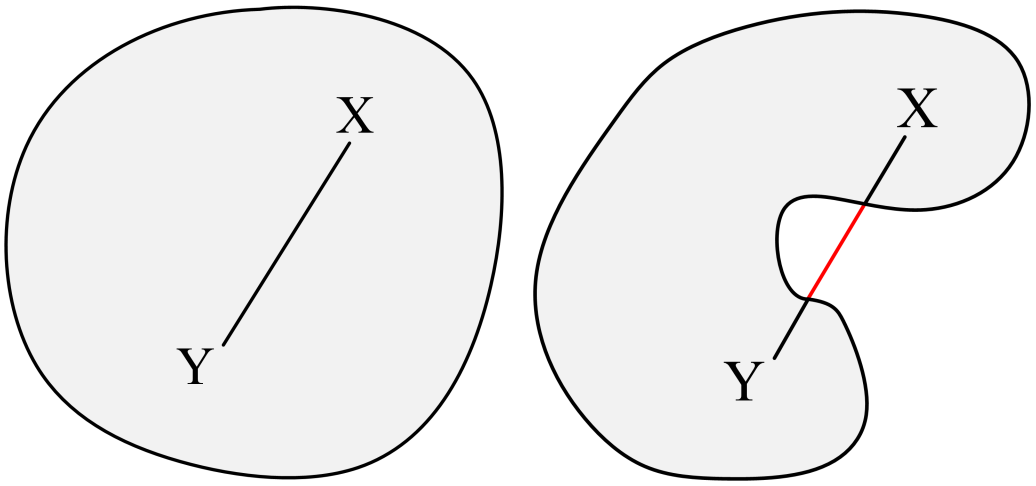
- The empty set $\varnothing$, any single point, and the whole space $\mathbb{R}^n$ are affine, hence convex, subsets of $\mathbb{R}^n$.
- Any line is affine. if it passes through zero.

Definition 3.3 (Polyhedra)**.** A *polyhedron* is defined as the solution of a finite number of linear equalities and inequalities:

$$P = \{x | a_j^T x \leq b_j, \quad j = 1, ..., m, \quad c_j^T = d_j, \quad j = 1, ...p\} \tag{3.2}$$

A polyhedron is the intersection of a finite number of halfspace and hyperplane. Affine sets (e.g. subspaces, hyperplanes, lines) and convex sets (line segments, halfspace) are all polyhedra. We can also use the more compact notation:

$$P = \{x | Ax \leq b, Cx = b\} \tag{3.3}$$



(A) *Convex set.* (B) *Non-convex set.*

FIGURE 3.1: Examples of convex and non-convex set.

3.2.2 Operation that preserve convexity

We are going to describe some operation that preserve convexity of sets, so as to allow us to construct convex sets from others.

Intersection

Convexity is preserved under *intersection*. If S_1 and S_2 are two convex sets, then $S_1 \cap S_2$ is convex. This property extends to the intersection of an infinite number of sets, e.g.: a polyhedron is the intersection of halfspaces and hyperplanes, (which are convex, and therefore is convex).

Affine transformation

If $f : \mathbb{R}^n \to \mathbb{R}^m$ is affine (see Def. 3.4), and $S \subseteq \mathbb{R}^n$ is convex, then the image of S under f, $f(S) = \{f(x)|x \in S\}$ is convex. Similarly, if $f : \mathbb{R}^k \to \mathbb{R}^n$ is affine, the *inverse image* of S under f, $f^{-1}(S) = \{f(x)|x \in S\}$ is convex.)

Some examples:

- *Scaling and translation.*If $S \subseteq \mathbb{R}^n$ is convex $\alpha \in \mathbb{R}^n$, and $a \in \mathbb{R}$, the sets αS and $S + a$ are convex.
- *Sum.* The sum of two convex set is convex. $S_1 + S_1 = \{x + y|x \in S_1, y \in S_2\}$.
- *Cartesian product.* The cartesian product of two convex sets is convex. $S_1 \times S_2 = \{(x_1, x_2) \quad | \quad x_1 \in S_1, x_2 \in S_2\}$.

3.3 Optimization functions

3.3.1 Affine functions

Definition 3.4 (Affine function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is *affine* if it is a sum of a linear function and a constant, i.e., if it has the form $f(x) = Ax + b$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

3.3.2 Convex functions

Definition 3.5 (Convex function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* in its $\mathbf{dom} f$ is a *convex* set and if for all $x, y \in \mathbf{dom} f$, and θ with $0 \leq \theta \leq 1$, we have:

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y) \tag{3.4}$$

Geometrically, this means that the line segment between $(x, f(x))$ and $(y, f(y))$ lies above the graph of f (see Fig. 3.2). A function f is *strictly convex* if struct inequality holds in Eq. 3.4. Furthermore, we say that f is *concave* in $-f$ is convex, and *strictly concave* if $-f$ is strictly convex.

For an affine function we have always equality in Eq. 3.4, so every affine function (hence also linear) are both convex and concave. Conversely, any function that is both concave and convex is affine.

3.3.2.1 First order conditions

Suppose f is differentiable (i.e., its gradient ∇f exists at each point in $\mathbf{dom} f$), then f is convex if and only if $\mathbf{dom} f$ is convex and:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \qquad \text{for all} \quad x, y \in \mathbf{dom} f \tag{3.5}$$

The affine function in y given by Eq. 3.5 is the first-order Taylor approximation of f near x. This shows that from *local information* about a convex function (i.e, its value and derivative at a point) we can derive *global information*.

Strict convexity can also be characterized by a first-order condition, just in Eq. 3.5 we have $>$ instead of $\geq$.

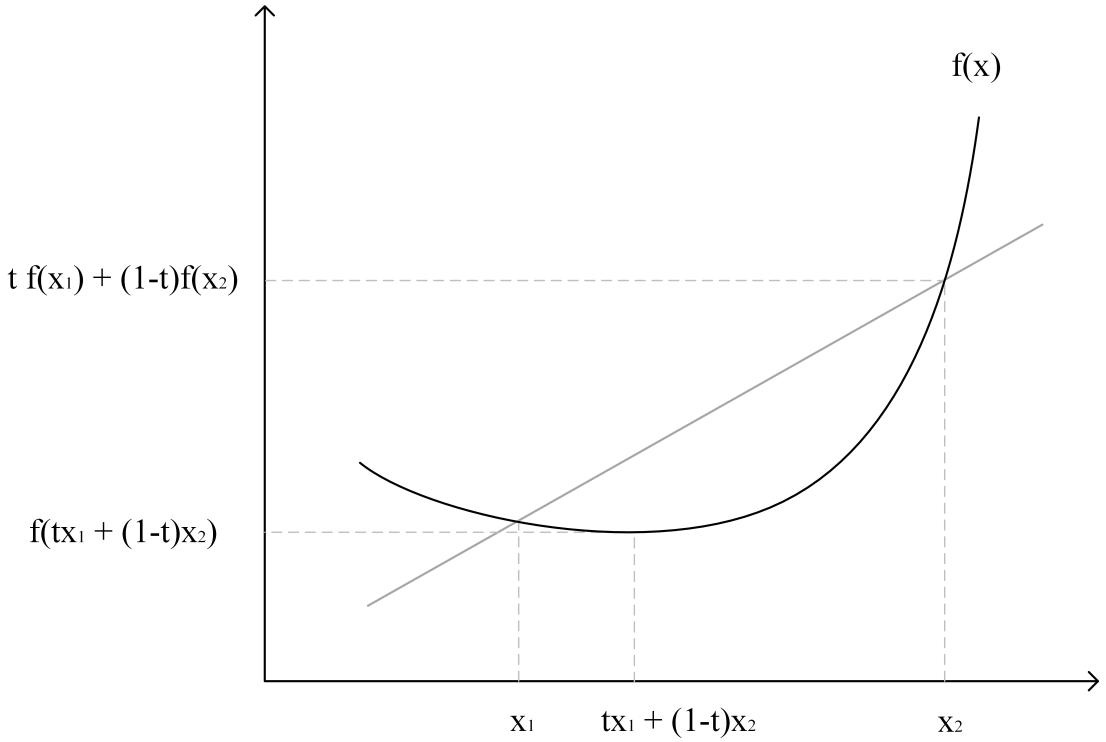


FIGURE 3.2: Graph of a convex function. Notice that the line between two points it is always entirely above the function evaluated between those two points.

3.3.2.2 Second order condition

Definition 3.6. A function f is convex if and only if $\mathbf{dom}f$ is convex and its Hessian is positive semi-definite, that is for all $x \in \mathbf{dom}f$:

$$\nabla^2 f(x) \geq 0 \tag{3.6}$$

Strict convexity can be partially characterized saying that if $\nabla^2 f(x) > 0$ for all $x \in \mathbf{dom}f$ then f is strictly convex, *but the converse is not true*, for example the function $f(x) = x^4$ is strictly convex but has zero second derivative at $x = 0$.

Examples

An example of convex function are the *quadratic functions.*

Definition 3.7 (Quadratic functions)**.** A quadratic function $f : \mathbb{R}^n \to \mathbb{R}$ with $\mathbf{dom}f = \mathbb{R}^n$ is given by:

$$f(x) = \frac{1}{2}x^T Px + q^T x + r \tag{3.7}$$

with $P \in \mathbb{R}^{nxn}$,$q \in \mathbb{R}^n$ and $r \in \mathbb{R}$. The second order condition states that this function is convex if $P \geq 0$, is strictly convex if $P > 0$, while is concave if $P \leq 0$ and strictly concave if $P < 0$.

Other examples of convex function on $\mathbb{R}$:

- *Exponential.* e^{ax} is convex on $\mathbb{R}$ for any $a \in \mathbb{R}$;
- *Powers.* x^a is convex on $\mathbb{R}^+$ when $a \geq 1$ or $a \leq 0$, and concave for $0 \geq a \geq 1$;
- *Powers of absolute value.* $|x|^p$, for $p \geq 1$, is convex in $\mathbb{R}$;
- *Logarithm.* $\log x$ is concave on $\mathbb{R}^+$

Then some example of convex function in $\mathbb{R}^n$:

- *Norms.* Every norm on $\mathbb{R}^n$ is convex;
- *Max function.* $f(x) = max(x_1, ..., x_n)$ is convex in $\mathbb{R}^n$;

3.3.3 Operation that preserve convexity

- *Multiplication by scalars.* If f is a convex function and $\alpha \geq 0$, αf is convex;
- *Nonnegative weighted sum.*If f_i are a convex function, $f = w_1 f_1 + ... + w_m f_m$ is convex;

In optimal control, as well as in estimation problem, the function of which we want find the minimum is a sum of convex functions:

$$\min_{u,x} \sum_{i=0}^{N-1} L_i(x_i, u_i) + E(x_N) \qquad s.t. \begin{cases} x_0 - \bar{x}_0 & = 0, \quad i = 0, ..., N-1 \\ x_{i+1} - f_i(x_i, u_i) & = 0, \quad i = 0, ..., N-1 \\ h_i(x_i, u_i) & \leq 0, \quad i = 0, ..., N-1 \\ g_i(x_i, u_i) & = 0, \quad i = 0, ..., N-1 \\ r(x_r) & \leq 0. \end{cases} \tag{3.8}$$

In our case L_i is going to be a quadratic function (Def. 3.7) with $P > 0$ and a set of convex constraints, in particular polyhedra (see Def. 3.3) formed by the intersection of hyperplanes.

3.3.4 Optimality

Definition 3.8 (Optimal value.)**.** p^* is an *optimal value* of the problem 3.1 if:

$$p^* = \inf \{f(x) | g(x) \leq 0 \quad \text{for} \quad i = 1, ..., l \quad \text{and} \quad c(x) = 0 \quad \text{for} \quad i = 1, ..., p\}$$

Definition 3.9 (Locally optimal point.)**.** A feasible point x is *locally optimal* if there is a $R > 0$ such that

$$f_0(x) = \inf \{f(x) | h(x) \leq 0 \quad \text{for} \quad i = 1, ..., l \quad \text{and} \quad g(x) = 0 \quad \text{for} \quad i = 1, ..., p, \|z - x\|_2 \leq R\} \tag{3.9}$$

in other words, x is optimal point of a *neighbourhood* of *radius* R.

Definition 3.10 (Optimal point.)**.** We say that x^* is an *optimal point* or equivalently we say that solve the problem 3.1, if $x^* \in \mathbf{dom} f$ (it is *feasible*), and $f(x^*) = p^*$. The *set of optimal points* is given by :

$$X_{opt} = \{f(x) | h(x) \leq 0 \quad \text{for} \quad i = 1, ..., l \quad \text{and} \quad g(x) = 0 \quad \text{for} \quad i = 1, ..., p, \quad f(x^*) = p^*\}$$

3.4 Convex optimization

A convex optimization problem is one of the form:

$$\min_x f(x) \qquad \text{s.t.} \begin{cases} g(x) & \leq 0 \\ Ax & = b \end{cases} \tag{3.10}$$

where $f(x)$ is a convex function, $g(x)$ is a vector convex functions and $A \in \mathbb{R}^{n \times n}$. Comparing this form with the problem 3.1 the convex problem has three additional requirements:

- the objective function must be convex;
- the inequality constraint functions must be convex;
- the equality constraint functions must be affine.

Notice that the feasible set of a convex optimization problem is convex since each $g_i(x)$ is convex and $a_i^T x = b$ is convex (it's an hyperplane), and we have seen in § 3.2.2 that the intersection of convex set is convex. Thus, in a convex optimization problem, we minimize a convex objective function over a convex set.

Theorem 3.11 (Solution of a strictly convex optimization problem.)**.** *The problem written in Eq. 3.10 has a unique solution if and only if the objective function is strictly convex.*

3.4.1 Local and global optima

Theorem 3.12. *If the convex problem 3.1, has a locally optimal point, that point is also globally optimal.*

Proof. To see this, suppose that x is a locally optimal point for the convex problem, i.e. x is feasible and:

$$f(x) = \inf\{f(z) \quad | \quad z \quad \text{feasible}, \quad \|z - x\|_2 \leq R\}$$

for some $R > 0$. Suppose that x is *not* globally optimal, thus there is a feasible y such that $f(y) < f(x)$. Evidently $\|y - x\|_2 > R$, since otherwise $f(x) \geq f(y)$. Now if we consider a point

$$z = (1 - \theta)x + \theta y, \qquad \theta = \frac{R}{2\|y - x\|_2}.$$

We have $\|z - x\|_2 = R/2$, where x is feasible by convexity of the feasible set. By convexity of f we can say

$$f(z) \leq (1 - \theta)f(x) + \theta f(y) \leq f(x)$$

which contradicts 3.9, therefore there exists no feasible y with $f_0(y) < f_0(x)$ i.e., x is globally optimal. □

3.4.2 Optimal criterion for differentiable f

The problem now is how to built a criterion that allows us to practically know if a point x is optimal. Suppose that the objective function f of a convex optimization problem is differentiable, so that:

$$f_0(y) > f_0(x) + \nabla f_0(x)^T(y - x) \tag{3.11}$$

Let us denote as X the feasible set, namely:

$$X = \{x | g(x) \leq 0, i = 1, ..., m, \quad h(x) = 0, \quad i = 1, ..., p\}$$

Then the x is optimal if and only if $x \in X$ and

$$\nabla f_0(x)^T(y - x) \leq 0 \qquad \text{for all} \quad y \in X \tag{3.12}$$

this can be seen graphically in Fig. 3.3, and it means that the vector $-\nabla f(x)$,that indicate the direction towards the function decreases, and the vector $(y - x)$ form an acute angle.

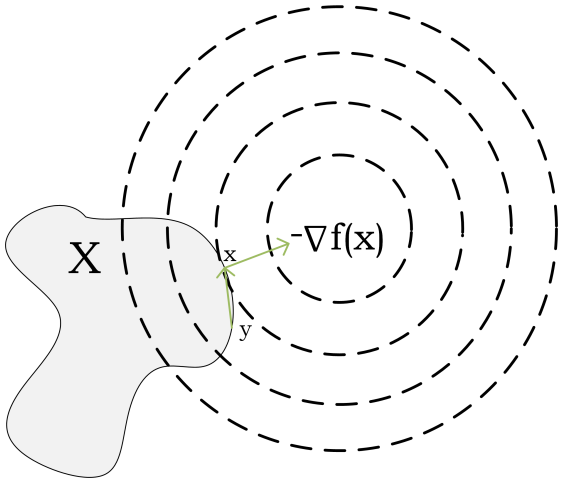


FIGURE 3.3: Depiction of Eq.3.12, notice that the angle between the vectors is acute. Since the x is the optimal point, it can be easily seen that taking any y point in X, the angle is always acute.

3.4.3 Unconstrained problem

For an unconstrained problem, the condition 3.12 reduces to:

$$\nabla f(x) = 0 \tag{3.13}$$

If there are no solution to Eq.3.13 there are no optimal points. Here we can distinguish two cases: the problem is *unbounded below* or the optimal value is finite but not reachable. In the other hand we can have more than one solution, in which case each solution is a minimizer of f.

Example - *Unconstrained quadratic optimization*

Consider the quadratic problem:

$$f(x) = \frac{1}{2}x^T Px + q^T x + r$$

where P is symmetric semidefinite positive, the necessary sufficient condition for x to be minimizer of f is:

$$\nabla f(x) = Px + q^T = 0$$

Now we can have several cases, depending on whether this linear equation has no solution, one solution or many solutions.

- If $q \notin \mathcal{R}(P)$, then there is no solution, in this case f is unbounded below.

- If $P > 0$ namely is f is strictly convex, then there is a unique minimizer, $x^* = -P^{-1}q$.
- If P is singular, but $q \in \mathcal{R}(P)$ then the (affine) set of optimal points is $X_{opt} = -P^{\dagger}q + \mathcal{N}P$ where $P^{\dagger}$ denotes the pseudo-inverse of P (see A.2).

3.4.4 Problem with only equality constraint

Consider the case when we have equality constraints but no inequality constrains:

$$\min_x f(x) \qquad \text{s.t.} \quad Ax = b$$

The feasible set is affine (it is again an intersection between hyperplanes, (namely it is a polyhedron) and we assume that it is no empty, otherwise the problem is infeasible. The optimality condition is that:

$$\nabla f(x)^T (y - x) \geq 0$$

must hold for all y satisfying $Ay = b$. Since x is feasible, every feasible y has the form $y = x + v$ where $v \in \mathcal{N}(A)$, therefore we can express the optimality condition as :

$$\nabla f(x)^T v \geq 0 \qquad \text{for all } v \in \mathcal{N}(A)$$

But we know that if a linear function is nonnegative on a subspace, then must be zero on all the subspace, so it follows that

$$\nabla f(x)^T v = 0 \quad \rightarrow \quad \nabla f(x) \perp \mathcal{N}(A).$$

We know also that $\mathcal{N}(A)^{\perp} = \mathcal{R}(A^T)$, that means $\nabla f(x) \in \mathcal{R}(A^T)$, i.e. there exist a $v \in \mathbb{R}^p$ such that:

$$\nabla f(x) + A^T v = 0$$

Together with the $Ax = b$ (the x has to be feasible) this is the classical Lagrange multiplier optimality condition, that we are going to cover later in this work.

3.5 Linear optimization problems

Definition 3.13 (Linear program)**.** When the objects and constraint function are all affine, the problem is called *linear program* (LP). Generally it has the form:

$$\min_x c^T + d \qquad \text{s.t.} \quad \begin{cases} Gx & \leq h \\ Ax & = b \end{cases} \tag{3.14}$$

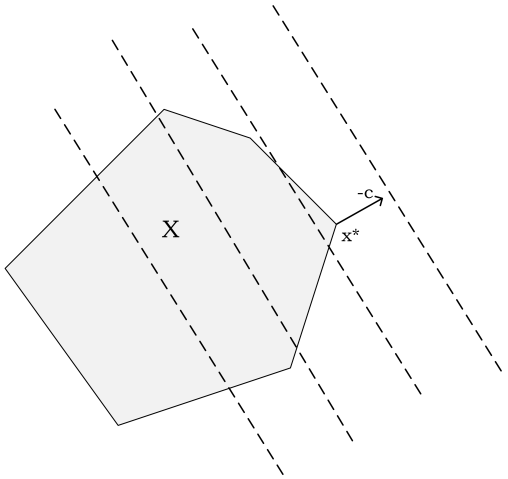


FIGURE 3.4: This is a geometric interpretation of a LP, similar to the generic one that we gave previously. In this case notice that the optimal point is going to be *always* in the boundary of the set X

where $G \in \mathbb{R}^{m \times n}$ and $A \in \mathbb{R}^{p \times b}$. The problem, naturally, is convex.

It is common to omit the constant d, since it does not affect the optimal point. A figure representative of the problem is Fig. 3.4.

Definition 3.14 (Standard LP formulation)**.** We define *standard LP formulation* or *standard LP form* the linear problem in the form:

$$\min_x c^T x \qquad \text{s.t.} \quad \begin{cases} Ax & = b \\ x & \leq 0 \end{cases} \tag{3.15}$$

Notice that the Eq. 3.14 can be always written in the standard form.

3.6 Quadratic optimization problem

Definition 3.15 (Quadratic problem)**.** The convex optimization problem 3.10 is called *quadratic program* (QP) if the objective function is (convex) quadratic and the constrain functions are affine.

$$\min_x \frac{1}{2} x^T P x + q^T x + r \qquad \text{s.t.} \quad \begin{cases} Gx & \leq h \\ Ax & = b \end{cases} \tag{3.16}$$

where $P \geq 0$, $G \in \mathbb{R}^{m \times n}$, and $A \in \mathbb{R}^{p \times n}$.

Definition 3.16 (Quadratically constrained quadratic program)**.** If not only the objective function, but also the inequality constraints are quadratic, we call this problem *quadratically constrained quadratic function*

$$\min_x \frac{1}{2}x^T Px + q^T x + r \qquad \text{s.t.} \quad \begin{cases} \frac{1}{2}x^T Gx + h^T x + l \leq 0 \\ Ax = b \end{cases} \tag{3.17}$$

3.7 Duality

We consider a standard form of the optimization problem as in 3.1;

$$\min_x f(x) \qquad \text{s.t.} \quad \begin{cases} g(x) & \leq 0 \\ h(x) & = 0 \end{cases} \tag{3.18}$$

Let us assume the domain $\mathcal{D} = \mathbf{dom}(g) \cap \mathbf{dom}(h)$ is nonempty, and let be p^* the optimal value.

