SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

INSTITUTE OF INFORMATION ENGINEERING, AUTOMATION, AND MATHEMATICS

Complexity Reduction of Explicit Model Predictive Control

Diploma thesis

FCHPT-5414-50911

2012 **Bc. Juraj Holaza**

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

INSTITUTE OF INFORMATION ENGINEERING, AUTOMATION, AND MATHEMATICS

Complexity Reduction of Explicit Model Predictive Control

Diploma thesis

FCHPT-5414-50911

Study program: automation and information in chemical and food technology
Number and name of the field of study: automation 5.2.14
Educational department: Faculty of Chemical and Food Technology
Supervisor: Doc. Ing. Michal Kvasnica, PhD.
Consultant: Ing. Alexander Szűcs

2012 **Bc. Juraj Holaza**

Slovak University of Technology in Bratislava
Institute of Information Engineering, Automation and Mathematics

Faculty of Chemical and Food Technology
2011/2012

DIPLOMA THESIS TOPIC

Registration number:	FCHPT-5414-50911
Student's ID:	50911
Author of thesis:	Bc. Juraj Holaza (50911)
Study programme:	Automation and Informatization in Chemistry and Food Industry
Study field:	5.2.14 automation
Thesis supervisor:	doc. Ing. Michal Kvasnica, PhD.
Consultant:	Ing. Alexander Szűcs

Topic: **Complexity Reduction of Explicit Model Predictive Control**

Length of thesis: 50

Topic specifications:

In Explicit Model Predictive Control (MPC) the on-line implementation effort is considerably reduced by pre-computing the optimal feedback law, for each feasible initial condition, and storing it in form of a Piecewise Affine (PWA) function. The function, which is defined over a set of polytopic regions, however, is often very complex and prohibits cheap implementation. The thesis therefore proposes an approach to reduce complexity of PWA feedback laws. The procedure is based on recovering a simpler, sub-optimal PWA feedback law from a particular approximation of the value function.

Main goals of the thesis are as follows:
1) provide an overview of Model Predictive Control and its implementation in closed loop;
2) derive optimal PWA approximation of a set of points;
3) propose an automated approach to constructing PWA functions which are bounded from below and from above;
4) recover a PWA feedback law from approximation of the value function;
5) illustrate the procedure by means of simple examples.

Diploma thesis topic submission date:	**13. 02. 2012**
Deadline for submission of Diploma thesis:	**19. 05. 2012**

Bc. Juraj Holaza
Student

prof. Ing. Miroslav Fikar, DrSc.
Head of office

prof. Ing. Miroslav Fikar, DrSc.
Study programme supervisor

ABSTRAKT

Vzhľadom na to, že prediktívne riadenie má relatívne veľké výpočtové nároky, jeho implementácia bola obmedzená iba na riadenie pomalých procesov. Z tohoto dôvodu cie tejto diplomovej práce je overiť možnosti explicitného prediktívneho riadenia a nájsť takú metódu, ktorá by bola schopná rozšíriť jeho využitie. V prvých kapitolách tejto práce sa lepšie oboznámime s danou problematikou. Dočítame sa aj o tom, ako by sme dokázali implementovať dané riadenie aj do procesov s rýchlou dynamikou. Práve tento poznatok nás bude viesť k našej metóde, ktorá sa bude snažiť znížiť dátovu záťaž linearizovaného systému a to aj za cenu zníženia výkonu riadenia. V každej nasledujúcej kapitole budeme riešiť všetky potrebné prvky, ktoré nás dovedú až k finálnemu tvaru našej metódy. O tom, že daná metóda je plne funkčná a, že bolo dosiahnuté aj požadované zrýchlenie, sa môžeme presvedčiť v poslednej kapitole.

Kľúčové slová: MATLAB, prediktívne riadenie (MPC), po častiach afinná funkcia (PWA)

ABSTRACT

The main goal of this diploma thesis is to verify the possibilities of model predictive control approach for a purpose of finding a method which will be able to extend its applicability, since this control strategy has relatively large computational demands. In the sequel we will be familiar with the whole problem more closely, therefore a main objective of this method will be specified. This objective will be based on decreasing capacity of explicit model predictive controls data of this method, which will achieve faster computation time of the optimal input to the system at the expense of suboptimality. In sequel Chapters we will gradually deal with all the important elements that will lead us to desired method. To prove that this method is fully functional and that desired objective is achieved, is reported in the last Chapter.

Keywords: MATLAB, model predictive control (MPC), Piecewise affine function (PWA)

I would like to thank my supervisor Doc. Ing. Michal Kvasnica, PhD. for his guidance, valuable discussions and comments. I would also like to thank to Ing. Alexander Szűcs for his patience and constructive critique. The largest thanks belongs to my family who supported me during my study.