We can reformulate the problem 3.18 in a *dual* representation, called *dual problem*. The solution to the dual problem provides a lower bound to the solution of the primal (minimization) problem, and under certain assumption, we are going to see how these two results match. There exist many dual problems, however the most famous is the *Lagrangian duality*, indeed we are going to refer only to this case.

The basic idea is to modify the objective function adding the sum of the constraints function. We define the *Lagrangian* $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$:

$$L(x, \lambda, v) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{i=1}^{p} v_i h_i(x) \tag{3.19}$$

where $\mathbf{dom}(L) = \mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$. We refer to λ_i as the *Lagrange multiplier* associated with the ith inequality constraints, and v_i as the Lagrange multiplier associated with the equality constraints. The vectors λ and v are called *Lagrange multiplier vectors*.

Definition 3.17 (Lagrange dual function)**.** The *Lagrange dual function* or *dual function* $g \in \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ as the minumum of the Lagrangian over x:

$$G(\lambda, v) = \inf_{x \in \mathcal{D}} L(x, \lambda, v) = \inf_{x \in \mathcal{D}} \left(f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{i=1}^{p} v_i h_i(x) \right)$$

Notice that since the Langrangian is a family of affine functions of (λ, v), is is *concave, even when the problem 3.18 is not convex.*

Theorem 3.18 (Lower bound of the dual function)**.** *The dual function gives a lower bound of the optimal value p^* of the problem 3.18 for any $\lambda \leq 0$, namely*

$$G(\lambda, v) \leq p^*. \tag{3.20}$$

Proof. Suppose $\tilde{x}$ is a feasible point in 3.18, i.e., $g(\tilde{x}) \leq 0, h(\tilde{x}) = 0$ and $\lambda \geq 0$. Then we have :

$$\sum_{i=1}^{m} \lambda_i g(\tilde{x})_i + \sum_{i=1}^{p} v_i h_i(\tilde{x}) \leq 0,$$

since each element of the first sum is negative (remember $\lambda \geq 0$) and each element of the second sum zero, therefore:

$$L(\tilde{x}, \lambda, v) = f(\tilde{x}) + \sum_{i=1}^{m} \lambda_i g_i(\tilde{x}) + \sum_{i=1}^{p} v_i h_i(\tilde{x}) \leq f(\tilde{x}).$$

and finally

$$G(\lambda, v) = \inf_{x \in \mathcal{D}} L(x, \lambda, v) \leq L(\tilde{x}, \lambda, v) \leq f(\tilde{x}).$$

□

The effect of these two terms in the cost function is of increase the *cost* when the constraints are violated. For example, if for some $\tilde{x}, h_i(\tilde{x}) \neq 0$ the value of the dual function increases, the same happen when for some $\tilde{x}, f(\tilde{x}) \geq 0$ (that is why λ has to be always positive) therefore the optimization algorithm will try to avoid that looking for some x that is in the feasible region.

Now we want to find the *best* lower bound of our that we can obtain from the Lagrange function. This correspond to the problem

$$\max_{\lambda, v} G(\lambda, v) \qquad \text{s.t.} \quad \lambda \geq 0. \tag{3.21}$$

We refer to (λ^*, v^*) as the *dual optimal* or *optimal Lagrange multiplier*. The Lagrange dual problem is a convex optimization problem since the objective function is concave and the constraints are convex. This happen whether or not the primal problem 3.1 is convex. The dual function gives a nontrivial lower bound only when $\lambda \leq 0$ and $(\lambda, v) \in \mathbf{dom}(g)$ i.e. $g(\lambda, v) > -\infty$.

Example - Lagrangian function of a LP problem

Consider the LP in the standard form:

$$\min_x c^T x \qquad \text{s.t.} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases} \tag{3.22}$$

Let us form the Lagrangian as seen previously, we obtain:

$$L(x, \lambda, v) = c^T x - \sum_{i=1}^{n} \lambda_i x_i + v^T (Ax - b) = -b^T v + (c + A^T v - \lambda)^T x$$

the corresponding dual function is

$$G(\lambda, v) = \inf_x L(x, \lambda, v) = -b^T v + \inf_x (c + A^T v - \lambda)^T x$$

we notice that the function that we have to minimize is linear, therefore the limit is not $-\infty$ only if $(c + A^T v - \lambda)^T x = 0$. In conclusion:

$$G(\lambda, v) = \begin{cases} -b^T v & \text{when} \quad (A^T v - \lambda + c)^T x = 0 \\ -\infty & \text{otherwise} \end{cases}$$

3.8 Optimal condition

If we can find a dual feasible (λ, v) we obtain a lower bound on the optimal value of the primal problem, namely $p^* \geq g(\lambda, v)$. If x is feasible for the primal problem (*primal feasible*):

$$f(x) - p^* \leq f(x) - G(\lambda, v)$$

in particular, we say that x is ϵ-suboptimal, with $\epsilon = f(x) - g(\lambda, v)$. ϵ is called *duality gap*. It gives the gap between the solution of the primal problem and the best solution of the dual problem.

It is possible to define a stopping criteria of an algorithm by choosing a *tollerance* on the duality gap. Suppose an algorithm produces a sequence of primal feasible $x^{(k)}$ and dual feasible $(\lambda^{(k)}, v^{(k)})$ for $k = 1, 2, ...$ We want an absolute accuracy of ϵ_{toll}, then the stopping criterion could be:

$$f(x^{(k)}) - G(\lambda^{(k)}, v^{(k)}) \leq \epsilon_{toll}$$

but in general, is not sure that we can find a solution for arbitrarily small tolerances, because the dual problem could not approach the primal problem near enough. In this case we need to make another assumption, that we explain in the next section.

3.8.1 Strong duality and Slater's constraint qualification

If the duality gap is zero, we say that *strong duality* holds. Generally strong duality does not hold, but if we have a convex problem (Eq. 3.10) and the *Slater's condition* holds, we have also strong duality.

Theorem 3.19 (Slater's condition)**.** *Given a convex problem, if there exist an* $x \in$ ***relint***(D) *such that* $g(x) < 0$ *and* $Ax = b$ *this point is called* strictly feasible *(because the strict inequality holds) and* strong duality *holds.*

If some of the inequality constraints are affine, e.g. $g_1, g_2, ..., g_k$ then we can have a weaker condition for the strong duality. Indeed in this case the x has to be strictly feasible only for the non-affine inequality constraints $g_i(x) < 0 \quad i = k+1, ..., m$.

Now we make some important observations: suppose that the strong duality holds, that means:

$$\begin{aligned} f(x^*) &= G(\lambda^*, v^*) \\ &= \inf \left(f(x) + \sum_{i=1}^{m} \lambda_i^* g_i(x) + \sum_{i=1}^{m} v_i^* h_i(x) \right) \\ &\leq f(x^*) + \sum_{i=1}^{m} \lambda_i^* g_i(x^*) + \sum_{i=1}^{m} v_i^* h_i(x^*) \\ &\leq f(x^*) \end{aligned}$$

The first line is true when the strong duality holds, then in the second line we have insert the definition of the dual function, in the third we use the fact that the point where the minimum for the dual function exists is equal to the point where the minimum of the primal function exists, while the last inequality is due to the fact that $\lambda_i^* \geq 0, g(x^*) < 0, i = 1, ..., m$ and $h_i(x^*) = 0, i = 1, ..., p$. We can assert that:

1. Since the second and third lines are equal x^* minimizes $L(x, \lambda^*, v^*)$ over x (but it could have other minimizers);
2. $\sum_{i=1}^{m} \lambda_i^* g_i(x^*) = 0$, and since each term is nonpositive of course $\lambda_i^* g_i(x^*) = 0, i = 1, ..., m$;

the second assertion is called *complementary slackness*. That means that, in all cases, when strong duality holds, for any primal optimal and any dual optimal :

$$\lambda_i^* > 0 \rightarrow g_i(x^*) = 0$$

or

$$g_i(x^*) < 0 \rightarrow \lambda_i^* = 0$$

3.8.2 Karush-Kuhn-Tucher (KKT) optimality conditions

Let us assume that the functions $f(x), g(x), h(x)$ are differentiable, if $x^*, (\lambda^*, v^*)$ are the primal and dual optimal points with zero duality gap. Since x^* minimize the Lagrangian, it follows that its gradient must be zero at x^*:

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{i=1}^{p} v_i^* \nabla h_i(x^*) = 0$$

therefore we obtain the KKT conditions for non-convex problems:

$$\begin{aligned} g_i(x^*) &\leq 0, \qquad i = 1, ..., m \\ h_i(x^*) &= 0, \qquad i = 1, ..., p \\ \lambda_i^* &\geq 0, \qquad i = 1, ..., m \\ \lambda_i^* g_i(x^*) &= 0, \qquad i = 1, ..., m \\ \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{i=1}^{p} v_i^* \nabla h_i(x^*) &= 0 \end{aligned} \tag{3.23}$$

Lemma 3.20 (KKT condition for non convex problem)**.** *This means that for* any *optimization problem with differentiable objective and constraint function with strong duality, any pair of primal and dual optimal points must satisfy the KKT condition. Notice that vice versa is not always true.*

Lemma 3.21 (KKT condition for convex problem)**.** *For a convex problem, the KKT condition are* necessary and sufficient *to prove that $x^*, (\lambda^*, v^*)$ are the primal and dual optimal points with zero duality gap.*

Proof. Indeed, the first two condition state that x^* is feasible. Furthermore since $\lambda_i \geq 0$, $L(x, \lambda^*, x^*)$ is convex in x and the last condition states that its gradient is zero at x^* it follows that it minimize $L(x, \lambda^*, x)$, so:

$$G(\lambda^*, v^*) = L(x^*, \lambda^*, v^*) = f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{i=1}^{p} v_i^* \nabla h_i(x^*) = f(x^*)$$

□

Chapter 4

Review on optimization algorithms

In Chapter 2 we have shown that the KKT condition must hold if the in a the optimal point of a optimization problem. Along this line, we have shifted our problem :

$$\min_x \ f(x) \qquad \text{s.t.} \quad \begin{cases} g(x) = 0 \\ h(x) \leq 0 \end{cases}$$

where our unknown variable is x to the problem :

$$\begin{aligned} g_i(x^*) &= 0, \qquad i = 1, ..., m \\ h_i(x^*) &\leq 0, \qquad i = 1, ..., p \\ \lambda_i^* &\geq 0, \qquad i = 1, ..., m \\ \lambda_i^* g_i(x^*) &= 0, \qquad i = 1, ..., m \\ \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{i=1}^{p} v_i^* \nabla h_i(x^*) &= 0 \end{aligned} \tag{4.1}$$

where instead we have three unknown variable x^*, λ^* and v_i. Now we are going to see how this approach can lead has to solve the original problem.

4.1 Unconstrained minimization

For understanding the *hierarchy* that an optimization algorithm usually implements, we have to start from the simplest case of all, the unconstrained minimization. We have

$$\min \ f(x) \tag{4.2}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex and twice continuously differentiable (therefore $\mathbf{dom}(f)$ is open). We assume that there exists an optimal x^*, and we denote $p^* = \inf\{f(x)\} = f(x^*)$. Since f is differentiable and convex, a necessary and sufficient condition for a point x^* to be optimal is:

$$\nabla f(x^*) = 0$$

therefore solving this problem is the same as solving the problem 4.2, which is a set of n equation in n variables. In some cases we can find analytically a solution but usually we have to solve it with an iterative algorithm. This algorithm compute a sequence of points $x^{(0)}, x^{(1)}... \in \mathbf{dom}(f)$ with $f(x^{(k)}) \to p^*$ as $k \to \infty$. We call this sequence *minimization sequence*. The algorithm stops when $f(x^{(k)}) - p^* \leq \epsilon$, where $\epsilon > 0$ is a specified tolerance.

The method described require that the starting point $x^{(0)}$ must lie in $\mathbf{dom}(f)$, and the sublevel set defined as follows:

$$\mathcal{S} = \{x \in \mathbf{dom}\ f | f(x) \geq f(x^{(0)}\} \tag{4.3}$$

must be closed. This condition is satisfied for all $x^{(0)} \in \mathbf{dom}(f)$ if the function is closed.

4.1.1 Strong convexity

Definition 4.1 (Strong convex function)**.** A function is *strongly convex* on $\mathcal{S}$ if there exist an $m > 0$ such that

$$\nabla^2 f(x) \geq mI \tag{4.4}$$

for all $x \in \mathcal{S}$.

As a consequence we have that for $x, y \in \mathcal{S}$

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T\nabla^2 f(z)(y - x)$$

where z is some point in the line segment from x to y. So is f is strongly convex:

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{m}{2}\|y - x\|_2^2 \tag{4.5}$$

this gives us a better lower bound that we can use to bound $f(x) - p^*$. Indeed, let us minimize the right side of 4.5 in respect to y, we obtain $\tilde{y} = x - (1/m)\nabla f(x)$. If we substitute:

$$f(y) \geq f(x) - \frac{1}{2m}\|\nabla f(x)\|_2^2$$

therefore we obtain a lower bound for the optimal point:

$$p^* \geq f(x) - \frac{1}{2m}\|\nabla f(x)\|_2^2. \tag{4.6}$$

From this we can deduce that if the gradient is small in a point, that point is nearly optimal. We can make other two observations, we can use this result as a condition for suboptimality:

$$\|\nabla f(x)\|(2m\epsilon)^{1/2} \rightarrow f(x) - p^* \leq \epsilon$$

and we can derive a bound on the distance from the optimal point:

$$\|x - x^*\|_2 \leq \frac{2}{m}\|\nabla f(x)\|_2$$

As well as the lower bound we can derive a upper bound, since that the Eq. 4.5 requires that $\mathcal{S}$ is bounded. Therefore the maximum eigenvalue of the hessian $\nabla^2 f(x)$ is bound above, such that:

$$\nabla^2 f(x) \leq MI \tag{4.7}$$

this implies

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{M}{2}\|y - x\|_2^2 \tag{4.8}$$

therefore

$$p^* \leq f(x) - \frac{1}{2M}\|\nabla f(x)\|_2^2. \tag{4.9}$$

4.1.2 Condition number

The condition number of a function $y = f(x)$ is a parameter that gives us an idea on how much a small variation on the x reflects on the y. If the condition number is high, it means that small variations (usually errors) on the x, cause large variation on the y, while if it is small, variations on the x do not cause a significant variations on the y.

It is possible to demonstrate that the condition number of the Hessian matrix is the ratio between the largest and the smallest eigenvalue. Therefore the condition of strong convexity:

$$mI \leq \nabla^2 f(x) \leq MI \tag{4.10}$$

gives us a upper bound for the condition number, that is $k = M/m$.

We can define as well the *condition number of a set*. First let us define the *width* of a convex set $\mathcal{C} \subseteq \mathbb{R}^n$ in the direction q with $\|q\| = 1$ as:

$$W(C, q) = \sup_{z \in \mathcal{C}} q^T z - \inf_{z \in \mathcal{C}} q^T z$$

then define the *minimum* and *maximum width*:

$$W_{min} = \inf_{\|q\|=1} W(C,q), \qquad W_{max} = \sup_{\|q\|=1} W(C,q)$$

the condition number is defined as:

$$\textbf{cond} = \frac{W_{max}^2}{W_{min}^2}.$$

Geometrically it represents the *eccentricity* of the set, namely it is *thin* if the condition number is high, instead *uniform* in all direction if the condition number is approximately one.

Now we want to derive something that bounds the condition number of a set in a α-sublevel $\mathcal{C}_\alpha = \{x | f(x) \leq \alpha\}$ where $p^* < \alpha < f(x^0)$, since it is going to be a crucial factor in some algorithms. From Eqs.4.5 and 4.8 we can say:

$$p^* + \frac{M}{2}\|y - x^*\|_2^2 \geq f(y) \geq p^* + \frac{m}{2}\|y - x^*\|_2^2$$

Since it is true for all $y \in \mathcal{C}$ we can see that the set $\mathcal{C}$ contains a set $\mathcal{B}_{inner}$ and is contained in $\mathcal{B}_{outer}$ where:

$$B_{inner} = \{y | \|y - x^*\|_2 \leq (2(\alpha - p^*)/M)^{1/2}\}$$

$$B_{outer} = \{y | \|y - x^*\|_2 \leq (2(\alpha - p^*)/m)^{1/2}\}$$

Geometrically, B_{inner} is the ball *inscribed* and B_{outer} the ball *circumscribed* in the set. We can easily see that they give us an idea of the conditional number of the set, since if the set is very *thin* the inner ball is going to be way smaller than the outher ball, therefore

$$\textbf{cond}(\mathcal{C}_\alpha) \leq \frac{M}{m}$$

is large.

It must be remembered that most of the time we do not know m and M so we cannot impose a stopping criteria that depends on these values. Anyway important results on the convergence analysis can be carried out thanks these bounds.

4.2 Descent methods

The first class of methods we are going to see, are the *descend methods*. These methods generate a sequence $x^{(k)}, k = 1, ...$ where

$$x^{(k+1)} = x^{(k)} + t^{(k)} \Delta x^{(k)}$$

in which $t^{(k)} > 0$ (if the $x^{(k)}$ is not optimal) is called *step size or step length* (even though in general it is not equal to $\|x^{(k+1)} - x^{(k)}\|$, $\Delta x^{(k)} \in \mathbb{R}^n$ is called *step or search direction* (notice that it does not have to have unit norm). Let us suppose that these methods are *descent*, namely it always holds that $f(x^{(k+1)}) < f(x^{(k)})$ except when the point is optimal [1]. Since the function is convex $\nabla f(x^{(k)})^T(y - x^{(k)}) \geq 0$ this implies $f(y) \geq f(x^{(k)})$, so the search direction must satisfy

$$\nabla f(x^{(k)})^T \Delta x^{(k)} < 0$$

geometrically, this means that the search direction has to *pull* our point towards the direction where the gradient is negative, or at least with an acute angle with such direction.

For the moment let us suppose that we have the search direction. How can we choose the step size? Several method have been proposed, we are going to have a glance at two of them. For a deeper dissertation please refer to [27, Chapter 9].

- **Exact line search:** this solves the problem $t = \underset{s \geq 0}{\operatorname{argmin}} f(x + s\Delta x)$. This can be used if the cost of the minimization is lower than the research of the step direction;
- **Backtracking line search:** this found a know step in order to minimize the function of a certain amount small enough. The algorithm works like that:
  1. Given a descent direction Δx for f at $x \in \mathbf{dom}\ f$ and $\alpha \in (0, 0.5), \beta \in (0, 1)$
  2. $t := 1$
  3. While $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x, \qquad t := \beta t$

Let us now conduct a convergence analysis on those two method.

4.2.1 Convergence analysis

We assume that f is strongly convex on $\mathcal{S}$ and we have the two constants M and m. Let us define also a function $\tilde{f}(t) = f(x - t\nabla f(x))$ namely the value of the function f calculated in a point y that lies on the line depicted by the gradient of f in x. Substituting $y = x - t\nabla f(x)$ we obtain:

$$\nabla f(t) < f(x) - t\|\nabla f(x)\|_2^2 + \frac{Mt^2}{2}\|\nabla f(x)\|_2^2. \tag{4.11}$$

[1] Notice that most of the optimization algorithm are based on this assumption. In this case we say that the algorithm gives a *relaxational* sequence. Conversely the FG method does not make this assumption.

Now the exact line search method tries to find an exact $t = t^*$, so if we derive on respect to t the right hand side we obtain that is minimized by $t = 1/M$. Substituting :

$$f(x^+) = \tilde{f}(t^*) \geq f(x) - \frac{1}{2M}\|\nabla f(x)\|_2^2.$$

where for matter of simplicity $x^+ = x^{(k+1)}$ and $x = x^{(k)}$. Now let us subtract p^* from both sides:

$$f(x^+) - p^* = \tilde{f}(t^*) \geq f(x) - p^* - \frac{1}{2M}\|\nabla f(x)\|_2^2$$

then from Eq. 4.6 we have:

$$f(x^+) - p^* \leq (1 - m/M)(f(x) - p^*) \quad \rightarrow \quad f(x^{(k)}) - p^* \leq c^k(f(x^{(k)}) - p^*) \tag{4.12}$$

We can make these important conclusion:

- since $c = 1 - m/M < 1$ if $k \to \infty$ the algorithm converges to p^*;
- we have that $f(x^{(k)}) - p^* \leq \epsilon$ after k iterations:

$$\frac{\log((f(x^{(0)}) - p^*)/\epsilon)}{\log(1/c)};$$

- the number of iteration increase logaritmicaly with the distance between initial point and optimal value;
- the number of iteration increase for large condition number, in particular if M/m is large we have that $\log(1/c) = -log(1 - m/M) \approx m/M$, so the number of iteration increases almost linearly with the condition number bound;
- the error $f(x^{(k)}) - p^*$ converges to zero at least as fast as a geometric series. This is called *linear convergence*, indeed if we plot a graph where on the y-axis there is the log of the error and on the x-axis there is the number of iteration, we can easily see that it is linear.

We can in the same way as before, prove that the converge of the backtracking algorithm has the same form:

$$f(x^{(k)}) - p^* \leq c^k(f(x^{(k)}) - p^*)$$

but this time

$$c = 1 - \min\{2m\alpha, 2\beta\alpha m/M\} < 1$$

we do not include the proof, but it can be found in [27, Chapter 9] . We can conclude that the algorithm converges as fast as a geometric series with an exponent that depends on the condition number bound. Therefore the convergence is linear. Now we have just to find a good step direction. We are going to talk about that in the next section.

4.2.2 Gradient descend method

If we choose $\Delta x = -\nabla f(x)$ we have the *gradient descent method*. In Algorithm 1 we report the idea behind the method.

Algorithm 1 Gradient Descend method

Require: $x \in$ **dom** f.
  while err $>$ toll **do**
    $\Delta x \leftarrow -\nabla f(x)$.
    Line search: choose a step size t via exact or backtracking line search.
    Update: $x \leftarrow x + t\Delta x$.
  end while

4.2.3 Conclusions

Through convergence analysis we can summarize the following:

- The gradient method often has approximately linear convergence, i.e. the error $f(x^{(k)}) - p^*$ converges to zero linearly;
- It can be shown that the choice of α, β in the backtracking parameters has an important impact, but non dramatic effect on the convergence. An exact line search sometime improves the convergence, but not in a significant way (see [27]);
- The convergence rate depends greatly on the condition number of the Hessian or the sublevel set. Convergence can be very slow even for problem that are moderately well conditioned.

Therefore the main advantage of these methods is their simplicity, but the main disadvantage is that the convergence rate depend strongly on the condition number of the Hessian or the sublevel sets.

4.2.4 Steepest descend direction

As in the above methods, let us approximate the function in a point x with a linear function:

$$f(x+v) \approx \hat{f}(x+v) = f(x) + \nabla f(x)^T v$$

where $\nabla f(x)^T v$ is the *directional derivative* of the function in the direction v. Now we can wonder which direction v the directional derivative is as negative as possible. Geometrically this means that we are *going down* the function along the *steepest path*, that intuitively will lead us to a faster convergence (usually).

Since the directional derivative is sensible to the magnitude of v let us normalize it, so as to make our choice only based on the direction. We define *normalized steepest descend direction* with respect to a generic norm $\|\cdot\|$ as

$$\Delta x_{nsd} = \text{argmin}\{\nabla f(x)^T v | \|v\| = 1\}. \tag{4.13}$$

Notice that there can be more than one solution.

Algorithm 2 Steepest descend method

Require: a starting point $x \in$ **dom** f
  while err $>$ toll **do**
    Compute steepest descend direction
    Line search. Choose t via backtracking or exact line search
    Update. $x \leftarrow x + t\Delta x$
  end while

Of course if we choose the euclidean norm, the steepest descend direction (not normalized) is $\Delta x_{sd} = -\nabla f(x)$, therefore the steepest descent method coincides with the gradient descent method. Instead with other norms we obtain different directions.

We report the rate of convergence (see [27] for the complete derivation). We know that the euclidean norm is a lower bound for all the other norms, is follows that:

$$\|x\| \geq \gamma \|x\|_2 \qquad \|x\| \geq \tilde{\gamma} \|x\|_2$$

as well as for the previous methods we obtain:

$$f(x^+) - p^* \leq c(f(x) - p^*) \tag{4.14}$$

where

$$c = 1 - 2m\alpha\tilde{\gamma}^2 \text{min}\{1, \beta\gamma^2/M\} < 1.$$

In this case to we have linear convergence like in the gradient method.

Now we can make an interesting observation. Let us suppose that we want to use the *quadratic norm*:

$$\|z\|_P = (z^T P z)^{1/2} = \|P^{1/2}\|_2$$

the last term shows that the quadratic norm correspond to the euclidean norm after a change of coordinates, namely is as like we are minimizing:

$$\bar{f}(\bar{u}) = f(P^{-1/2}\bar{u}) = f(u).$$

Now we said that the gradient methods do not work well when the condition number of the sublevel set is large, that is when the Hessian is ill conditioned. But suppose we

know at least an approximate Hessian $\hat{H}$ in a certain point, a very smart choice would be $P = \hat{H}$, so as the new Hessian :

$$\hat{H}^{-1/2}\nabla^2 f(x^*)\hat{H}^{1/2} \approx I$$

that has a low condition number.

Of course not always we have an approximate Hessian, but when we have it, the steepest descent method with quadratic norm could bring very good results.

4.2.5 Newton's method

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable, working out a second order Taylor expansion we obtain:

$$\hat{f}(x+v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v, \tag{4.15}$$

that is a paraboloid that approximate the function in x. We can easily find the step v that minimize this paraboloid deriving the Eq. 4.15 that is :

$$v = \Delta x_{nt} = -\nabla^2 f(x)^{-1} \nabla f(x)$$

this is called *Newton step*. Since $\nabla^2 f(x)$ is positive definite we can say

$$\nabla f(x)^T \Delta x_{nt} = -\nabla f(x)^T \nabla^2 f(x)^{-1} \Delta f(x) < 0$$

An other interesting interpretation of the Newton's step is that we can see it as the steepest descend direction at x for the Hessian quadratic norm:

$$\|u\|_{\nabla^2 f(x)} = (u^T \nabla^2 f(x) u)^{1/2}.$$

As we said above the steepest descend direction with matrix P converges rapidly if P is equal (or at most similar) to the Hessian, and as we can see, in the Newton's method this matrix is the Hessian indeed. This gives us an insight of the efficiency of the Newton's method.

We now define a quantity that is useful in for the stop criterion and for the analysis of Newton's method. We call *Newton decrement* the quantity:

$$\lambda(x) = \left(\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)\right)^{1/2}.$$

that we can see as:

$$f(x) - \min_y \hat{f}(y) = f(x) - \hat{f}(x + \Delta x_{nt}) = -\frac{1}{2}\lambda(x)^2$$

where we have used the expression $\Delta x_{nt} = -\nabla^2 f(x)^{-1}\nabla f(x)$. Therefore $\lambda^2/2$ is an estimate of $f(x) - p^*$ based on on the quadratic approximation of f at x. Or we can see the Newton decrement also as:

$$\lambda(x) = \left(\Delta x_{nt}^T \nabla^2 f(x)^{-1}\right).$$

Here we present the algorithm. This is usually called *damped* or *guarded* Newton method, because it does not use a fixed step size $t = 1$ like the *pure* Newton method.

Algorithm 3 Newton method

Require: a starting point $x \in$ **dom** f and a tolerance toll > 0
  while err $>$ toll **do**
    Compute the Newton step and decrement.
    Stopping criterion. Terminate if $\lambda^2/2 < \epsilon$.
    Line search. Choose step site t by backtracking line search.
    Update. $x \leftarrow x + t\Delta x_{nt}$
  end while

Convergence analysis

We report the rate of convergence. For the complete derivation see [27] (Chapter 9, page: 488).

Firs of all, we assume that f is twice continuously differentiable and strongly convex, namely $\nabla^2 f(x) \geq mI$ for $x \in \mathcal{S}$. This implies that there exist $M > 0$ such that $\nabla^2 f(x) \leq MI$ for $x \in \mathcal{S}$. In addition we assume that:

$$\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq L\|x - y\|_2 \tag{4.16}$$

that means the Hessian of f is Lipschitz continuous on $\mathcal{S}$ with constant L. Notice that the more similar the function f is to a quadratic function, the smaller is L. Thus a small L means that the function is approximated very well by a quadratic function.

For some $0 < \eta \leq m^2/L$ and $\gamma > 0$ it is possible to demonstrate that:

- If $\|\nabla f(x^{(k)}\|_2 \geq \eta$:

$$f(x^{(k+1)}) - f(x^{(k)}) \leq -\gamma \tag{4.17}$$

- If $\|\nabla f(x^{(k)}\|_2 < \eta$, then the backtracking line search selects $t^(k) = 1$

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)}\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(x^{(k)}\|\right)^2. \tag{4.18}$$

When the second condition $\|\nabla f(x^{(k)}\|_2 < \eta$ is verified we can demonstrate that the algorithm converges *quadratically*, indeed this phase is called *quadratically convergent phase*, instead the first phase, when $\|\nabla f(x^{(k)}\|_2 \geq \eta$ is called *damped phase* because the backtracking chooses a $t < 1$ and converges linearly.

Conclusions

The Newton's method has numerous advantages over gradient and steepest descent methods:

1. Convergence is rapid especially near the optimal point where becomes quadratic. Usually in this phase only 6-7 iterations are needed to reach the optimum with extreme accuracy;
2. Is affine invariant. That means it is insensitive to the choice of coordinates or the condition number of the sublevel sets of the objective.
3. The problem size does not affect dramatically the performance. A problem in $\mathcal{R}^{1000}$ requires only a few steps more than a problem in $\mathcal{R}^{10}$
4. The performance does not depend on the choice of the algorithm parameters, while in the steepest descend are a very critical choice.

The pitfall is mainly the computational effort in storing the Hessian and computing the Newton step. We can solve this problem exploiting the structure of the problem avoiding in this way part of the computational cost, or using a *quasi-Newton method* that uses approximation of the Hessian so as to reduce computational and memory demand.

4.3 Equality constrained minimization

Even though MHE problems have inequality constraints, we are going to see that we can reduce a problem with inequality constrains in a problem with only equality constraints, for this reason let us have a look at the following problem:

$$\min_x f(x) \qquad \text{s.t.} \quad Ax = b \tag{4.19}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex and twice continuously differentiable, $A \in \mathbb{R}^{p\times n}$ with $\mathbf{rank}(A) = p < n$. We assume that there exists an optimal x^*, and we denote $p^* = \inf\{f(x)|Ax = b\} = f(x^*)$.

The KKT conditions tell us that a point $x \in \mathbf{dom}(f)$ is optimal for the problem if and only if there exist a $v^* \in \mathbb{R}^p$ such that

$$Ax^* = b, \qquad \nabla f(x^*) + A^T v^* = 0 \tag{4.20}$$

therefore the problem 4.19 is equivalent to solve the problem 4.20, which is a set of $n+p$ equation in $n+p$ variables. We call *primal feasibility equations* $Ax^* = b$ and *secondary feasibility equations* $\nabla f(x^*) + A^T v^* = 0$ and in general they are non linear.

Now we can solve the problem in two ways:

- Any equality constrained minimization problem can be reduced to an equivalent unconstrained problem by eliminating the constraints, simply working out the equality constraints in order to obtain an expression of our unknown vector and substituting it in the function.
- If the dual function is twice differentiable we can solve the dual unconstrained problem and recover the solution of the equality constrained problem.

Both methods sometimes can destroy a useful *sparsity* that the problem has that can be useful for solving effectively the problem. For this reason we are going to have a look at a Newton method that can directly handle constraints.

Let us see first how to solve a quadratic convex minimization problem because we need it for a complete understanding of the Newton method.

If we have the following problem:

$$\min_x \ f(x) = \frac{1}{2}x^T Px + q^T x + r \qquad \text{s.t.} \quad Ax = b \tag{4.21}$$

where $P \in \mathbb{R}^{n\times n}, P \geq 0$ and $A \in \mathbb{R}^{p\times n}$. The following optimality condition hold:

$$Ax^* = b \quad Px^* + q + A^T v^* = 0$$

which we can write as

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ v^* \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}. \tag{4.22}$$

The square matrix of this problem is called *KKT matrix*. The result of this linear system has to be interpreted as follows:

- If the KKT matrix is nonsingular there is a unique optimal dual pair (x^*, v^*);
- If the KKT matrix is singular, but is solvable, any solution leads to an optimal pair;
- If the KKT matrix is singular and not solvable, it means that the problem is not bounded below or the point is infeasible.

4.3.1 Newton's method with equality constraints

This method similar to the Newton's method without constraints but has two differences: the starting point must be feasible ($x \in \mathbf{dom}\ f$ and $Ax = b$) and the Newton step is modified in order to make sure that the direction is feasible($A\Delta x_{nt} = 0$, this means that the direction is perpendicular to the plane that describes the constraints).

We have the problem 4.19, and through a second-order Taylor approximation we obtain:

$$\min_x \ \hat{f}(x+v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v \qquad \text{s.t. } A(x+v) = b \tag{4.23}$$

This is a convex quadratic minimization problem and can be solved analytically. Indeed at the optimal point is true that:

$$Ax^* = b, \qquad \nabla f(x^*) + A^T v^* = 0$$

Substituting $x^* = x + \Delta x_{nt}$ and $w = v$ and linearising the gradient we obtain:

$$A(x + \Delta x_{nt}) = b, \qquad \nabla f(x + \Delta x_{nt}) + A^T w \approx \nabla f(x) + \nabla^2 f(x) \Delta x_{nt} + A^T w = 0$$

and since $Ax = b$

$$A\Delta x_{nt} = 0, \qquad \nabla^2 f(x) \Delta x_{nt} + A^T w = -\nabla f(x)$$

therefore Newton step is going to be the solution to the quadratic problem:

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{nt} \\ w \end{bmatrix} = \begin{bmatrix} -\Delta f(x) \\ 0 \end{bmatrix}$$

that we have construct following the same procedure that we used in Eq. 4.22. We can notice that the Newton step is always a feasible direction because we imposed $A\Delta x_{nt} = 0$, so every point $x + t\Delta x_{nt}$ is feasible (unless $x^{(k)}$ is optimal), indeed $A(x + tv) = b$. Δx_{nt} is also a descend direction.

4.4 Interior point method

We want to solve a convex optimization problem like the following:

$$\min_x f(x) \qquad \text{s.t.} \quad \begin{cases} h(x) & \leq 0 \\ Ax & = 0 \end{cases} \tag{4.24}$$

where $f, g : \mathbb{R}^n \to \mathbb{R}$ are convex and twice continuously differentiable, and $A \in \mathbb{R}^{p \times n}$ with $\mathbf{rank}(A) = p < n$. Let us assume that the problem is solvable, namely that x^* exist, and let us denote $f(x^*) = p^*$, we assume also that the problem is strictly feasible, that is there exist $x \in \mathcal{D}$ that satisfies $Ax = b$ and $g(x) \leq 0$, this means that Slater's constraint qualification holds, and there exist a dual optimal $\lambda^* \in \mathbb{R}^m, v^* \in \mathbb{R}^p$ which satisfy the KKT conditions.

$$\begin{aligned} Ax^* = b, \quad h(x^*) \leq 0 \\ \lambda^* \geq 0 \\ \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \lambda g(x^*) + A^T v^* = 0 \\ \lambda_i^* = 0 \end{aligned} \tag{4.25}$$

Notice that we can look at the optimization problem as as divided in steps (or *hierarchy* [27]): the interior point methods solve an optimization problem with linear equality constraints and nonlinear inequality constraints by reducing it to a sequence of linear equality constrained problems. Once we have them we can apply the Newton's method to solve these equality constrained problems by reducing them to a sequence of quadratic linearly constrained problem. These are reduced into a unconstrained problem through the KKT condition, that can be solved with a method for unconstrained minimization, like Newton, Steepest descendent etc.

We are going to see the *barrier method* since we are going to use this method for the MHE problem.

4.4.1 Logarithmic barrier

The idea is to eliminate the inequality constraints by modifying the objective function with a term that increases the cost if these constraints are violated. This barrier can be a *logarithmic barrier*:

$$\min_x f(x) + \sum_{i=1}^{m} -(1/t)\log(-g_i(x)) \qquad \text{s.t.} \quad Ax = b \tag{4.26}$$

Notice that the objective is still convex if f is convex, since the added term is convex, increasing with u and differentiable. The problem now is an approximation of the original problem, we can intuitively understand that the more the parameter t grows the more the approximation improves. Let us denote:

$$\Phi(x) = \sum_{i=1}^{m} \log(-h_i(x))$$
$$\nabla\Phi(x) = \sum_{i=1}^{m} \frac{1}{-h_i(x)} \nabla h_i(x)$$
$$\nabla^2\Phi(x) = \sum_{i=1}^{m} \frac{1}{h_i(x)^2} \nabla h_i(x) \nabla h_i(x)^T + \sum_{i=1}^{m} \frac{1}{h_i(x)} \nabla^2 h_i(x).$$

We need first the definition of *central path*.

Definition 4.2 (Central path)**.** Let us assume that the Eq. 4.26 has a unique solution for all $t > 0$. We define $x^*(t)$ the solution and *central path* as the set of points $x^*(t)$ for $t > 0$, and let us call these points *central points*. The latter must respect the following necessary and sufficient condition:

$$Ax^*(t) = b \qquad h_i(x^*(t)) < 0 \quad i = 1; ..., m,$$

and there exist a v^* such that:

$$t\nabla f(x^*(t)) + \nabla\Phi(x^*(t)) + A^T\hat{v} = t\nabla f(x^*(t)) + \sum_{i=1}^{m} \frac{1}{-h_i(x^*(t))} \nabla h_i(x^*(t)) + A^T\hat{v} = 0$$

From the last formula,dividing by t, we can see that the feasible points $\lambda_i^*(t), v^*(t)$ are:

$$\lambda_i^*(t) = \frac{1}{-h_i(x^*(t))}, \quad i = 1; ..., m, \qquad v^*(t) = \hat{v}/t \tag{4.27}$$

therefore

$$G(\lambda_i^*(t), v^*(t)) = f(x^*(t)) + \sum_{i=1}^{m} \lambda_i^*(t) h_i(x^*(t)) + v^*(t)^T(Ax^*(t) - b) = f(x^*(t)) - m/t.$$

The duality gap associated with $x^*(t)$ and the dual feasible pair $\lambda_i(t), v^*(t)$ is m/t. And since $g(\lambda_i^*(t), v^*(t))$ it is an underestimation of the optimal point and $f(x^*(t))$ a upperestimation of the optimal point, we obtain:

$$f(x^*(t)) - p^* \leq m/t$$

namely $x^*(t)$ is maximum an m/t-suboptimal, and $x^*(t)$ converges to an optimal point as $t \to \infty$. So it would be enough to choose $t = m/\epsilon$ and solve the problem 4.26 to

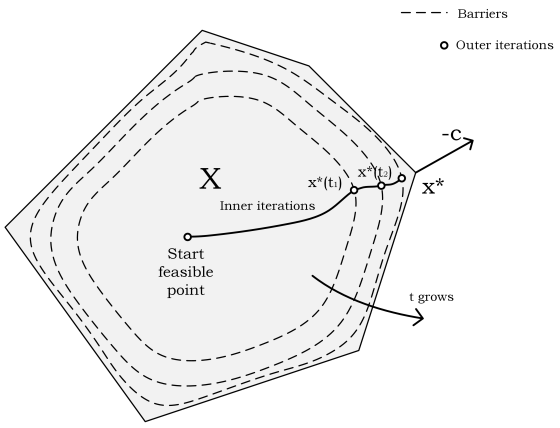


FIGURE 4.1: Example interior point method. We can see that when t grows the barriers are closer to the constraints. Is important to remember that these barriers are not actually constraints but zones where the cost function starts to increase logarithmically.

have the required accuracy, but it has been seen that for high dimensional problems and for high accuracy it does not work well. For this reason the *barrier method* or *path following method* has been proposed, where the factor t is augmented throughout we get close to the optimal point, starting at each iteration from the point calculated the previous iteration.

In Algorithm 4 we show the method. At each iteration is computed a *central point*

Algorithm 4 Barrier method

Require: a strictly feasible $x, t \leftarrow t^{(0)} > 0, \mu > 1$ e tolerance $toll > 0$ Repeat
  while $m/t < toll$ **do**
    Centering step. Compute $x^*(t)$ by minimizing $tf + \phi$ s.t.$Ax = b$ starting at x;
    Update. $x \leftarrow x^*(t)$;
    Increase t. $t \leftarrow \mu t$.
  end while

$x^*(t)$ that is the minimizer of the problem 4.26 with a certain t (*outer iteration* phase) also called *centering problem*. This point is computed through, for example, a Newton method that handles equality constraints (*inner iteration* phase). After that the t factor is increased and the problem is solved again but starting from the previous $x^*(t)$.

We wonder now: how much do we have to be accurate with during the centering phase? Of course by definition the central path will bring us to the solution of the original problem anyway, but it has been seen that the difference in number of iterations in calculating a good centering and a excellent is not so large, therefore usually is we consider to calculate an exact centering.

A another important parameter is μ. If μ is big, there will be large outer steps, therefore a few number of outer iteration, but we pay the price of an large number of inner iteration, since the new starting point is not probably good, namely its far from the next starting

point. Conversely if μ is small, we made a higher number of outer steps (but more precise) and a lower number of inner steps. The optimal μ changes from problem to problem but it has been seen that a values between 10 to 20 seem to work well.