Juraj Holaza

Content

Introduction **13**

1 Introduction to model predictive control (MPC) **14**

1.1 Comparison of PID, LQR and MPC control approaches 15

1.2 Advantages of model predictive control 17

1.3 Mathematical formulation of MPC 17

1.3.1 Convex optimization problems 17

1.3.2 Norms 18

1.3.3 Constraints 21

1.3.4 Mathematical formulation of the problem 23

1.4 Basic feature of model predictive control 24

1.4.1 Comparison of two standard forms of model predictive control 24

1.4.2 Creation of data for explicit model predictive control 25

1.4.3 Processing data of explicit model predictive control 27

1.5 MATLAB 29

2 Applicability of model model predictive control **30**

2.1 Sampling time 31

2.2 Computation time of explicit predictive control 32

2.3 Reducing the number of regions 33

3 Reconstruction of objective function **35**

3.1 Direct construction of piecewise affine objective function 35

3.1.1 Formulation in MATLAB 39

3.2 Other methods of construction piecewise affine function 41

4 Boundaries **42**

4.1 Lower boundary J 42

4.2 Upper boundary ***J*** 43

4.2.1 Definitions and theorems 43

4.2.2 MPC formulation 44

4.2.3 Creating the upper boundary 44

4.3 Admissible stable area 46

5 Fitting of the new objective function 47

5.1 Objective function and the upper boundary 47

5.2 Objective function and the lower boundary 48

5.3 Creating a new objective function 49

5.4 Implementation in MATLAB 50

6 Certification 51

6.1 Certification of the upper boundary 51

6.1.1 Formulation in MATLAB 51

6.2 Certification of the lower boundary 52

6.2.1 Formulation in MATLAB 54

6.3 Formulation in MATLAB 54

7 Control law 57

7.1 Approximate simplical control law 57

7.2 Formulation in MATLAB 58

8 Examples 61

8.1 Example 1 (lti_1d_stable) 61

8.1.1 Formulation in MATLAB 63

8.2 Example 2 (lti_1d_unstable) 64

8.2.1 Formulation in MATLAB 66

8.3 Summarization of examples 67

8.3.1 Summarization of example 1 67

8.3.2 Summarization of example 2 71

Conclusion **72**

Resumé **73**

References **76**

Appendix A – Main program **77**

Appendix B – Systems **78**

lti_1d_stable 78

lti_1d_unstable 78

Appendix C – prepare_data **79**

J_add_norm 80

get_J_epigraph 80

Appendix D – reindex_pwa **82**

Appendix E – Jlow_Jup **83**

Appendix F – fit_PWA **84**

Appendix G – Certifications **85**

cert_Jup 85

cert_Jlow 85

Appendix H – draw_PWA **86**

get_pwa_max 86

Appendix I – get_u **87**

get_Vu 87

norminf 87

norm1 88

Appendix J – plot_PWA_u **91**

Appendix K – get_PWA ... 92

Appendix L – Points_into_PWA ... 93

Appendix M – Commands and functions used in sumarization ... 94

Comparison of time requirements ... 94

get_time_m1 ... 95

get_time_m2 ... 95

Comparition of control laws ... 95

Table of used shortcuts

LQ	–	Linear Quadratic
LQR	–	Linear-Quadratic Regulator
MEMS	–	Micro-Electro-Mechanical Systems
MPC	–	Model predictive control
MPT	–	Multi-Parametric Toolbox
PID	–	Proportional–Integral–Derivative
PWA	–	PieceWise Affine

Introduction

Model predictive control is an atractive control strategy, where the optimal input sequence is obtained via solving optimization problems. Actually, this fact represents the main drawback of the aforementioned so-called on-line method, because such an optimization procedure requires a powerful computer with operating system and last, but not least an appropriate solver. Hence, this methodology only on slow processes can be implemented, since the optimal input value has to be obtainable within one sampling time.

This work is organized as follows: The first one has a theoretical character and there we can find a brief introduction to model predictive control, but also the advantages of convex functions, the types of norms and restrictions and much more. The advantages of model predictive control are explained by a simple comparison of PID, LQR and MPC control where the controlled system is a car and the driver is the appropriate regulator. The procedure of obtaining the data necessary for process control and its subsequent processing, are properly described and also for better interpretation graphically shown. The second part addresses the issue of implementation of model predictive control on fast systems. Here are mentioned reasons why it is necessary that the computation time, required to obtain an optimal (or suboptimal) input to the system, must be smaller or equal than sampling time of the discrete system. From this point forward our goal will be to reduce the number of regions, even at the cost of reducing the control quality. Thorough the Chapters we will be more familiar with this problem and step by step we will achieve a method, which will be capable of solving this problem (at a price of suboptimal solution) and at the same time it will guarantees the stability of the controlled system. In order to proof the functionality of this method we will try to implement it into a several examples and the results graphically illustrate.