4.4.2 Inner iterations - Newton method

We are going to see how the Newton method faces up with this equality constrained problem that we have obtained:

$$\min_x \ f(x) + \frac{1}{t}\Phi(x) \qquad \text{s.t.} \quad Ax = b. \tag{4.28}$$

Let us write the KKT conditions:

$$\begin{aligned} \nabla f(x) + \frac{1}{t}\nabla\Phi(x) + A^T\nu &= 0 \\ Ax &= 0 \end{aligned}$$

we said that $\nabla\Phi(x) = \sum_{i=1} m(-1/g_i(x))\nabla g_i(x)$ therefore:

$$\nabla f(x) + \frac{1}{t}\sum_{i=1}^{m} \frac{1}{-g_i(x)}\nabla g_i(x) + A^T v = 0, \qquad Ax = 0$$

The Newton method find the step that minimize the second order approximation of the objective:

$$\begin{aligned} &\nabla f(x+v) + \sum_{i=1}^{m} \frac{1}{-tg_i(x+v)}\nabla g_i(x+v) = \\ &\approx \nabla f(x) + \sum_{i=1}^{m} \frac{1}{-tg_i(x)}\nabla g_i(x) + \nabla^2 f(x)v + \sum_{i=1}^{m} \frac{1}{-tg_i(x)}\nabla^2 g_i(x)v + \\ &+ \sum_{i=1}^{m} \frac{1}{tg_i(x)^2}\nabla g_i(x)\nabla g_i(x)^T v \end{aligned}$$

this leads to the linear equations:

$$Hv + A^t\nu = -g, \qquad Av = 0,$$

where

$$\begin{aligned} H &= \nabla^2 f(x)v + \sum_{i=1}^{m} \frac{1}{-tg_i(x)}\nabla^2 g_i(x)v + \sum_{i=1}^{m} \frac{1}{tg_i(x)^2}\nabla g_i(x)\nabla g_i(x)^T v \\ g &= \nabla f(x) + \sum_{i=1}^{m} \frac{1}{-tg_i(x)}\nabla g_i(x) \end{aligned}$$

where, since

$$H = \nabla^2 f(x) + \frac{1}{t}\nabla^2\Phi(x), \qquad g = \nabla f(x) + \frac{1}{t}\nabla\Phi(x)$$

finally

$$tH\Delta x_{nt} + A^T v_{nt} = -tg, \qquad A\Delta x_{nt} = 0, v = \Delta x, \quad \nu = (1/t)\nu$$

4.5 Nesterov's Fast gradient method

This method was developed by Yurii Nesterov in 1983. The basic ideas are: dropping the condition of a relaxation sequence (namely we do not ask to our method that $f^k \geq f^{k+1}$ where k is the iteration) and using an *estimated sequence.* Let us star with this definition:

Definition 4.3 (Estimate sequence)**.** A pair of sequences $\{\phi_k(x)\}_{k=0}^{\infty}$ and $\{\lambda_k\}_{k=0}^{\infty}, \lambda \geq 0$ is called *estimate sequence* of a function $f(x)$ if:$\lambda_k \to 0$ and for any $x \in \mathbb{R}^n$ and $k > 0$ we have:

$$\phi_k(x) \leq (1 - \lambda_k)f(x) + \lambda_k\phi_0(x)) \tag{4.29}$$

Lemma 4.4. *notice that if for some sequence* $\{x_k\}$ *we have:*

$$f(x_k) \leq \phi_k^* \equiv \min_{x\in\mathbb{R}^n} \phi_k(x) \tag{4.30}$$

then $f(x_k) - f^* \leq \lambda_k[\phi_0(x^*) - f^*] \to 0$

Proof. Indeed:

$$\begin{aligned} f(x_k) \leq \phi_k^* = \min_{x\in\mathbb{R}^n} \phi_k(x) &\leq \min_{x\in\mathbb{R}^n} [(1 - \lambda_k)f(x) + \lambda_k\phi_0(x)] \\ &\leq (1 - \lambda_k)f(x^*) + \lambda_k\phi_0(x^*). \end{aligned}$$

□

Thus, for any sequence $\{x_k\}$ satisfying Eq. 4.30 we can derive its rate of convergence $\{f(x_k) - f*\}$ directly from the rate of convergence of sequence $\{\lambda_k\}$.

In the next sections we are going to see how to find an estimate sequence and how to ensure Eq. 4.30.

4.5.1 Estimate sequence

Lemma 4.5 (Estimated sequence)**.**

- *If f is strictly convex;*
- $\phi_0(x)$ *is an arbitrary function on* $\mathbb{R}^n$*;*
- $y_k{}_{k=0}^{\infty}$ *is an arbitrary sequence in* $\mathbb{R}^n$*;*
- $\{\alpha_k\}_{k=0}^{\infty}: \quad \alpha_k \in (0,1), \quad \sum_{k=0}^{\infty} = \infty$
- $\lambda_0 = 1$

then we can choose this estimate sequence:

$$\begin{aligned} \lambda_{k+1} &= (1-\alpha_k)\lambda_k \\ \phi_{k+1}(x) &= (1-\lambda_k)\phi_k(x) + \alpha_k[f(y_k) + \nabla^T f(y_k)(x-y_k) + \frac{\mu}{2}\|x-y_k\|^2] \end{aligned} \tag{4.31}$$

Proof. Indeed, $\phi_0(x) \le (1-\lambda_0)f(x) + \lambda_0\phi_0(x) \equiv \phi_0(x)$. Then for some $k \ge 0$:

$$\begin{aligned} \phi_{k+1}(x) &\le (1-\alpha_k)\phi_k(x) + \alpha_k f(x) \\ &= (1-(1-\alpha_k)\lambda_k)f(x) + (1-\alpha_k)(\phi_k(x) - (1-\lambda_k)f(x)) \\ &\le (1-(1-\alpha_k)\lambda_k)f(x) + (1-\alpha_k)\lambda_k\phi_0(x) \\ &= (1-\lambda_{k+1})f(x) + \lambda_{k+1}\phi_0(x) \end{aligned}$$

□

Now let us decide the form of the function $\phi_0(x)$ if we choose the form $\phi_0(x) = \phi_0^*(x) + \frac{\gamma_0}{2}\|x-v_0\|^2$ then the process 4.31 preserves the canonical form of function:

$$\phi_k(x) = \phi_k^*(x) + \frac{\gamma_0}{2}\|x-v_0\|^2 \tag{4.32}$$

where the sequences $\{\gamma_{k+1}\}, \{v_k\}$ and $\{\phi_k^*(x)\}$ are defined as follows:

$$\begin{aligned} \gamma_{k+1} &= (1-\alpha_k)\gamma_k + \alpha_k\mu \\ v_{k+1} &= \frac{1}{y_{k+1}}[(1-\alpha_k)\gamma_k v_k + \alpha_k\mu y_k - \alpha_k f'(y_k)] \\ \phi_0^*(x) &= (1-\alpha_k)\phi_k + \alpha_k f(y_k) - \frac{\alpha^2}{2\gamma_k+1}\|f'(y_k)\|^2 + \\ &+ \frac{\alpha_k(1-\alpha_k)\gamma_k}{\gamma_{k+1}}\left(\frac{\mu}{2}\|y_k-v_k\|^2 + \nabla^T f(y_k)(v_k-y_k)\right) \end{aligned}$$

We are close to an algorithm scheme. Assume that we have $\Phi_k^* \geq f(x_k)$, then in view of the previous lemma,

$$\Phi_{k+1}^* \leq (1-\alpha_k)f(x_k) + \alpha_k f(y_k) - \frac{\alpha_k^2}{2\gamma_{k+1}}\|f'(y_k)\|^2 + \frac{\alpha_k(1-\alpha_k)\gamma_k}{\gamma_{k+1}}f'^T(y_k)(v_k - y_k).$$

Since $f(x_k) \geq f(y_k) + f^T(y_k)(x_k - y_k)$ we get the following estimate:

$$\Phi_{k+1}^* = f(y_k) - \frac{\alpha_k^2}{2\gamma_{k+1}}\|f'(y_k)\|^2 + (1-\alpha_k)(f'^T(y_k)\frac{\alpha_k\gamma_k}{y_{k+1}}(v_k - y_k) + x_k - y_k).$$

We want to have $\Phi_{k+1}^* \geq f(x_{k+1})$. The simplest way to ensure the inequality:

$$f(y_k)\frac{1}{2L}\|f'(y_k)\|^2 \geq f(x_{k+1})$$

is to take the gradient step $x_{k+1} = y_k - 1/Lf'(x_k)$. Let us define α_k as follows:

$$L\alpha_k^2 = (1-\alpha_k)\gamma_k + \alpha_k\mu.$$

then we have $\alpha_k^2/2\gamma_{k+1} = 1/2L$ and we can replace the previous inequality by the following:

$$\Phi_{k+1}^* \geq f(x_{k+1}) + (1-\alpha_k)f'^T(y_k)\left(\frac{\alpha_k\gamma_k}{\gamma_{k+1}}(v_k - y_k) + x_k - y_k\right).$$

Now we can use choice of y_k, that we can find from the equation $(\alpha_k\gamma_k)/(\gamma_{k+1})(v_k - y_k) + x_k - y_k = 0$, that is:

$$y_k = \frac{\alpha_k\gamma_k v_k + \gamma_{k+1}x_k}{\gamma_k + \alpha_k\mu}$$

from this we can obtained a constant step scheme in Algorithm 5. Algorithm 5 can be

Algorithm 5 Constant Step Scheme

Require: $x_0 \in \mathbb{R}^n$ and $\gamma_0 > 0$.
  $v_0 \leftarrow x_0$.
  while *err* > *toll* **do**
    Compute $\alpha_k \in (0,1)$ from $L\alpha_k^2 = (1-\alpha_k)\gamma_k + \alpha_k\mu$
    $\gamma_{k+1} \leftarrow (1-\alpha_k)\gamma + \alpha_k\mu$
    $y_k = \frac{\alpha_k\gamma_k v_k + \gamma_{k+1}x_k}{\gamma_k + \alpha_k\mu}$.
    Compute $f(y_k)$ and $f'(y_k)$.
    $x_{k+1} \leftarrow y_k - 1/Lf'(y_k)$.
    $v_{k+1} \leftarrow \frac{1}{\gamma_{k+1}}[(1-\alpha_k)\gamma_k v_k + \alpha_k\mu y_k - \alpha_k f'(y_k)]$.
  end while

written in simpler terms as in Algorithm 6 where $\beta_k = \alpha_k(1-\alpha)/(\alpha_k^2 + \alpha_{k+1})$.

Algorithm 6 Constant Step Scheme II

Require: $x_0 \in \mathbb{R}^n$ and $\alpha_0 \in (0,1)$.
  $v_0 \leftarrow x_0$ and $q \leftarrow \mu/L$.
  while $err > toll$ **do**
    Compute $f(y_k)$ and $f'(y_k)$.
    $x_{k+1} \leftarrow y_k - 1/Lf'(y_k)$.
    Compute $\alpha_k \in (0,1)$ from $L\alpha_k^2 = (1-\alpha_k)\gamma_k + \alpha_k\mu$
    $\beta_k \leftarrow \alpha_k(1-\alpha_k)/(\alpha_k^2 + \alpha_{k+1})$
    $y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k)$
  end while

it is possible to demonstrate that if we chose $\alpha_0 = \sqrt{\mu/L}$ we obtain:

$$\alpha_k = \sqrt{\frac{\mu}{L}} \qquad \beta_k = \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}.$$

For the constrained case we cannot use the gradient step because could lead to an infeasible point (or the initial point could be infeasible) therefore we introduce the concept of *Gradient mapping* that inherits the most important properties of the gradient.

Definition 4.6 (Gradient mapping)**.** Let $\gamma \in \mathbb{R}^+$ we define *gradient mapping of f on the set $\mathcal{Q}$*

$$g_Q(\bar{x};\gamma) = \gamma(\bar{x} - x_Q(\bar{x},\gamma)) \tag{4.33}$$

where

$$x_Q(\bar{x};\gamma) = \arg\min \left[f(\bar{x}) + \nabla f^T(\bar{x})(x-\bar{x}) + \frac{\gamma}{2}\|x-\bar{x}\|^2 \right] \tag{4.34}$$

we can calculate the next step $x_{k+1} = x_Q(y_k; L)$. In this work we use the *constant step scheme* [28, Scheme 2.2.9] choosing $\alpha_k = \sqrt{\mu/L}$ and consequently obtaining the Algorithm 7 in § 5.3.

Chapter 5

MHE algorithm and validation

We can consider our problem consisting of four parts:

I System;

II Estimation;

III Optimization;

IV Control;

we deal with these parts in the following sections.

5.1 System

Our system is *linear time-invariant constrained* and *discrete*:

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{5.1}$$

$$y_k = Cx_k + v_k \tag{5.2}$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $y \in \mathbb{R}^p$. The constraints are *box constraints*, that is:

$$\mathcal{X} = \{x \in \mathbb{R}^n | x_l \leq x_k \leq x_r\} \tag{5.3}$$

$$\mathcal{V}_k = \{x \in \mathbb{R}^n | v_l \leq y_k - Cx_k \leq v_r\} \tag{5.4}$$

where l and u represent the lower and upper bound respectively. We call $\mathcal{X}$ and $\mathcal{V}_k$ the *sets of constraints*. Notice that the set $\mathcal{V}_k$ needs to be updated each time step with new measurement.

An important note on the box constraints: The box constrained problem is not as over-restrictive as one can think. Indeed is not rare to have a system where we have constraints only on the the state's maximum and minimum (we can think of a upper bound as a maximum capacity of a tank, or a maximum speed of a vehicle), and also the sensors accuracy, which can give us valid bounds on the measurement, represents a box set of constraints. Nevertheless this assumption must be kept in mind along all this dissertation, since it represents the mainstay on which the fast gradient method grounds because, as we have already said, it permits to compute effectively the projection. Anyway, other type simple constraints, like polytopic constraints, could be applied in principle, maintaining a simple projection formulation.

5.2 Estimation

The MHE problem can be written in a similar way of the MPC:

$$J(\hat{v}, \hat{w}, \hat{x}_{k-N}) = \sum_{i=k-N}^{k-1} \frac{1}{2}\|\hat{w}_i\|^2_{Q^{-1}} + \sum_{i=k-N}^{k} \frac{1}{2}\|\hat{v}_i\|^2_{R^{-1}} + \frac{1}{2}\|\tilde{x}_{k-N} - \hat{x}_{k-N}\|^2_{P^{-1}_{k-N}} \tag{5.5}$$

where the $\hat{(\cdot)}$ represents the optimization variable, N is the horizon length, $\tilde{x}_{k-N}$ is our best priori state information and $\|z\|^2_X$ is the short-hand notation for $z^T X z$. $Q^{-1} \in \mathbb{R}^{n\times n}$ and $R^{-1} \in \mathbb{R}^{p\times p}$ are the weightings matrices of the cost function and should be wisely chosen. The matrix $P^{-1}_{k-N} \in \mathbb{R}^{n\times n}$ is prior weighting matrix. Let us formulate the optimization problem in a *condensed way*, namely we are going to use *only* the state sequence $\{\hat{x}_i\}$ as optimization variables following the idea of [29] we hide the errors w_i and v_i using Eq. 5.1 so the cost function looks like:

$$J = \sum_{i=k-N}^{k-1} \frac{1}{2}\|\hat{x}_{i+1} - A\hat{x}_i - Bu_i\|^2_{Q^{-1}} + \sum_{i=k-N}^{k} \frac{1}{2}\|y_i - C\hat{x}_i\|^2_{R^{-1}} + \frac{1}{2}\|\tilde{x}_{k-N} - \hat{x}_{k-N}\|^2_{P^{-1}_{k-N}} \tag{5.6}$$

we can also write, after a little algebra:

$$J = \frac{1}{2}\hat{\mathbf{x}}^T H \hat{\mathbf{x}} + \hat{\mathbf{x}}^T f(\tilde{x}_{k-N}, \mathbf{y}, u) + g(\tilde{x}_{k-N}, \mathbf{y}, u) \tag{5.7}$$

where we denoted

$$\hat{\mathbf{x}} = (\hat{x}^T_{k-N|k}, ..., \hat{x}^T_{k|k})^T \tag{5.8}$$

and the notation $\hat{(\cdot)}_{k-N|k}$ indicates the state estimation at the time step $k-N$ computed at the time step k.

We want to find the solution of the following problem:

$$\hat{\mathbf{x}} = \arg\min_{x} J(\mathbf{x}, \mathbf{y}, u) \tag{5.9}$$
$$x_{i|k-1} \in \mathcal{U}_i$$

where $\mathcal{U}_i = \mathcal{X} \cap \mathcal{V}_i$ is the box constrained set.

5.2.1 Arrival cost matrix update

Since the MHE problem neglects the informations that are not in the horizon arises the problem of estimation stability. We update the prior weighting matrix P_{k-N}^{-1} using the Kalman Filter theory, namely using an *unconstrained full information estimation*. It is possible to demonstrate that for a linear *detectable* systems with convergent disturbances to zero and an quadratic objective function like 5.5the constrained MHE is Globally Asymptotically Stable 2.7. In our the noises are not convergent, but are limited, therefore the estimation will be stable, but not asymptotically, to the real state.

5.3 Optimization - Fast Gradient method

Once we have obtained the problem 5.7, we have to solve it through our Fast Gradient (FG) method drafted in Algorithm 7. L and μ are maximum and minimum eigenvalue

Algorithm 7 Fast gradient algorithm

Require: L,μ,$\mathbf{x}^0$,ϵ
  $z \leftarrow \mathbf{x}^0$, $x_{old} \leftarrow \mathbf{x}^0$
  $\nabla J \leftarrow Hz + f$ ▷ Calculate gradient
  while $err > \epsilon$ **do**
    $x \leftarrow z - 1/L\nabla J$
    $x \leftarrow \text{proj}(x, \mathcal{U}_i)$ ▷ Projection over $\mathbb{W}$
    $z \leftarrow x + \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(x - x_{old})$ ▷ Step
    $x_{old} \leftarrow x$
    Find err ▷ Error evaluation
    $\nabla J \leftarrow Hz + f$
  end while
  return $\mathbf{x}^*$ ▷ Optimization result

of H, $\mathbf{x}^0$ is the starting point of the iterative algorithm that is chosen to be:

$$\mathbf{x}^0 = (\hat{x}^{*T}_{k-N|k-1}, ..., \hat{x}^{*T}_{k-1|k-1}, \mathbf{0}^T)^T$$

where $(\cdot)^*$ represents the solution of the optimization problem and $\mathbf{0} \in \mathbb{R}^n$ is a all-zero vector. This is called *warm starting* because we take advantage of the solution of the

optimization problem in the previous time-step in order to be closer (hopefully) to the solution a time-step further.

As we are going to see the *eigenvalues evaluation* and the *error evaluation* are the most critical parts of the algorithm, and they can be approached in several way.

5.4 Stopping criteria selection

We have studied several stopping criteria in order for our optimization algorithm:

Tolerance on the solution: namely we impose a tolerance on the difference $|x^{(k)} - x^{(k+1)}| \leq toll_x$. However, we can discard immediately this method because is could be ill-defined for some important problem classes, e.g. for a smooth convex objective function that lacks in strong convexity [28, §2.1.2], therefore we can immediately discard this approach.

Tolerance on complementary slackness: Let us have a look at an approach that uses the KKT condition. As we said in § 3.8.2 at the optimal point the solution must respect the KKT conditions:

$$\begin{aligned} g_i(x^*) &= 0, \qquad i = 1, ..., m \\ h_i(x^*) &\leq 0, \qquad i = 1, ..., p \\ \lambda_i^* &\geq 0, \qquad i = 1, ..., m \\ \lambda_i^* h_i(x^*) &= 0, \qquad i = 1, ..., m \\ \nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{i=1}^{p} v_i^* \nabla h_i(x^*) &= 0 \end{aligned} \tag{5.10}$$

that in our case can be simplified:

$$Hx + f^T + \lambda_1 - \lambda_2 = 0 \tag{5.11a}$$

$$(x - x_u) \leq 0, \quad (x_l - x) \leq 0 \tag{5.11b}$$

$$\lambda_1 \geq 0, \quad \lambda_2 \geq 0 \tag{5.11c}$$

$$\lambda_1(x - x_u) = 0, \quad \lambda_2(x_l - x) = 0. \tag{5.11d}$$

The idea is to use the Eq. 5.11a in order to find $\lambda_1 - \lambda_2$ since we can always calculate the gradient $Hx + f^T$. Furthermore from the 5.11c equation we know that the Lagrange multipliers λ_1, λ_2 have to be positive and since we have box constraints they are going to be perpendicular, therefore we can calculate λ_1 and λ_2. The 5.11b condition instead

is automatically respected if after the projection step. So the only condition that we have to check is the 5.11d the *complementary slackness*. We check that $\max(\lambda_1(x - x_u), \lambda_2(x_l - x)) \geq toll_{cs}$. Notice that in order to have the 5.11b respected we have to compute the gradient in 5.11a *after* projection. Since calculating again the gradient could be computational expensive, we have decided to check the conditions every 10 steps.

Off-line number of iterations computation: we can apply the concept in [30, Preposition 2] to the MHE case. We can have the condition $f(x^{(k)}) - f(x^*) \leq \epsilon$ after at most:

$$k_{max} = \min \left\{ \left\lceil \frac{\ln 2\epsilon - \ln Ld^2}{\ln\left(1 - \sqrt{\frac{\mu}{L}}\right)} \right\rceil, \left\lceil \sqrt{\frac{2Ld^2}{\epsilon}} - 2 \right\rceil \right\} \tag{5.12}$$

where

$$d^2 = \max_{x \in \mathcal{U}_i^N} \|x - x^0\|^2.$$

where x^0 is the starting point of the algorithm and it is *the same at each time-step* and $\mathcal{U}_i$ is the box constrained set. This means that the approach is *cold starting*, thus we do not take advantage of the solution in the previous step but we start the optimization every time from the same point. Since our bound are simple, if we choose x^0 as the center of the box constraints d^2 is nothing else that the squared radius of the set $\mathcal{X}$ and can be easily computed.

Stopping criteria tailored for Fast Gradient method [15] A corollary of *gradient mapping theorem* [28, Theorem 2.2.7] is that:

$$f(x^{(k)}) - f^* \leq \frac{1}{2}\left(\frac{1}{\mu} - \frac{1}{L}\right) \|g(z^{(k-1)}, 1/L)\|^2 \tag{5.13}$$

where $g(z^{(k-1)}, 1/L) = L(z^{(k-1)} - x^{(k)})$ is the gradient mapping (Def. 4.6). From this follow that

$$\frac{1}{2}\left(\frac{1}{\mu} - \frac{1}{L}\right) \|L(z^{(k-1)} - x^{(k)})\|^2 \leq \epsilon \tag{5.14}$$

where x and w are defined in Algorithm 7. Notice that the FLOPs go linearly with the matrix dimension. The Eq. 5.14 has been used in our algorithm.

In Figs.5.1 & 5.2 we present the comparison between these methods, for a random sparse symmetric positive-definite matrix 100×100. As a reference we have plotted the ideal stopping criteria $|f(x^{(k)}) - f(x^*)| \leq toll_f$ where $f(x^*)$ as been calculated with active-set. The tolerance is 10^{-8} in all cases. The blue line represents the FG convergence rate and the crosses and circle represent where the respective algorithm has stopped.

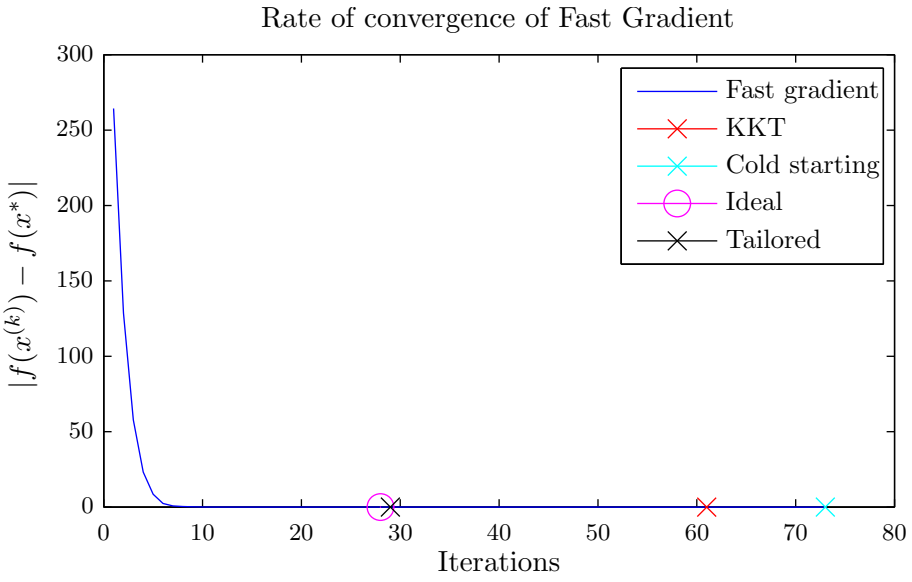


FIGURE 5.1: Comparison between stopping criteria methods - Linear scale.

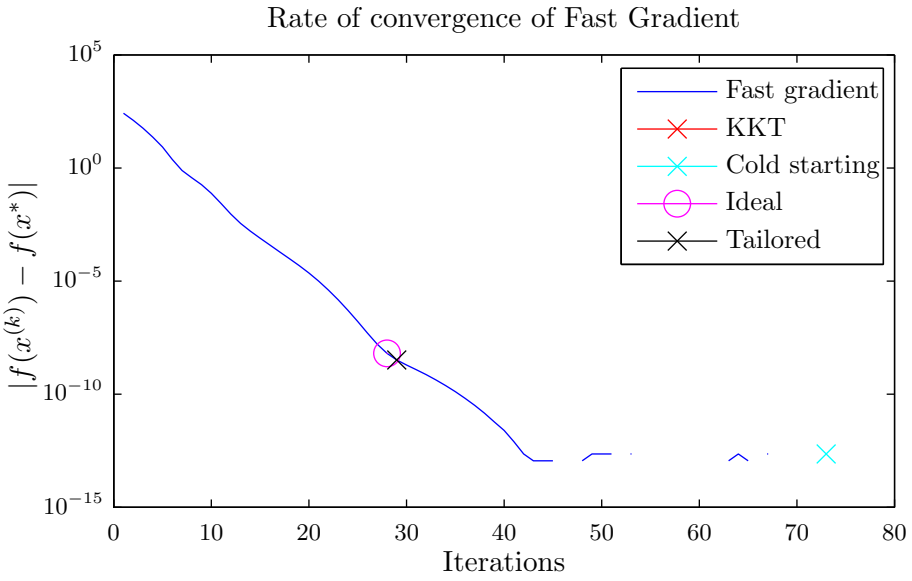


FIGURE 5.2: Comparison between stopping criteria methods - Logarithmic scale. The "holes" are due to the fact that the value in those points is under the machine precision and is approximated to zero, that cannot clearly plotted in a logarithmic scale.

Remark: a comparison with the KKT is not possible since in that case we use a tolerance on the complementary slackness and not on the function, therefore the KKT criteria is likely to have stopped when $|f(x^{(k)}) - f(x^*)| \leq toll_f \neq 10^{-8}$. Anyway we have decided to report where the KKT criteria stops with a tolerance on the complementary slackness of 10^{-8}. As we can see the stopping criteria tailored stops near the ideal case while the cold starting criteria stops after more than two times the same number iterations. The same simulation has been repeated for several random matrices and the pattern has been always the same.

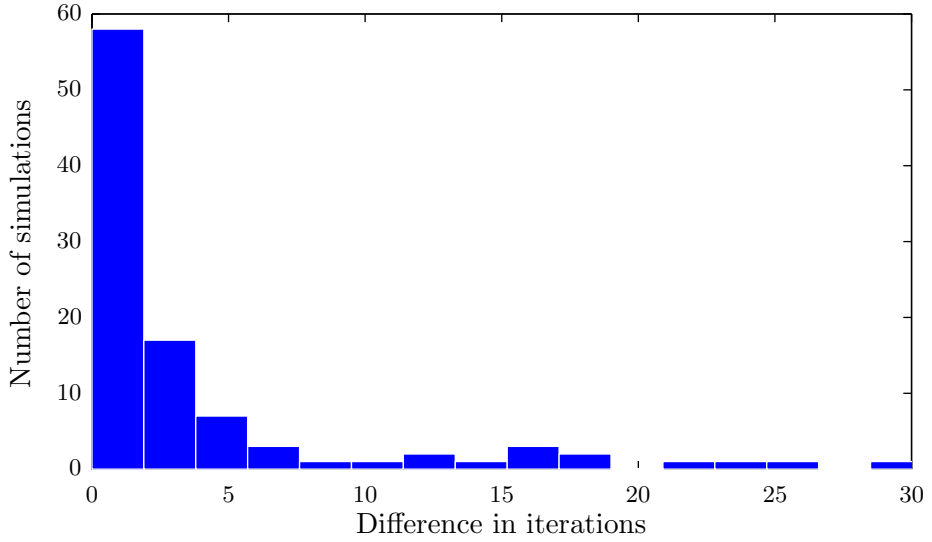


FIGURE 5.3: Difference in FG iteration between ideal and tailored stopping criteria.

In Fig. 5.3 we report the the difference between the number of iteration needed with the ideal stopping criteria and tailored stopping criteria. We run 100 simulations with random matrices. We notice that most of the time the tailored method estimates less than 4 iterations more than necessary.

5.5 Algorithm bottleneck

Let us have a look at the time-bottleneck of the algorithm using the approaches discussed in the § 5.4 when we use the function eig in order to calculate the eigenvalues. The horizon length is 100 steps large, the system has 10 states, therefore the matrix H that goes into the optimizer is 1000×1000. The total time spent for solving the entire MHE problem is $186.217s$ and of this time the 96.8% ($180.267s$) is spent for in the FG algorithm. If we exploit the time spent in each part of the FG algorithm we notice that the *eigenvalues computation step* is the most time demanding, followed by the *gradient computation step* (see Fig. 5.4). Notice that the eigenvalues are calculated ones in each time step, whereas the gradient $\nabla J = Hw + f$ is calculated several times in the same step because is inside the FG iteration loop (see 7).

Now if we look at how the time spent in the different parts varies if we increase the horizon from 10 to 100 steps with the same system (Fig. 5.5), we see that the eigenvalue computation becomes the bottleneck for large scale matrices (notice that the matrix dimension varies accordingly from 100×100 to 1000×1000). Since the MHE problems usually very large, it is clear that the eigenvalue calculation represents the algorithm bottleneck, therefore we have put our effort into it in order to speed the solution up.

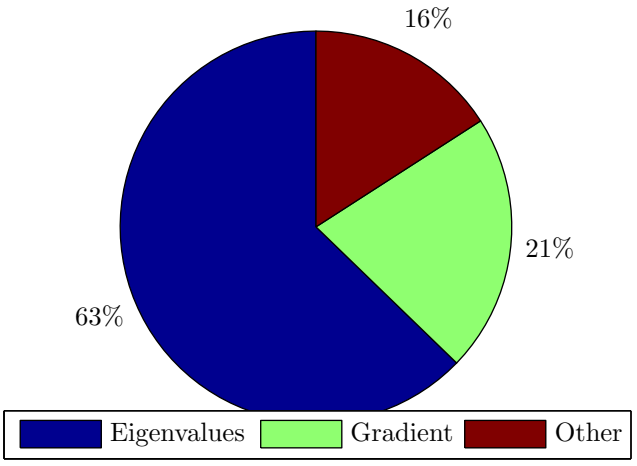


FIGURE 5.4: Most time-demanding steps in the FG algorithm. Total time required: 180.267s

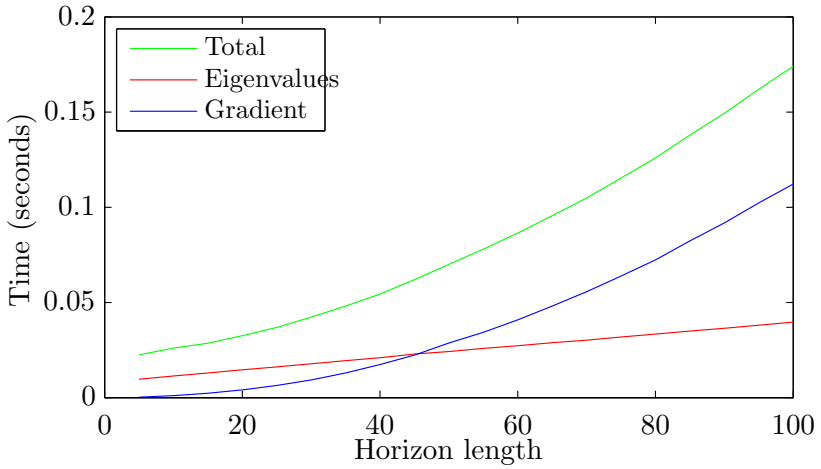


FIGURE 5.5: Time spent for gradient and eigenvalue computation with the matrix dimension.

5.6 Comparison of various eigenvalues algorithms

We are going to see and compare various methods of eigenvalue evaluation in order solve the bottleneck encountered in the previous section. Remember that we are interested only to the maximum and minimum eigenvalue, the `eig` function instead computes all of them, so we wonder if there are other approximated methods that give us only the largest and smallest hopefully in less time time.

The first question that one might ask is: how much is the FG sensible to an error on the eigenvalues? If the method is *robust* enough we could opt for a fast, but imprecise algorithm. We now exploit this problem looking at how much the number of iterations increases when the FG uses wrong eigenvalues.

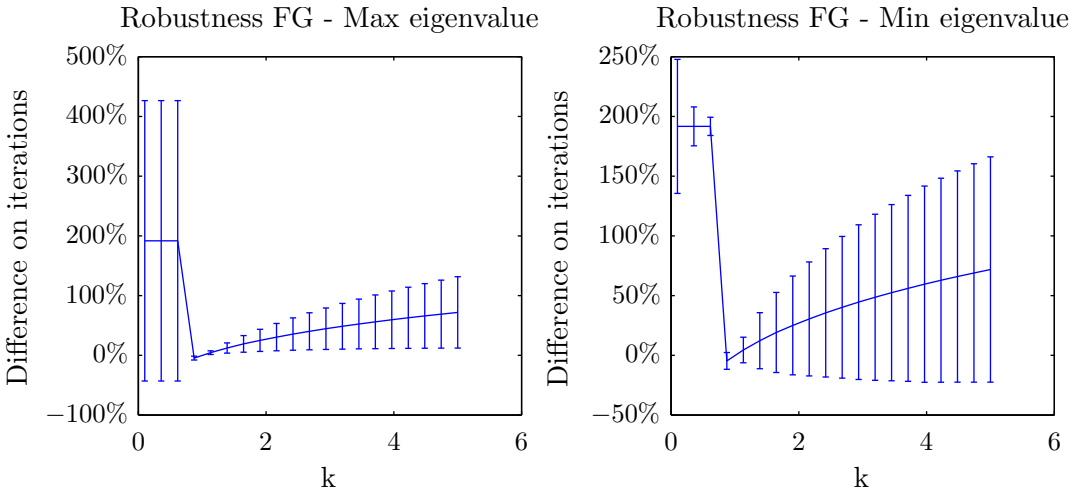


FIGURE 5.6: Difference on iteration of the FG when we have errors on the eigenvalue. The left image refer to the maximum eigenvalue while the minimum is taken as the exact value, the right vice versa. 100 simulation with random system with dimension $n = 10$, the error bar represent the standard deviation. The MHE matrix has been constructed taking the prior weighting matrix P constant and equal to the steady-state value, the vectors of constraints and the f are random.

5.6.1 Robustness of FG to error on the eigenvalues

Note: For this study we need to choose some reference algorithms that are known for their reliability and precision. In particular we need to calculate reference eigenvalues and a reference solution of the problem $f^* \approx f(x^*)$ that have to be good enough in order to use it for the ideal stopping criteria $|f(x^*) - f(x^{(k)})|$. Indeed we have decided not to use any stopping criteria of Par. 5.4 since they *depend* on the eigenvalues, therefore they would have brought another level of error that we are not interested to take into account, since our goal is to study the intrinsic *robustness* of the FG to errors on eigenvalues. Those algorithm are eig for the eigenvalues and active-set of Matlab.

eig: this function uses subroutines of LAPACK library [31]. for real symmetric sparse problem it uses the subroutine DSBTRD that operate on band symmetric matrices computing a tridiagonal reduction $A = QTQ^T$. Since this implement a direct method we assume that it gives a more precise result than our iterative methods.

active-set: we can claim that the active-set algorithm of quadprog function of Matlab gives a good solution whether during the solving process no problem have occurred.

The Fig. 5.6 has been constructed calculating mean value and standard deviation of:

$$\left(\frac{\text{numb. iterations} - \text{reference numb. iterations}}{\text{reference numb. iterations}} \right)_i$$

and

$$\lambda_i = k\lambda_{rif,i}$$

Where i is the simulation number. As we can see in Fig. 5.6, we have the minimum number of iterations when k is slightly less than 1. In both cases if the eigenvalues are larger than the true ones the number of iteration increases but not dramatically, instead if the eigenvalue are taken less than approximately 75% of the real eigenvalues, the number of iteration increases significantly. (Note: for the error on the largest eigenvalue the algorithm has stopped after 5000 iterations without reaching convergence, therefore it may be that in average the iterations required in order to get the same solution are more than 200% respect to exact eigenvalue.)
In conclusion we are surely motivated on finding an eigenvalues approximation with a tolerance of about 10%.
In the following sections we show the result obtained with several eigenvalue computation strategies:

- Tridiagonal reduction (implemented in eig Matlab function);
- Power iteration method with Cholesky decomposition (code written by the author);
- Arnoldi method (implemented in eigs Matlab function);
- Implicit Restarted Block Lanczos method (implemented in irbleigs written by J.Baglama, D.Calvetti, L.Reichel, [32]) ;
- Inverse power iteration method with Cholesky decomposition (code written by the author).

5.6.2 eig,eigs and irbleigs

In Fig. 5.7 is depicted the distribution over 100 different random matrices (notice: not systems) of the time spent for calculating the eigenvalues with the different algorithms. The matrices are 1000×1000 block tridiagonal with the same pattern of the matrix in the MHE problem. As we can see the **eig** function performs better than the other method. We noticed also (but we do not report the results) that if the matrix is symmetric, definite-positive, sparse but *not* block tridiagonal the irbleigs gives the best results (the comparison has been made with a matrix with same dimensions and same percentage of sparsity, but block-tridiagonal).

5.6.3 eig, Inverse Iteration and Power Iteration methods.

The Inverse Iteration Method that we implemented is actually made of other two ingredients:

- Rayleigh quotient;
- Cholesky decomposition.

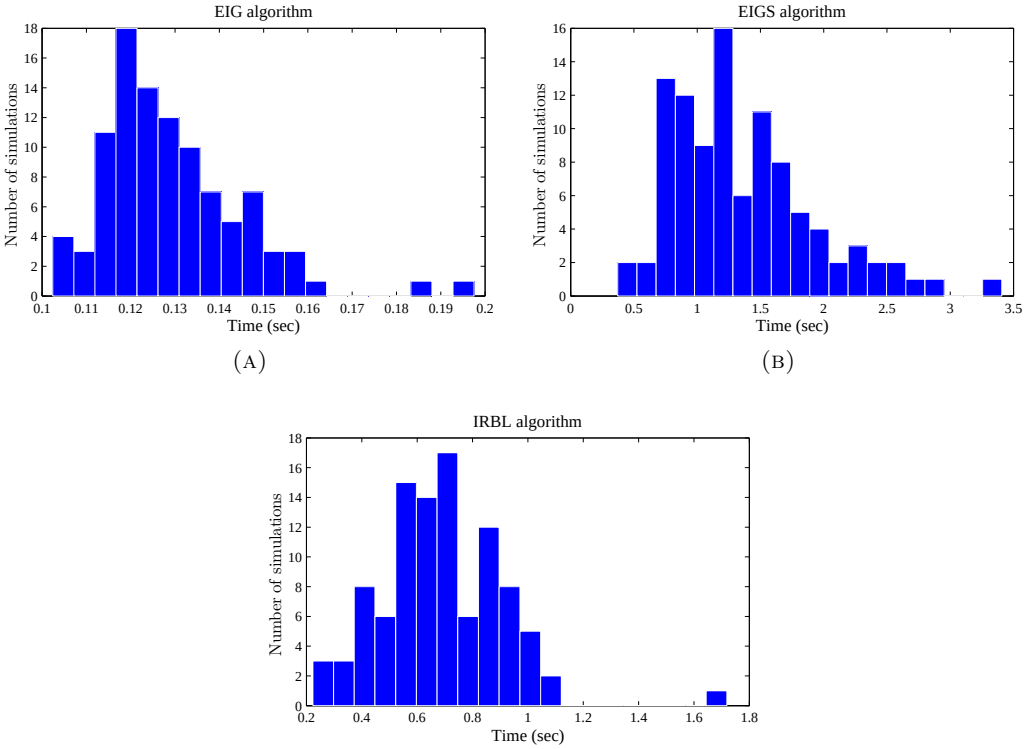


FIGURE 5.7: Statistical time distribution of eigenvalue calculation for eig,eigs and irbleigs. We can see that the eig function is the fastest in all cases.

Rayleigh quotient

In our case we have a symmetric definite-positive $H \in \mathbb{R}^{Nn \times Nn}$, where N is the number of steps, and n the number of states of the system. We assume $\|\cdot\| = \|\cdot\|_2$. Let $\lambda_1, ...; \lambda_{Nm}$ be the (real) eigenvalue of H and $q_1, ..., q_{Nm}$ the associated orthonormal eigenvectors. Let us introduce first the *Rayleigh Quotient*:

$$r(x) = \frac{x^T H x}{x^T x}. \tag{5.15}$$

Notice that if x is the eigenvector, $r(x) = \lambda$ is the corresponding eigenvalue. This formula comes simply from trying to find what scalar α, given an eigenvector x, act more like an eigenvalue in minimizing the value $\|Ax - \alpha x\|$. It turns out that $\alpha = r(x)$. Now let us derive $r(x)$ to see what happen when x is near an eigenvector:

$$\nabla r(x) = \frac{2}{x^T x}(Ax - r(x)x)$$

from this we can see that $\nabla r(x) = 0$, with $x \neq 0$ if x is an eigenvector and $r(x)$ is the corresponding eigenvalue.

Let q_J be one of the eigenvalue of H, since $\nabla r(q_J) = 0$ and the function is smooth except in the origin, we can say that:

$$r(x) - r(q_J) = O(\|x - q_J\|^2)$$

thus the Raylegh quotient converges *quadratically* to the eigenvalue. For this reason its implementation in iterative algorithms is very effective.

Inverse Iteration method

The concept of inverse iteration comes from the *Power Iteration* [17], but it computes an eigenvector, once we have the associated eigenvalue, faster than the power iteration.

Let us take a a scalar μ that is not a eigenvalue of H, we notice that the eigenvectors of $(H - \mu I)^{-1}$ are the same as the eigenvectors of H and the corresponding eigenvalue are $\{\lambda_j - \mu\}$ there $\{\lambda_j\}$ are the eigenvalues of H. Suppose μ is close to the eigenvalue λ_J of H, in this case $(\lambda_J - \mu)^{-1}$ may be much bigger than $(\lambda_j - \mu)^{-1}$ with $j \neq J$. Furthermore, since the power iteration method works better if the difference between the *largest eigenvalue* and the *second largest eigenvalue* is large [17, Pag.205], the idea is to apply the power iteration method to $(H - \mu I)$ in order to amplify this difference and consequently have a faster convergence.

Generally the Inverse iteration method works as follows: The algorithm, like Power

Algorithm 8 Inverse iteration method

Require: $v^{(0)}$,with $\|v^{(0)}\| = 1$ and μ ▷ Random vector
  while $err > toll$ **do**
    $w \leftarrow (H - \mu I)^{-1} v^{(k-1)}$
    $v^{(k)} \leftarrow w/\|w\|$
    $\lambda^{(k)} \leftarrow (v^{(k)})^T A v^{(k)}$ ▷ Rayleigh quotient
    Compute err
  end while
  return λ, v ▷ Eigenvalue and eigenvector

Iteration, has a linear rate of convergence, but we have the advantage that we can choose to which eigenvector the algorithm is going to converge by choosing a value of μ close enough to the associated eigenvalue, furthermore the more μ is close to the eigenvalue the faster is the convergence. In our case we are interested in the maximum eigenvalue and minimum eigenvalue, hence a good choice is $\mu = \|H\|$ for the maximum and $\mu = \|H^{-1}\|$ for the minimum.

We could demonstrate that if λ_J is the closest eigenvalue to μ and λ_K is the second closest, thus $|\mu - \lambda_J| < |\mu - \lambda_K| \leq |\mu - \lambda_j|$ for each $j \neq J$ and $q_J^T v^{(0)} \neq 0$, then the iterates of Algorithm 9 satisfy:

$$\|v^{(k)} - q_J\| = o\left(\left|\frac{\mu - \lambda_J}{\mu - \lambda_K}\right|^k\right), \qquad |\lambda^{(k)} - \lambda_J| = o\left(\left|\frac{\mu - \lambda_J}{\mu - \lambda_K}\right|^{2k}\right)$$

as $k \to \infty$. So the method converges *quadratically* to the eigenvalue.

Cholesky decomposition

The Cholesky decomposition (or factorization) is a decomposition of a symmetric (more generally Hermitian) positive-definite matrix H into a product of a lower triangular matrix and its transpose (more generally conjugate transpose) $H = LL^T$.

This decomposition is used for solving systems of linear equations in the form $Ax = b$. Indeed after the decomposition we can use backward and forward substitution, but twice faster than the LU decomposition. For *dense* matrices the cost of factorization is $1/3o(n^3)$. The cost for solving the linear problem is also $1/3o(n^3)$ while the LU needs $2/3o(n^3)$. But if the matrix is sparse like in our case, the number of flops can be much less than $1/3o(n^3)$ depending on the degree of sparsity.

The idea is to use this decomposition in order to calculate w in the Algorithm 9, since we can see $w = (H - \mu I)^{-1} v^{(k-1)}$ as $(H - \mu I)w = v^{(k-1)}$

Why not Rayleigh Quotient Iteration method? The reader might ask why we have not used the *Rayleigh Quotient Iteration method*(RQI) [17]. This method consists in updating μ at each iteration with the Rayleigh quotient λ instead of keeping it constant and it would have led us to have a *cubic* convergence to the eigenvalue, instead of quadratic. The reason is that Cholesky factorization is applicable only for positive semi-definite matrices and in general the matrix $(H - \mu I)$ could be not positive semi-definite, but when $\mu = \|H|$ or $\|H^{-1}\|$ it is negative definite and becomes positive definite if we take $-H + \mu I$. Notice that the sign does not represent a problem. We have opted for the Cholesky decomposition renouncing the advantage of RQI because, while the number of iterations needed in order to calculate the eigenvalues is modest even with Inverse Iteration (about 100 for the maximum and 10 for the minimum in the cases we studied) the advantage on computing $(H - \mu I)w = v^{(k-1)}$ with practically $o(n)$ flop, as we are going to see in practice in the next chapter, is much more significant.

We report in Algorithm 9 the pseudocode implemented in the m file invit that we used.

Algorithm 9 invit - Inverse iteration method with Cholesky factorization

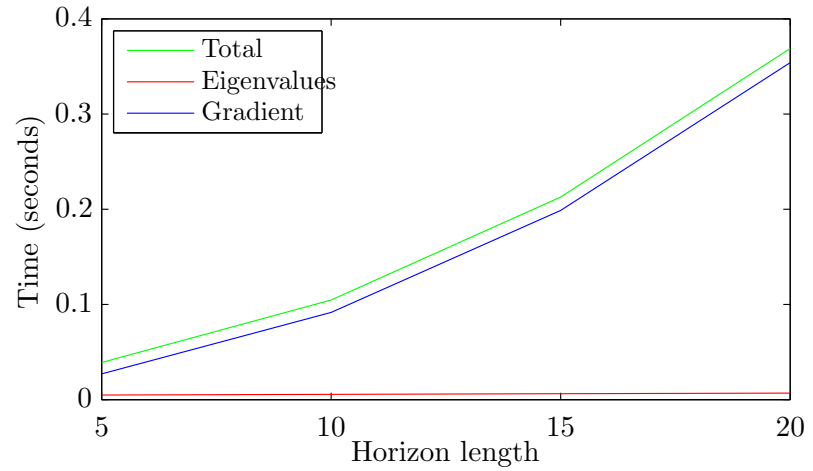
Require: *toll* ▷ Tolerance
$v^{(0)} \leftarrow \text{rand}\ (h, 1)$ ▷ Random vector
$c \leftarrow \|H\|$
$cH \leftarrow \text{chol}(-H + cI)$ ▷ Cholesky decomposition
while $err > toll$ **do**
$w^{(k)} \leftarrow v^{(k-1)}/|v^{(k-1)}|$
$v^{(k)} \leftarrow -cH \backslash cH^T \backslash w^{(k)}$
$L^{(k)} \leftarrow (v^{(k)})^T H v^{(k)}/|q|$ ▷ Rayleigh quotient
$err \leftarrow |(Hv - L^{(k)}v)|$ ▷ Eigenvalue definition
end while
return L ▷ Maximum eigenvalue
$v^{(0)} \leftarrow \text{rand}\ (h, 1)$ ▷ Random vector
$cH \leftarrow \text{chol}(H)$ ▷ Cholesky decomposition
while $err > toll$ **do**
$w^{(k)} \leftarrow v^{(k-1)}/|v^{(k-1)}|$
$v^{(k)} \leftarrow cH \backslash cH^T \backslash w^{(k)}$
$\mu^{(k)} \leftarrow |q|/(v^T q)$ ▷ Rayleigh quotient
$err \leftarrow |(Hv - \mu^{(k)}v)|$ ▷ Eigenvalue definition
end while
return μ ▷ Minimum eigenvalue

In Fig 5.8 compare the results obtained with the three methods on a random system with 10 states while increasing the horizon length. Since the difference in computational time is remarkable we have not proceeded with statistical study on other random systems. Notice that the x-axis of 5.8a reaches a matrix dimension of 200×200, conversely in the other case is 1000×1000.

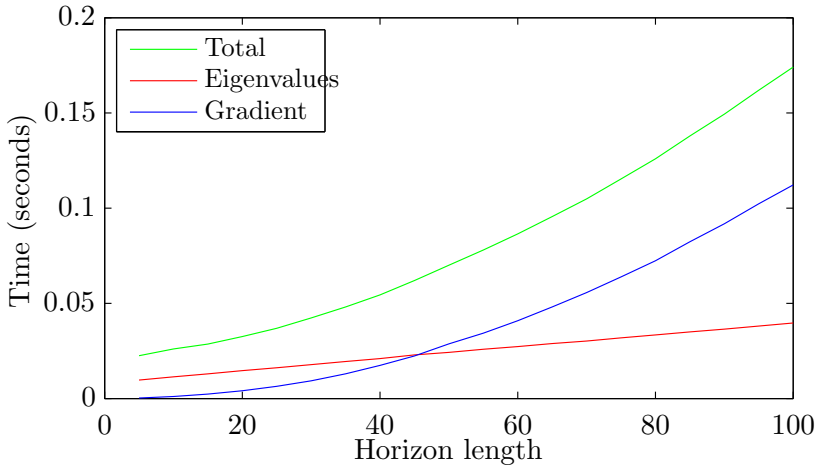
We could have expected that the Power Iteration takes more time for the reasons outlined previously in the chapter.

5.7 Performance of Fast Gradient with Inverse Iteration and Cholesky factorization (FGIIC).

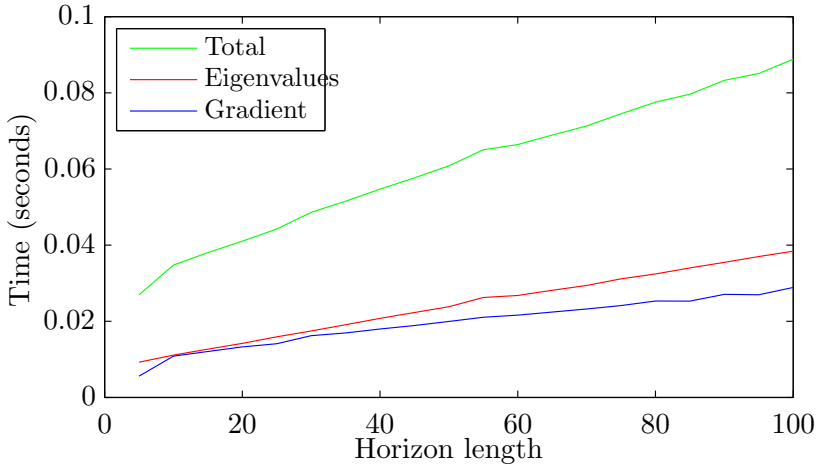
The Inverse Iteration method with Cholesky factorization has shown the best performance in terms of computational time. To make sure that the problem solved via this method gives the same result as with the eig method, we report in Fig. 5.9 the differences on the solutions. As we can see the solution are sufficiently close to each other. We want to test our algorithm with 100 random system. The FG algorithm has been set up as in Tab.5.1. The first important thing that we notice is that not all the simulations have converged within the 5000 iterations. In Fig. 5.10 we report with red crosses the non-converged simulations and in green crosses the converged ones. Notice that for non-converged we mean the simulations that have not converged at least in one



(A) Power iteration with Cholesky factorization.

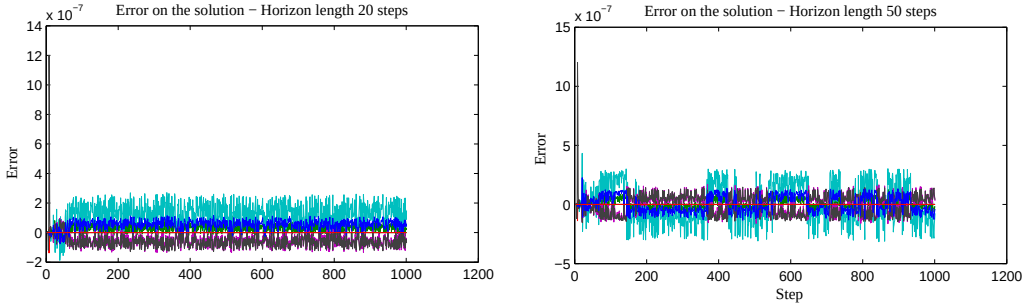


(B) eig.



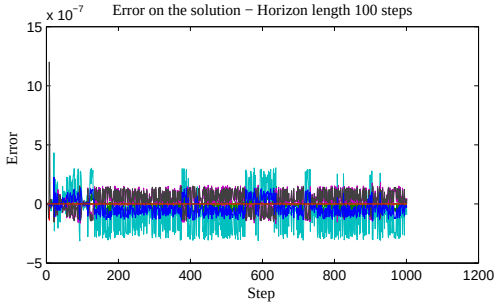
(C) Inverse iteration with Cholesky factorization.

FIGURE 5.8: Time spent for a MHE problem varying horizon length with different algorithm for the eigenvalue calculation. The time refers to a time step, thus for solving a a complete optimization.



(A) Horizon length 20 steps.

(B) Horizon length 50 steps.



(C) Horizon length 100 steps.

FIGURE 5.9: Error on the 10 states with different horizon length when we use Inverse Iteration method respect to the same problem solved with eig, namely $(\hat{x}_{i,j} - x_{i,j})/\hat{x}_{i,j}$ where $\hat{x}$ is the solution with eig and x the solution with Inverse iteration, i in the state and j is the simulation step. We can see how the solutions are sufficiently close to each other along all the simulation.

TABLE 5.1: Problem set up. Refer to Alg.7 and 9

Fast Gradient	
Max iterations	5000
Tolerance	10^{-8}
Stopping criteria	Tailored
Starting point	Warm start
Inverse iteration	
Max iteration	Till converg.
Tolerance on eigenvalues	10^{-3}
Characteristics	Cholesky decomp.

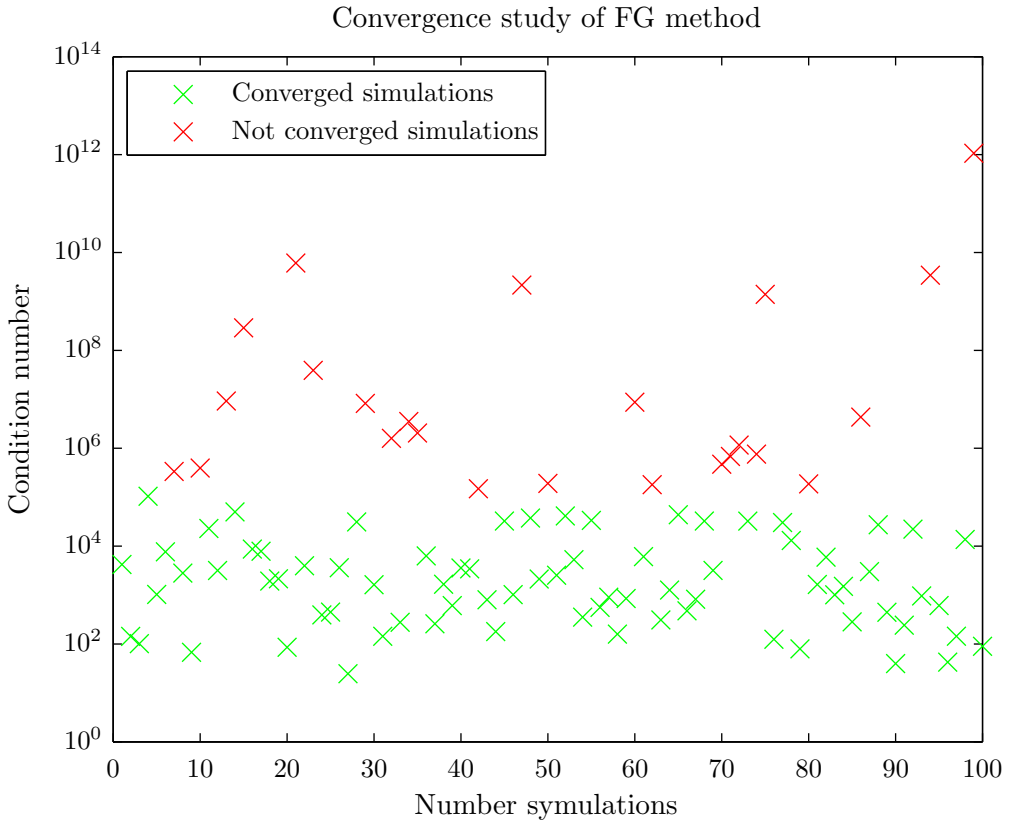


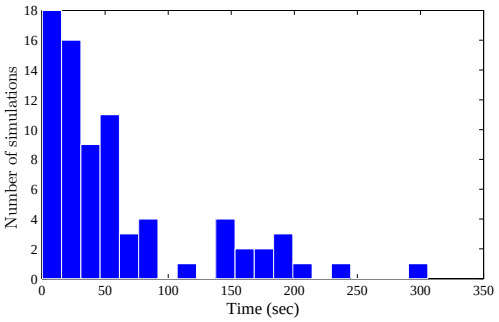
FIGURE 5.10: The green crosses represent the converged simulations. 77 simulations over 100 have converged. The maximum condition number of the converged ones is 1.0456×10^5 while the minimum condition number of the non-converged one is 1.4883×10^5.

time step. In the y-axis we report the mean condition number of the problem along the simulation time (however it reaches a constant value after a few steps). We notice that the algorithm does not converge when the condition number is greater than $\approx 10^5$.

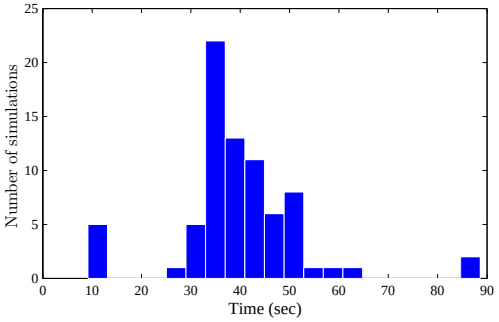
In the case of the system in Fig. 5.8c the bottleneck is the gradient computation, we can see if it is always the case through measuring the time spent in the different step for the converged simulation. In Fig. 5.11 are depicted the results of these simulations and in Tab. 5.2 some characteristic data of the time distributions. As we can see in some cases the eigenvalues computation still reveals to be the bottleneck.

TABLE 5.2: Time distribution for the MHE problem solved through FG with Inverse iteration

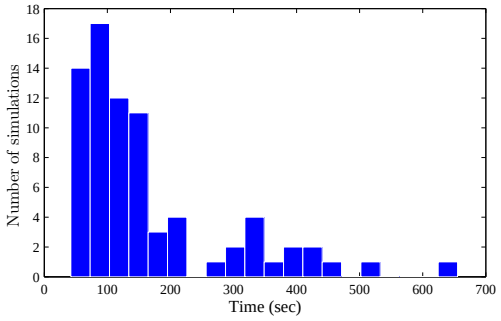
Statistical data (in seconds)			
	Gradient	**Eigenvalue**	**Total**
Mean	62.29	39.72	167.59
Min	0.21	9.14	42.065
Max	305.77	88.64	655.19
Median	38.87	38.37	116.6764



(A) Gradient time distribution.



(B) Eigenvalue time distribution.



(C) Total time distribution.

FIGURE 5.11: Time distribution. MHE problem of 77 random systems solved with an horizon length of 100 steps, using Fast Gradient method and Inverse iteration method.

We know want to explore in which cases the Gradient step is more time demanding than the eigenvalue step. As we notice in Fig. 5.12 when the condition number is smaller than 10^2 the gradient computation is faster than the eigenvalue computation. .

5.7.1 Conclusions

Since the majority of matrices of our interest have an high condition number, the bottleneck in our cases remains the *gradient computation*. We tried to decrease the time spent during the matrix-vector multiplication Hx using, instead of a H matrix sparse block-tridiagonal, a full "skinny" matrix (where we stored the blocks in the diagonals as columns) end multiplying this matrix *blockwise* with the vector x. It turned out that it takes longer, probably because the algorithm used by Matlab of sparse-matrix vector multiplication uses C-language.

5.7.2 Final algorithm set-up

The best, and therefore final, version of the algorithm uses:

- Inverse Iteration method with Cholesky decomposition and Rayleigh quotient for calculating the eigenvalue;
- Stopping criteria tailored for Fast Gradient.

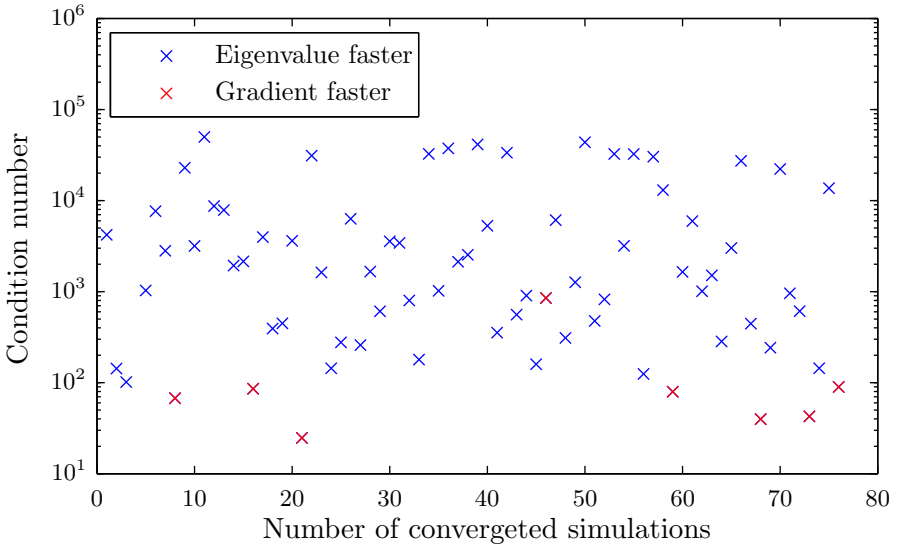


FIGURE 5.12: Bottleneck of the algorithm in function of the condition number. The blue crosses represent where the gradient is slowest step, therefore it is the bottleneck while the red crosses are where the eigenvalue computation is the slowest step.

Chapter 6

Examples of application

6.1 Upflow Anaerobic Sludge Blanket reactor (UASB)

The same example has been used in [33] for estimating the states with an Unscendend Kalman Filter (UKF). The UKF works directly on nonlinear systems but cannot take into account constraints on the variable, which in this case are important since we have some limit values that, if exceed, can lead to unsafe conditions [34]. We here resume the main information about the system and lately apply the MHE and MPC algorithm. Anaerobic digestion (AD) of organic substrate can be used to produce biogas consisting mainly of methane and carbon dioxide. If the AD reactor works well, the produced gas can be used for biogas combustible and, if in the digestate are not present toxic compounds it can be used as a fertilizer. The UASB type reactors allow for a relatively

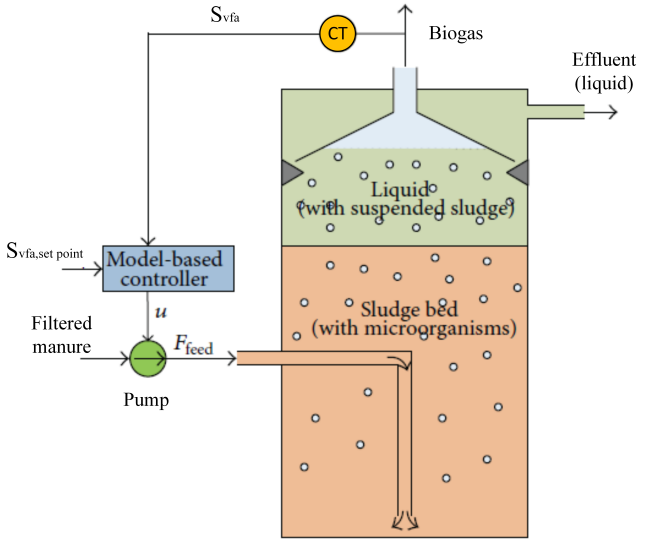


FIGURE 6.1: Upflow Anaerobic Sludge Blanket reactor (UASB) - Image taken from [33] (partially modified).

high load rates and/or small reactor volumes [35]. This reactors have a large retention time and this confers them high effectiveness.

We can apply several control policy to the plant, one aims to reach a constant set-point of methane flow given a requested power production. Another control policy is to keep the reactor in a safe operating point, namely the concentration of volatile fatty acids (VFA) is not above a certain concentration.

We use a modified Hill model [36] to simulate the system and we linearise analytically around the steady state value.

$$\dot{S}_{bvs} = (B_0 S_{vs_{in}} - S_{bvs}) \frac{F_{feed}}{V} - \mu k_1 X_{acid} \tag{6.1}$$

$$\dot{S}_{vfa} = (A_f B_0 S_{vs_{in}} - S_{vfa}) \frac{F_{feed}}{V} + \mu k_2 X_{acid} - \mu_c k_3 X_{meth} \tag{6.2}$$

$$\dot{X}_{acid} = \left(\mu - K_d - \frac{F}{bV}\right) X_{acid} \tag{6.3}$$

$$\dot{X}_{meth} = \left(\mu_c - K_{dc} - \frac{F}{bV}\right) X_{meth} \tag{6.4}$$

$$\begin{aligned} \mu &= \mu_m \frac{S_{bvs}}{K_s + S_{bvs}} \\ \mu_c &= \mu_{mu} \frac{S_{vfa}}{K_{sc} + S_{vfa}} \\ \mu_m = \mu_{mc} &= 0.013 T_{reac} - 0.129 \quad \text{for} \quad 20°C < T_{reac} < 60°C. \end{aligned} \tag{6.5}$$

in which S_{bvs} is the concentration of biodegradable volatile solids (BVS) ($g\ BVS/L$), S_{vfa} is the concentration of VFA ($g\ VFA/L$) in the reactor and the only measured variable since there exist sensors for on-line measurement [37], [38], F_{feed} is the feed flow (L/d) and the only manipulated variable, X_{acid} is the concentration of acidogens (g acidogens $/L$), X_{meth} is the concentration of methanogens (g methanogens $/L$), μ and μ_c are reaction rates expressed in Eq. 6.5, V is the reactor volume (L), b is a parameter (d/d), A_f is an acidity constant (($g\ VFA/L$)/($g\ BVS/L$)), B_0 a biodegradability constant (($g\ VFA/L$)/($g\ VS/L$)), k_1, k_2, k_3, k_4 are yield constant (g BVS/(g acidogens/L)), g VFA/(g acidogens/L)), g VFA/(g methanogens/L)) respectively), K_d is the specific death rate of acidogens (d^{-1}), K_dc is the specific death rate of methanogens (d^{-1}), K_s is the Monod half-velocity constant for acidogens ($g\ BVS/L$) and K_{sc} is the Monod half-velocity constant for methanogens. Notice that the reaction temperature T_{reac} is kept constant at $35°C$ with a separate temperature controller, therefore we assume it is constant [39].

The state vector is chosen as $\hat{x} = [S_{bvs}, S_{vfa}, X_{acid}, X_{meth}]$, while the control input is $u = F_{feed}$. Linearising and discretising the system with sampling time of 0.01 *day* we

TABLE 6.1: Parameters and steady state variable values. For units see below.

Parameter	Value	Variable	Steady state value
A_f	0.69	S_{vfa}	0.8
b	2.90	F_{feed}	35.3
B_0	0.25	F_{meth}	174.2
k_1	3.89	S_{bvs}	1.14
k_2	1.76	X_{acid}	1.8
k_3	31.7	X_{meth}	0.39
K_d	0.02	$S_{vs_{in}}$	30.2
K_{dc}	0.02	T_{reac}	35
K_s	15.5		
K_{sc}	3		
V	250		

obtain the following A, B and C matrix:

$$A = \begin{bmatrix} 0,9769 & 0 & -0,0264 & 0 \\ 0,0040 & 0,9068 & 0,0115 & -0,2075 \\ 0,0072 & 0 & 0,9999 & 0 \\ 1,6125 \cdot 10^{-5} & 0,0076 & 4,6772 \cdot 10^{-5} & 0,9991 \end{bmatrix} \quad B = \begin{bmatrix} 0,0014 \\ 0,0017 \\ -0,0002 \\ -4,6379 \cdot 10^{-5} \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$

In [40] is reported that a concentration of S_{vfa} that exceeds a value of $0.8g\ VFA/L$ can lead to a reaction failure for this reason we impose an upper limit on the S_{vfa} of 0.8. Furthermore we assume that we can have a maximum flow rate F_{feed} that we can supply of 200 L/d. There are no significant upper limits on other variables. Moreover we straightforwardly choose a lower limit 0 for all the states.

Target Calculation and MPC

Since we want to lead the S_{vfa} to a safety set point we are dealing with a control problem. For matter of simplicity in this section we consider a perfect model, namely there is no error between the mathematical model and the real system. The set point calculation in presence of constraints can be formulated as follows:

$$\begin{aligned} \min \ & \|Cx_s - y_{sp}\|_T^2 \\ & x_s = Ax_s + Bu_s \\ & x_{min} \leq x_s \leq x_{max} \\ & u_{min} \leq u_s \leq u_{max} \end{aligned}$$

in which $(\cdot)_s$ indicates the (new) steady-state value and $y_{sp} = S_{vfa} = 0.70$, $T \in \mathbb{R}^{p \times p}$ positive definite matrix. Notice that in this case it is a scalar positive value since $p = 1$.

After computing the steady-state values we can proceed with the MPC problem, defining $\tilde{x} = \hat{x} - x_s$ and $\tilde{u} = \hat{u} - u_s$ we solve the QP problem:

$$\begin{aligned}
&\min \frac{1}{2} \sum_{j=0}^{N_{MPC}-1} \left(\|\tilde{x}_j\|^2_{Q_{MPC}} + \|\tilde{u}_j\|^2_{R_{MPC}} \right) + \|\tilde{x}_{N_{MPC}}\|_{P_f} \\
&\tilde{x}_0 = \hat{x}* \\
&\tilde{x}_{j+1} = A\tilde{x}_j + B\tilde{u}_j \\
&x_{min} - x_s \leq \tilde{x}_j \leq x_{max} - x_s \\
&u_{min} - u_s \leq \tilde{u}_j \leq u_{max} - u_s
\end{aligned}$$

in which $\hat{x}*$ is the estimate of x_{k+1} of the MHE problem at time k, Q_{MPC}, R_{MPC} and P_f are positive definite matrices, in particular P_f has been found solving the Discrete Algebraic Riccati Equation (DARE) on the matrices A, B, Q_{MPC} and R_{MPC}:

$$A^T P_f A - P_f - A^T P_f B (B^T P_f B + R_{MPC})^{-1} B^T P_f A + Q_{MPC} = 0.$$

As we said in § 2.1.1 we apply to the plant only the first control input u_0 resulting from the MPC problem then the MHE and MPC problem will be repeated at time step $k+1$. The closed loop control system is shown in Fig. 6.2

Note: the standard way to implement a model-based controller with an estimator is to use the estimation of the variable x_k to calculate the control action at time u_k. As the reader may have noticed, we do not use this approach: our MPC uses the estimate of x_{k+1} state to compute the u_{k+1} control input. This can be useful when we have to take into account *delays* between the estimation and the application of the control action, indeed in this case it is possible to anticipate the state and to apply the input right when that input is needed, namely at time $k+1$.

6.1.1 Results

In Fig. 6.3 are reported the simulation results. The green line represent the solution of the problem $x_{k+1} = Ax_k + Bu_k$, the blue $x_{k+1} = Ax_k + Bu_k + w_k$ and the red crosses are the state estimated by MHE.

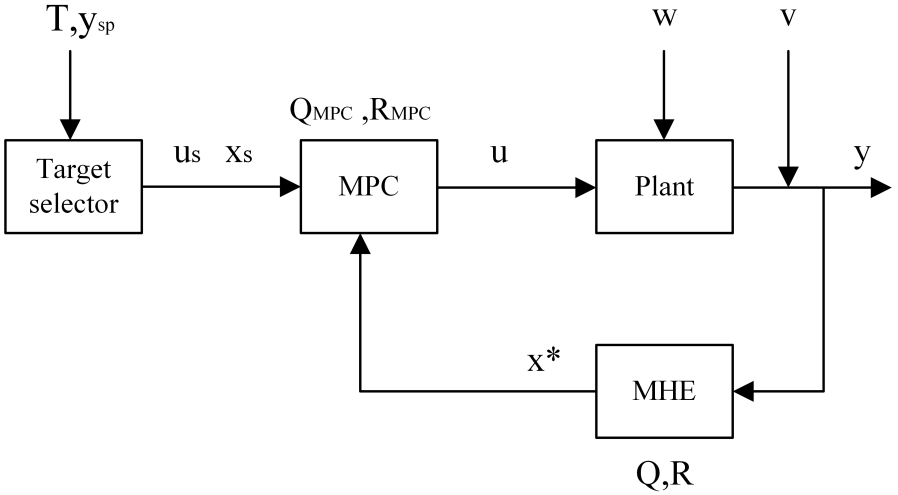


FIGURE 6.2: Closed loop control system. Moving Horizon Estimation with Model Predictive Control.

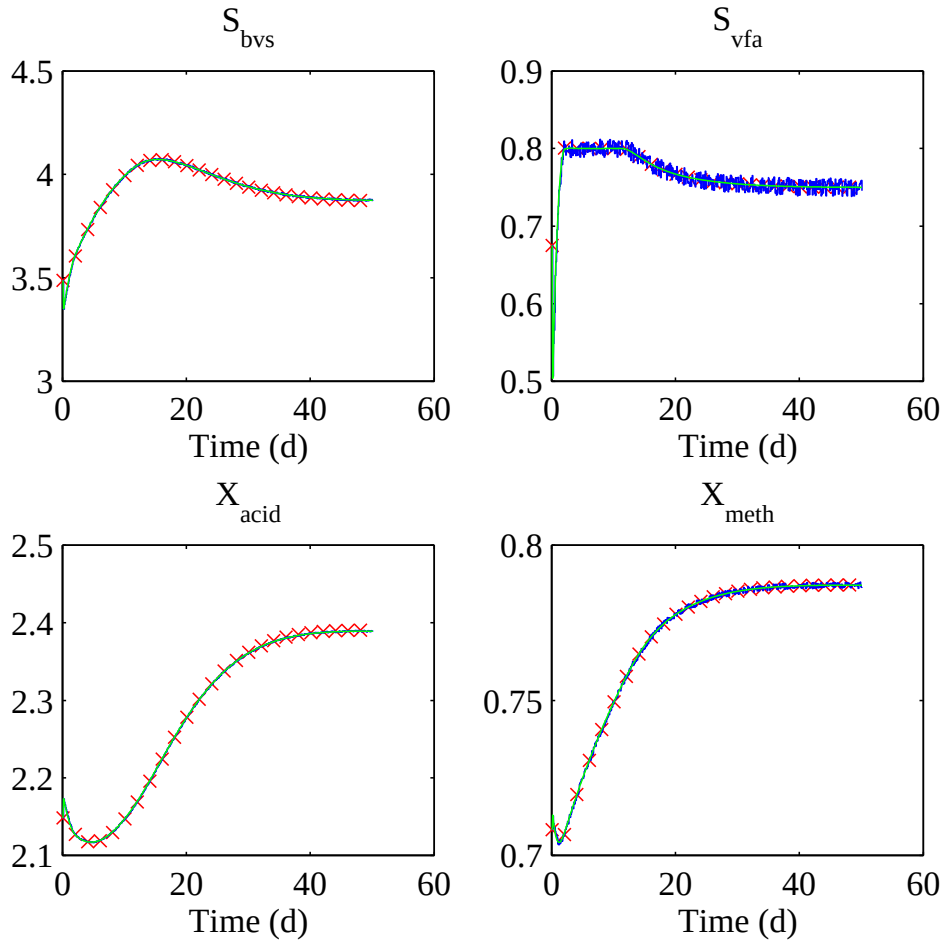


FIGURE 6.3: UASB estimated states. Red crosses: estimated states, green lines: real states, blue lines: disturbed states.

TABLE 6.2: Performance comparison in term of IAE index: Kalman Filter and Moving Horizon Estimation.

IAE Index		
State	**KF**	**MHE**
S_{bvs}	1.0587	0.8148
S_{svfa}	0.0273	0.0264
X_{acid}	0.3543	0.3831
X_{meth}	0.1103	0.0309

Comparison with Kalman Filter We applied at the same problem the KF. As a tool of comparison we use the IAE index for time-discrete cases:

$$IAE = \sum_{j=0}^{steps} |e_j|.$$

in which $e_k = x_k^* - x_k$ where x_k^* is the estimate of one of the methods and x_k is the real state, namely the state obtained from the model without noises. In Tab. 6.2 are reported the results. In absence of constraints, the two methods should give the same results since we used a quadratic cost function and we update the prior weighting matrix with KF [18], but since we used the information given by the constraints on the measured variable we obtained better results.

6.2 Two reactor chain with flash separator

For the practical application we are going to consider two CSTR followed by a nonadiabatic flash. The same problem has been consider by Venkat, Rawlings and Wright in [41] for a MPC problem. The scheme of the plant is shown in Fig. 6.4. The reaction consists of:

$$A \xrightarrow{k_1} B \xrightarrow{k_2} C$$

where B is our product and C is a unwanted side product. These reaction take place in both reactors. The product of the CSTR-2 is sent to the flash to separate the excess A which has the highest relative volatility from B and C. The vapour phase rich in A is partially purged and the remain part is condensed back to the CSTR-1. States and measured output are disturbed with a bounded white noise with zero mean. Let us assume that we cannot measure the temperature of the flash T_b and the mass fraction of A in the CSTR-1 x_{Ar} and we want to estimate them. Applying the first principle to the parts of the plant we obtain the Eqs. 6.6, 6.7, 6.8.

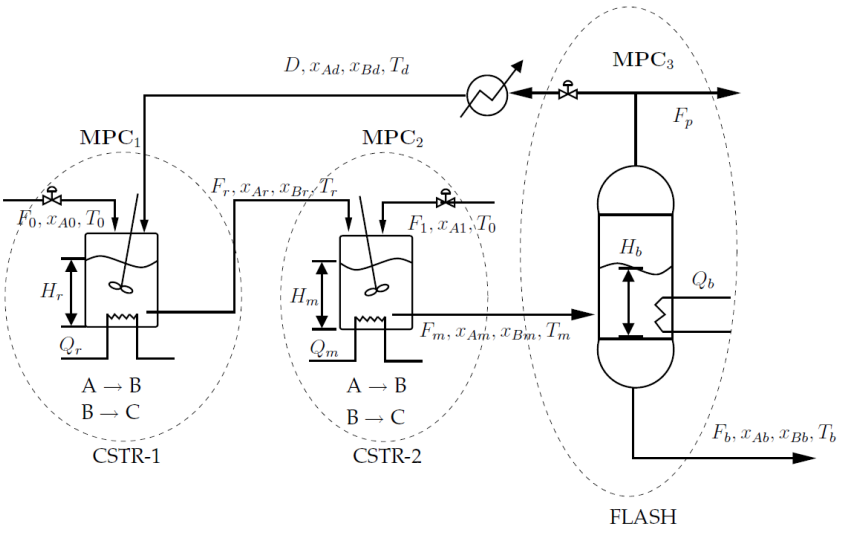


FIGURE 6.4: Scheme of the plant. Image taken from [41].

CSTR-1

$$\frac{dH_r}{dt} = \frac{1}{\rho A_r}[F_0 + D - F_r], \tag{6.6a}$$

$$\frac{dx_{A_r}}{dt} = \frac{1}{\rho A_r H_r}[F_0(x_{A_0} - x_{A_r})] + D(x_{A_d} - x_{A_r})] - k_{1_r} x_{A_r} \tag{6.6b}$$

$$\frac{dx_{B_r}}{dt} = \frac{1}{\rho A_r H_r}[F_0(x_{B_0} - x_{B_r})] + D(x_{B_d} - x_{B_r})] - k_{1_r} x_{A_r} - k_{2_r} x_{B_r} \tag{6.6c}$$

$$\frac{dT_r}{dt} = \frac{1}{\rho A_r H_r}[F_0(T_0 - T_r) + D(T_d - T_r)] - \frac{1}{C_p}[k_{1_r} x_{A_r} \Delta H_1 + k_{2_r} x_{B_r} \Delta H_2] + \frac{Q_r}{\rho A_r C_p H_r} \tag{6.6d}$$

CSTR-2

$$\frac{dH_m}{dt} = \frac{1}{\rho A_m}[F_r + F_1 - F_m], \tag{6.7a}$$

$$\frac{dx_{A_m}}{dt} = \frac{1}{\rho A_m H_m}[F_r(x_{A_r} - x_{A_m})] + D(x_{A_1} - x_{A_m})] - k_{1_m} x_{A_m} \tag{6.7b}$$

$$\frac{dx_{B_m}}{dt} = \frac{1}{\rho A_m H_m}[F_r(x_{B_r} - x_{B_m})] + D(x_{B_1} - x_{B_m})] - k_{1_m} x_{A_m} - k_{2_m} x_{B_m} \tag{6.7c}$$

$$\frac{dT_m}{dt} = \frac{1}{\rho A_m H_m}[F_r(T_r - T_m) + F_1(T_0 - T_m)] - \frac{1}{C_p}[k_{1_m} x_{A_m} \Delta H_1 + k_{2_m} x_{B_m} \Delta H_2] + \frac{Q_m}{\rho A_m C_p H_m} \tag{6.7d}$$

Nonadiabatic flash

$$\frac{dH_b}{dt} = \frac{1}{\rho_b A_b}[F_m - F_b - D - F_p], \tag{6.8a}$$

$$\frac{dx_{A_b}}{dt} = \frac{1}{\rho_p A_p H_p}[F_m(x_{A_m} - x_{A_b})] + (D + F_p)(x_{A_d} - x_{A_b}] \tag{6.8b}$$

$$\frac{dx_{B_b}}{dt} = \frac{1}{\rho_p A_p H_p}[F_m(x_{B_m} - x_{B_b})] + (D + F_p)(x_{B_d} - x_{B_b}] \tag{6.8c}$$

$$\frac{dT_b}{dt} = \frac{1}{\rho_b A_b H_B}[F_m(T_m - T_b)] + \frac{Q_b}{\rho_b A_b C_{p_b} H_b} \tag{6.8d}$$

$$F_r = k_r\sqrt{H_r} \qquad F_m = k_m\sqrt{H_m} \qquad k_{1_r} = k_1^* \exp\left(\frac{-E_1}{RT_r}\right)$$

$$F_b = k_b\sqrt{H_b} \qquad x_{C_r} = 1 - x_{A_r} - x_{B_r} \qquad k_{1_r} = k_2^* \exp\left(\frac{-E_2}{RT_r}\right)$$

$$x_{C_m} = 1 - x_{A_m} - x_{B_m} \qquad x_{C_b} = 1 - x_{A_b} - x_{B_b} \qquad k_{1_m} = k_1^* \exp\left(\frac{-E_1}{RT_m}\right)$$

$$x_{A_d} = \frac{\alpha_A x_{A_b}}{\sigma} \qquad x_{B_d} = \frac{\alpha_B x_{B_b}}{\sigma} \qquad k_{2_r} = k_2^* \exp\left(\frac{-E_2}{RT_m}\right)$$

$$x_{C_d} = \frac{\alpha_C x_{C_b}}{\sigma} \qquad \sigma = \alpha_A x_{A_b} + \alpha_B x_{B_b} + \alpha_C x_{C_b}$$

Linearisation

The system is nonlinear, therefore we linearise it around the steady state values reported in Tab. 6.3 with *numerical differentiation*. This technique is an approximation of the derivate. Namely if we have a system:

$$\dot{x}(t) = f(x_k, u_k) + w(t)$$

with $x(t) \in \mathbb{R}^{12}$, $u(t) \in \mathbb{R}^6$ and $f \in \mathbb{R}^{12} \times \mathbb{R}^6 \to \mathbb{R}^{12}$. We want to linearise it in the form:

$$\dot{x}(t) = Ax(t) + Bu(t) + w(t)$$

Let $x_s \in \mathbb{R}^{12}$ be the steady state vector and u_s the steady state control input, we can find the A matrix:

$$x^*_{(j)} = (0, ..., \overbrace{\epsilon}^{\text{j-position}}, ..., 0)$$

$$A_{(j)} = \frac{f(x_s + x^*_{(j)}, u_s)}{\epsilon}$$

TABLE 6.3: Steady state values that correspond to the maximum yield of B

Steady state values		
$\rho = \rho_b = 0.15 kgm^{-3}$	$\alpha_A = 3.5$	$\alpha_B = 1.1$
$\alpha_C = 0.5$	$k_1^* = 0.02 sec^{-1}$	$k_2^* = 0.018 sec^{-1}$
$A_r = 0.3 m^2$	$A_m = 3 m^2$	$A_b = 5 m^2$
$F_0 = 2.667 kgsec^{-1}$	$F_1 = 1.067 kgsec^{-1}$	$D = 30.74 kgsec^{-1}$
$F_p = 0.01D$	$T_0 = 313K$	$T_d = 313K$
$C_p = C_{p_b} = 25 kJ(kgK)^{-1}$	$Q_r = Q_m = Q_b = -2.5 kJsec^{-1}$	$x_{A_0} = 1$
$x_{B_0} = x_{C_0} = 0$	$x_{A_1} = 1$	$x_{B_1} = x_{C_1} = 0$
$\Delta H_1 = -40 kJkg^{-1}$	$\Delta H_2 = -50 kJkg^{-1}$	$\frac{E_1}{R} = \frac{E_2}{R} = 150K$
$k_r = 2.5 kgsec^{-1}m^{-1/2}$	$k_m = 2.5 kgsec^{-1}m^{-1/2}$	$k_b = 1.5 kgsec^{-1}m^{-1/2}$

where $A_{(j)}$ is the $j-$th column of A. We do the same thing for B:

$$u^*_{(j)} = (0, ..., \overbrace{\epsilon}^{\text{j-position}}, ..., 0)$$
$$B_{(j)} = \frac{f(x_s, u_s + u^*_{(j)})}{\epsilon}$$

The linearisation quality has been tested looking at the differences in the output between linearised model end the solution of the ODE (computed with Implicit Euler). Then the system is discretized with sampling time of 0.1s. The system is open loop stable so we do not apply any control action and it is also observable.

Measured variables

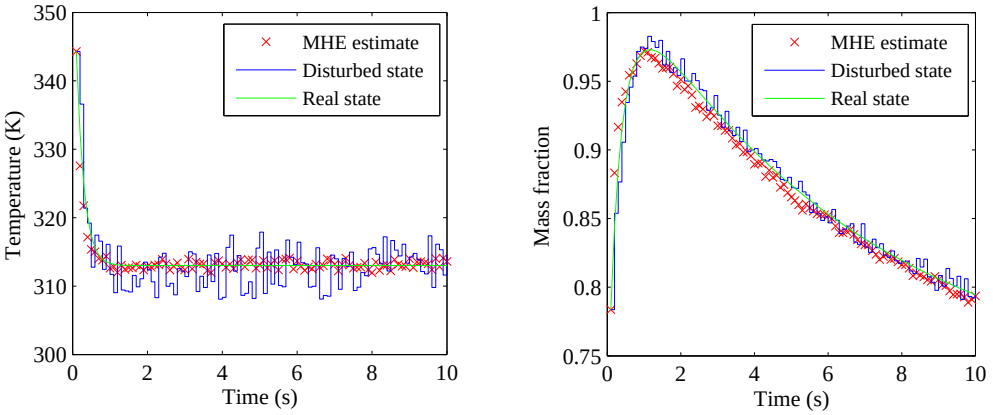
The measured variables are the level of liquid in the reactors H_r and H_m, the exit mass fractions x_{A_r}, x_{B_r} and x_{A_m}, x_{B_m} , the temperatures T_m and T_r and the same thing for the flash where we measure H_b,T_b and x_{A_b}, x_{B_b}. We assume that these variables are disturbed with a white noise.

Controlled variables(CVs)

For the CSTR-1 are H_r and T_r, for CSTR-2 are H_m and T_m. For the flash are the level in the flash H_b and the temperature T_b.

Manipulated variables (MVs)

For the CSTR-1 are the feed flowrate F_0 and the cooling duty Q_r, the same for CSTR-2 F_m and Q_m. For the flash are the recycle flowrate D and the cooling duty Q_b.



(A) *Flash out temperature* T_b. (B) *Mass fraction* x_{Ar}.

FIGURE 6.5: Results of the estimation in the chemical plant section considered.

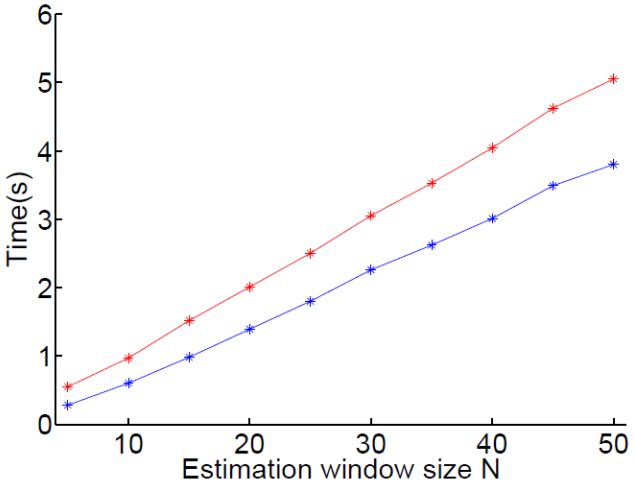


FIGURE 6.6: Computational times. Blue: fast gradient only, Red: overall algorithm.

6.2.1 Results

We report the graphs that show the dynamics of the variables that we have assumed not measurable. The horizon length is of 10 steps. The MHE estimate is a quite good approximation of the real state, which is the solution of the ODE. In Fig.6.6 is shown the computational time of the whole algorithm and only the FG obtained with different horizon lengths (called here Estimation window) on a simulation of 500 time-steps. Notice that the time requested grows linearly with the horizon length (the dimension of the H matrix grows proportionally). For comparison we carried out the same simulation with the *quadprog* solver of Matlab [1] on an Intel Core i7 - 4770 CPU 3.4 GHz of clock frequency. We obtained the following results: for a horizon length of 5 the Active set (the

[1] Since the build-in solvers of Matlab work in C++ code, for having a fair comparison the FG algorithm was rewritten in C++. Thanks to Markus Kögel who helped me with this.

fastest among the method implemented in quadprog in this case) takes about $1.95s$ for solving the entire problem while the FG takes $0.75s$, this means that the FG is 2.6 time faster. For an horizon of 50 the interior point method (the fastest among the method implemented in quadprog in this case) takes about $18.4s$ while the FG about $5s$, thus 3.7 time faster.

Chapter 7

Summary and Conclusions

We have implemented a simple and fast algorithm for solving MHE optimization problem for linear, time-invariant systems using the Fast Gradient method (§ 4.5) and taking advantage of other secondary algorithms for solving the eigenvalues problem (§ 5.6.3). We have validated the algorithm on random systems (chapter 5) and applied it on two practical examples (chapter 6). Under the assumption of simple set of constraints and using only the states as optimization variable, we showed as the algorithm has better performance in terms of computational time compared with the *quadprog* solver of Matlab (§ 6.2.1).

7.1 Other possible research directions

The algorithm does not show convergence when the condition Hessian's condition number is large, future works could exploit the structure of the problem and try to find a matrix configuration that reduce the condition number. Nevertheless, this structure allows us to use tailored algorithm in a larger extent respect that we did in this work. For example it would be possible to implement a block-wise matrix-vector multiplication and a block-tridiagonal Cholesky factorization that can further improve the algorithm performance. In terms of storage memory, there is no need to store the entire Hessian matrix as we showed in § 5.2, indeed storing only the single matrices could save space. Another possible research direction could be adapting the FG to nonlinear systems.

Note

The results of the work presented in this thesis have been reported (of course briefly) in an article called *"Simple and efficient moving horizon estimation using the fast gradient*

methods". The authors are Bruno Morabito, Markus Kögel, Eric Bullinger, Gabriele Pannocchia and Rolf Findeisen. At the moment in which this thesis was completed the paper had been submitted for the NMPC'15 conference. Hopefully it will be available to the public starting from November 2015.

Appendix A

Linear algebra

A.1 Closed set and closed function

A closed set is a set which contains its limit points. Some example of closed set are:

- intervals $[a, b]$ with finite $a, b \in \mathbb{R}$;
- intervals $[a, \infty), (-\infty, b]$ with with finite $a, b \in \mathbb{R}$;
- the sphere or circle $S = \{x \in \mathcal{X} | d(x, \mathcal{O}) \leq R\}$ where $\mathcal{O} \in \mathbb{R}^n$ and $R \in \mathbb{R}$.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is closed if for each $\alpha \in \mathbb{R}$ in the sublevel set $\{x \in \mathrm{dom} f | f(x) \leq \alpha\}$ is a closed set.

A.2 Pseudo-inverse

Let $A = U\Sigma V^T$ be the singular value decomposition of $A \in \mathbb{R}^{m \times n}$ with $\mathbf{rank}A = n$. We define the *pseudo-inverse* or *Moore-Penrose inverse* of A as:

$$A^\dagger = V\Sigma^{-1}U^T \quad \in \mathbb{R}^{n \times m}$$

or equivalently:

$$A^\dagger = \lim_{\epsilon \to 0}(A^T A + \epsilon I)^{-1}A^T = \lim_{\epsilon \to 0} A^T(A^T A + \epsilon I)^{-1}$$

where $\epsilon > 0$ to ensure that the inverses exist. If $\mathbf{rank}A = n$ then $A^\dagger = (A^T A)^{-1}A^T$, if $\mathbf{rank}A = m$ then $A^\dagger = A^T(AA^T)^{-1}$, if instead A is square and nonsingular then $A^\dagger = A^{-1}$.

The pseudo-inverse comes up in problems of least-square, minimum norm, quadratic minimization and Euclidean projection. Indeed, for example $A^\dagger b$ is in general a solution of the leas-square problem $min\|Ax - b\|_2^2$, and $A^\dagger A = UU^T$ gives the Euclidean projection on $\mathcal{R}(A)$ instead $AA^\dagger = VV^T$ gives the Euclidean projection on $\mathcal{R}(A^T)$.

A.3 Positive semi-definite and positive definite functions

A function $f \in \mathbb{R}^n \to \mathbb{R}$ is positive definite on a neighbourhood of the origin, D, if $f(0) = 0$ and $f(x) > 0$ for every nonzero $x \in \mathbb{D}$. Is semidefinite if $f(0) = 0$ and $f(x) \geq 0$ for every nonzero $x \in \mathbb{D}$.

Appendix B

Systems theory

B.1 Lyapunov function

We consider a nonlinear, continuous, time-invariant system

$$\dot{x} = f(x)$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$. Without loss of generality, we say that the null vector 0 is an equilibrium point of f, that means $f(0) = 0$ (indeed we can always change the coordinate in the form of $x = \tilde{x} - x_e$, where x_e is the equilibrium point of the old coordinate system, and obtain that 0 is a equilibrium point).

Definition B.1 (Generalized energy function)**.** Now let us consider a *positive definite* function $V : \mathbb{R}^n \to \mathbb{R}$ and calculate its time derivative when it takes as input the state x:

$$\dot{V}(x) = \frac{d}{dt}V(x(t)) = \frac{\delta V}{\delta x}\frac{dx}{dt} = \nabla V^T f(x)$$

we now are going to demonstrate that if *certain assumptions* on V and $\dot{V}$ are satisfied, the trajectory $x(t)$ has some stable proprieties.

We can immediately notice that since V is positive definite, it will always return a *strictly positive scalar* when applied to the system state x, *unless* x is the stationary point $x = 0$. Intuitively we can think of this function as a *generalized potential energy function*, that describes the state's *displeasure* in staying far from its stationary point.

Theorem B.2 (Lyapunov boundedness theorem)**.** *If the function of Def.B.1 exists and* $\dot{V}(x) \leq 0, \forall x$ *then all trajectories are bounded. Namely for each* $x(t)$ *there is a* R *such that* $\|x(t)\| \leq R$ *for all* $t \geq 0$.

Proof. We note that for any trajectory x

$$V(x(t)) = V(x(0)) + \int_0^t \dot{V}(x(\tau))d\tau \leq V(x(0)) \tag{B.1}$$

this means that the whole trajectory lies in the set $\{x|V(x) \leq V(x(0))\}$ which is bounded since V is positive definite. □

An intuitive explanation could be the following: we said that $V(x)$ increases when x increases, therefore $\nabla V(x)$ points in the direction where x increases more *rapidly*. If $\dot{V}(x) = \nabla V(x)^T f(x) \leq 0$ means that the vector $\nabla V(x)$ forms an angle greater than $\pi/2$ with the vector $f(x)$, namely the system dynamic $f(x)$ is *pushing* in the opposite direction of $\nabla V(x)$, that is where the state is decreasing.

Theorem B.3 (Lyapunov globally asymptotically stability theorem)**.** *If the function of Def.B.1 exists,* $\dot{V}(x) < 0, \forall x \neq 0$ *and* $\dot{V}(0) = 0$*, then every trajectory* $x(t)$ *converges to zero as* $t \to \infty$*. We call this system* globally asymptotically stable*.*

In the *potential energy* point of view, we can say that $\dot{V}(x)$ is the *dissipation* of potential energy that is bringing the $V(x)$ (the potential energy function) to its minimum at 0.

Appendix C

Matlab codes

C.1 Fast Gradient method

```
function [ x,it,L,mu,flag] = fastgradient_n(H,f,xl,xr,varargin)

%   This algorithm implements the Nesterov's fast gradient method
%
%
%   I. The eigenvalue are calcualted with inverse iteration + Cholesky
%   factorization
%
%  [ x,it,grad_time ] = fastgradient(H, f, xl, xr,mu ,L, x0, itmax, tollx)
%
%   INPUT:
%   H       ---> Matrix of the opt. problem  1/2 x'H x + v'f
%   f       ---> Vector of the opt. problem
%   xl      ---> Lower constraints vector
%   xr      ---> Upper constraints vector
%
%   OPTIONAL INPUT:
%   x0      ---> Starting point
%   itmax   ---> Maximum number of iteration (by default 1000);
%   tollx   ---> Tollerance over the vector of estimate ( by defauld 1e-6)
%
%   OUTPUT
%   x           ---> Results
%   it          ---> number of FG iteration
%   grad_time   ---> Time spent in gradient calculation
%   eig_time    ---> Time spend in eigenvalues calculation
%   L           ---> Largest eigenvalue
%   mu          ---> Smallest eigenvalue
```

```
29
30 numvarargs = length(varargin);
31
32 if numvarargs > 3
33     error('myfuns:somefun2Alt:TooManyInputs', ...
34         'requires at most 2 optional inputs');
35 end
36 h         = size(H,1)         ;
37 % Set the default variables
38 optargs = {zeros(h,1) 10000 1e-8};
39
40 % now put these defaults into the valuesToUse cell array,
41 % and overwrite the ones specified in varargin.
42 optargs(1:numvarargs) = varargin;
43
44 % Give names to the variables
45 [x0,itmax, tollf] = optargs{:}  ;
46
47 %-------------------------------------------------------------------------
48 %                   EIGENVALUE EV. INVERSE ITERATION + CHOLESKY
49 %-------------------------------------------------------------------------
50
51 [ mu,L] = invit_sparse(H);
52
53 %-------------------------------------------------------------------------
54 %                            END EIGENVALUE SECTION
55 %-------------------------------------------------------------------------
56 errf      = tollf+1  ;
57 w         = x0       ;
58 x_old     = x0       ;
59 flag      = -1       ;
60 it        = 0        ;
61
62 grad = H*w +f;
63
64 while errf > tollf
65     x = w - 1/L * grad;
66     x       = max(min(x,xr),xl);
67     errf    = 0.5*(1/mu - 1/L) * norm(L* (w - x))^2;
68     w       = x + ( sqrt(L) - sqrt(mu) ) / ( ( sqrt(L) + sqrt(mu) ) ) *...
69       (x - x_old) ;
70     x_old   = x      ;
71      grad    = H*w +f ;
72     it      = it + 1 ;
73     if it > itmax
74         % Define a variable flag to know why the algorithm has stopped
75         flag = 0;
76         break
```

```
77      end
78      flag = 1;
79  end
80
81      if flag == 0
82          warning(['The algorithm has reached the maximum'...
83          ' number of iteration ',num2str(itmax)...
84          ' without matching the tollerance'])
85       elseif flag ==1
86           disp('Found a minimum that matches the required tollx'...
87            num2str(tollx))
88      end
89  end
```

C.2 Inverse Iteration with Cholesky factorization

```
1  function [ mu,L,it_L,it_mu] = invit_sparse(H)
2  % Inverse iteration method with cholesky factorization
3  %           [ mu,L,it_L,it_mu ] = invit(H)
4  % This method founds the mimimum and maximum eigenvalue of the matrix H
5  % using Cholesky factorization to compute the inverse H^-1 needed for the
6  % minimum eigenvalue.
7  %
8  % OUTPUT:
9  %           mu      minimum eigenvalue;
10 %           L       maximum eigenvalue;
11 %           it_L    iteration for L;
12 %           it_mu   iteration for mu;
13 %
14 % INPUT:
15 %           H       Matrix
16 h = size(H,1);
17 eigv = rand(h,1);              % Inizialization of random vector
18 v = eigv;
19 it_L = 1;
20 err_L = 100;
21 err_mu = 100;
22 tol_L = 1e-2;
23 c = norm(H,1);
24 cH = chol(-H+c*speye(h));
25 while err_L > tol_L
26   q = v/norm(v); % step 1
27   v = -cH\(cH'\q); % step 2
28   L = q'*H*q; % Rayleigh Quotient
```

```
29    err_L = norm(H*v - L * v,inf);
30    it_L = it_L+1;
31  end
32
33  v = eigv;% inverse iteration method for mu
34  it_mu = 1;
35  cH = chol(H);
36  while err_mu > tol_L
37    q = v/norm(v);%step 1
38    v = cH\(cH'\q); %step 2
39    mu = 1/((v'*q)); % Rayleigh Quotient
40    err_mu  = norm(H*v - mu*v,inf);
41    it_mu = it_mu+1;
42  end
43
44  end
```

Bibliography

[1] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[2] Thomas Kailath, Ali H Sayed, and Babak Hassibi. *Linear estimation*, volume 1. Prentice Hall Upper Saddle River, NJ, 2000.

[3] Arthur Earl Bryson. *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[4] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *AeroSense'97*, pages 182–193. International Society for Optics and Photonics, 1997.

[5] C.V. Rao, J.B. Rawlings, and D.Q. Mayne. Constrained state estimation for nonlinear discrete-time systems: stability and moving horizon approximations. *Automatic Control, IEEE Transactions on*, 48(2):246–258, Feb 2003. ISSN 0018-9286. doi: 10.1109/TAC.2002.808470.

[6] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In *Nonlinear Model Predictive Control*, pages 391–417. Springer, 2009.

[7] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[8] Sidharth Abrol and Thomas F Edgar. A fast and versatile technique for constrained state estimation. *Journal of Process Control*, 21(3):343–350, 2011.

[9] SB Pope. Computationally efficient implementation of combustion chemistry using in situ adaptive tabulation. 1997.

[10] Mark L. Darby and Michael Nikolaou. A parametric programming approach to moving-horizon state estimation. *Automatica*, 43(5):885 – 891, 2007. ISSN 0005-1098. doi: http://dx.doi.org/10.1016/j.automatica.2006.11.021. URL http://www.sciencedirect.com/science/article/pii/S0005109807000283.

[11] Niels Haverbeke, Moritz Diehl, and Bart De Moor. A structure exploiting interior-point method for moving horizon estimation. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 1273–1278. IEEE, 2009.

[12] Markus Kögel and Rolf Findeisen. A fast gradient method for embedded linear predictive control. In *Proceedings of the 18th IFAC world congress*, pages 1362–1367, 2011.

[13] Pablo Zometa, M Kögel, Timm Faulwasser, and Rolf Findeisen. Implementation aspects of model predictive control for embedded systems. In *American Control Conference (ACC), 2012*, pages 1205–1210. IEEE, 2012.

[14] T. Faulwasser, D. Lens, and C.M. Kellett. Predictive control for longitudinal beam dynamics in heavy ion synchrotrons. In *Control Applications (CCA), 2014 IEEE Conference on*, pages 1988–1995, Oct 2014. doi: 10.1109/CCA.2014.6981595.

[15] S. Richter and M. Morari. Stopping Criteria for First-Order Methods. Technical report, May 2012. URL http://control.ee.ethz.ch/index.cgi?page=publications;action=details;id=4065.

[16] J.L. Jerez, P.J. Goulart, S. Richter, G.A. Constantinides, E.C. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at megahertz rates. *Automatic Control, IEEE Transactions on*, 59(12):3238–3251, Dec 2014. ISSN 0018-9286. doi: 10.1109/TAC.2014.2351991.

[17] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997. ISBN 0898713617.

[18] J.B. Rawlings and D.Q. Mayne. *Model Predictive Control: Theory and Design*. ISBN 0975937707, 9780975937709.

[19] J.M.Maciejowski. *Predictive control with constraints*. Prentice Hall, 2001.

[20] Dimitri P Bertsekas. Nonlinear programming. 1999.

[21] Richard Bellman. The theory of dynamic programming. Technical report, DTIC Document, 1954.

[22] David G Luenberger. Observers for multivariable systems. *Automatic Control, IEEE Transactions on*, 11(2):190–197, 1966.

[23] Douglas G Robertson, Jay H Lee, and James B Rawlings. A moving horizon-based approach for least-squares estimation. *AIChE Journal*, 42(8):2209–2224, 1996.

[24] Svenya K. Fahlbusch. Bachelor's thesis: Comparison of different formulation of moving horizon estimation. Technical report, Otto von Guericke University of Magdeburg, 2014.

[25] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004. ISBN 0262042193.

[26] D. Karaboga and B. Basturk. On the performance of artificial bee colony (abc) algorithm. *Applied Soft Computing*, 8(1):687 – 697, 2008. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2007.05.007. URL http://www.sciencedirect.com/science/article/pii/S1568494607000531.

[27] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

[28] Yurii Nesterov. *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London, 2004. ISBN 1-4020-7553-7. URL http://opac.inria.fr/record=b1104789.

[29] Giulio M Mancuso and Eric C Kerrigan. Solving constrained lqr problems by eliminating the inputs from the qp. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC) Orlando, FL, USA, December 12-15, 2011*, pages 507–512, 2011.

[30] S. Richter, C.N. Jones, and M. Morari. Real-Time Input-Constrained MPC Using Fast Gradient Methods. In *Conference on Decision and Control (CDC)*, pages 7387 – 7393, Shanghai, China, December 2009. URL http://control.ee.ethz.ch/index.cgi?page=publications;action=details;id=3384.

[31] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' Guide (Third Ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. ISBN 0-89871-447-8. URL http://www.netlib.org/lapack/.

[32] J Baglama, D Calvetti, and L Reichel. Irbl: An implicitly restarted block-lanczos method for large-scale hermitian eigenproblems. *SIAM Journal on Scientific Computing*, 24(5):1650–1677, 2003.

[33] Finn Haugen, Rune Bakke, and Bernt Lie. State estimation and model-based control of a pilot anaerobic digestion reactor. *Journal of Control Science and Engineering*, 2014:3, 2014.

[34] DT Hill, SA Cobb, and JP Bolte. Using volatile fatty acid relationships to predict anaerobic digester failure. *Trans. ASAE;(United States)*, 30(2), 1987.

[35] G. Lettinga, A. F. M. van Velsen, S. W. Hobma, W. de Zeeuw, and A. Klapwijk. Use of the upflow sludge blanket (usb) reactor concept for biological wastewater treatment, especially for anaerobic treatment. *Biotechnology and Bioengineering*, 22(4):699–734, 1980. ISSN 1097-0290. doi: 10.1002/bit.260220402. URL http://dx.doi.org/10.1002/bit.260220402.

[36] Finn Haugen, Rune Bakke, and Bernt Lie. Adapting dynamic mathematical models to a pilot anaerobic digestion reactor. 2013.

[37] Dale E Seborg, Duncan A Mellichamp, Thomas F Edgar, and Francis J Doyle III. *Process dynamics and control*. John Wiley & Sons, 2010.

[38] Michael Madsen, Jens Bo Holm-Nielsen, and Kim H Esbensen. Monitoring of anaerobic digestion processes: A review perspective. *Renewable and Sustainable Energy Reviews*, 15(6):3141–3155, 2011.

[39] Finn Haugen, Rune Bakke, and Bernt Lie. Temperature control of a pilot anaerobic digestion reactor. 2013.

[40] DT Hill, SA Cobb, and JP Bolte. Using volatile fatty acid relationships to predict anaerobic digester failure. *Trans. ASAE;(United States)*, 30(2), 1987.

[41] Aswin N Venkat, James B Rawlings, and Stephen J Wright. Stability and optimality of distributed, linear model predictive control. part i: State feedback. *Texas-Wisconsin Modeling and Control Consortium*, 2006.