1 Introduction to model predictive control (MPC)

Model predictive control experienced in recent decades a significant change from the theoretical research to practical applications. Its development was strongly influenced by the requirements of the industry. Recently, model predictive control with a lot of real industrial applications, is one of the most modern control approaches implemented in industrial processes. The first model predictive control algorithms were used before more than twenty-five years ago in industry as an effective way of controlling multidimensional systems with constraints. [1]

In general we can say that MPC is a control approach, where optimized variables are obtained by optimization over a finite time horizon subject to constraints To realize such computation the model and the initial state has to be known. The result of such an optimization procedure is then the sequence of optimal inputs, of which only the first one is implemented to the system. This is the reason why the model predictive control significantly differs from conventional control methods, which use time-invariant control law. [1]

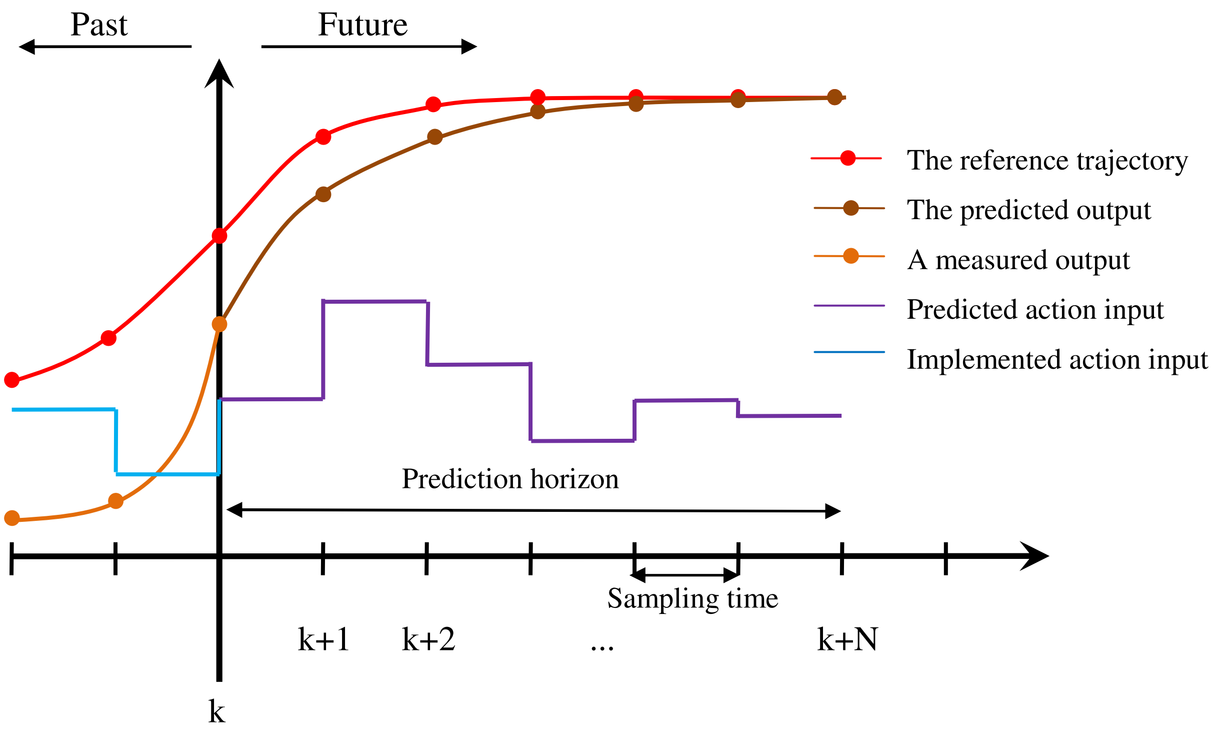


Figure 1.1: Graphic concept of receding horizon model predictive control

1.1 Comparison of PID, LQR and MPC control approaches

For a simplified comparison in order to compare the following control strategies, i.e. PID, LQR and MPC, consider the following situation:

The first car driver (PID) will know the initial state as well as the final target. His speed will have predetermined parameters of the car (controller). Driver of this car would not take into account traffic regulations since they will not be in his predetermined parameters. Because of this he will probably pay a lot of fines, therefore his driving will not be optimal. What is the worst is that he will be dangerous on the road. Since the driver will not look ahead, he will drive only by using rear-view mirrors and based on them he will adapt control. (PID controllers may adapt their control only by using feedback).

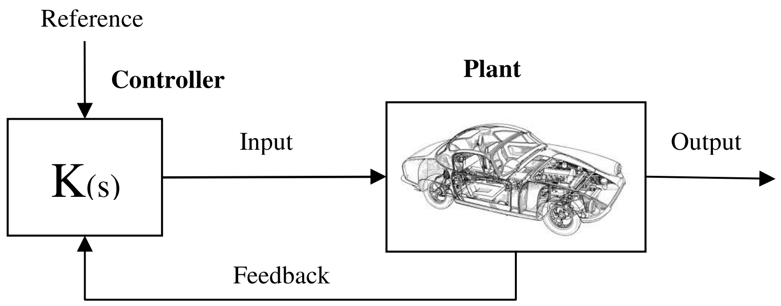


Figure 1.2: Scheme of control using PID controler

The driver (LQR) of the other car will start planning his own travel plan (optimal trajectory) before his journey will began. He should be thoroughly familiar with the possibilities in his trip in advance and thus he will know how to optimally control his car at any state (infinite prediction horizon). The problem arises when the driver is stubborn and does not intend to change his pre-made plan during the journey. Thus, he will drive only by his first planed trajectory and does not care of the possibility of complications (crashing into other cars, driver discomfort, blocked path, an unexpected obstacle on the way ...) what would most likely lead to a collision.

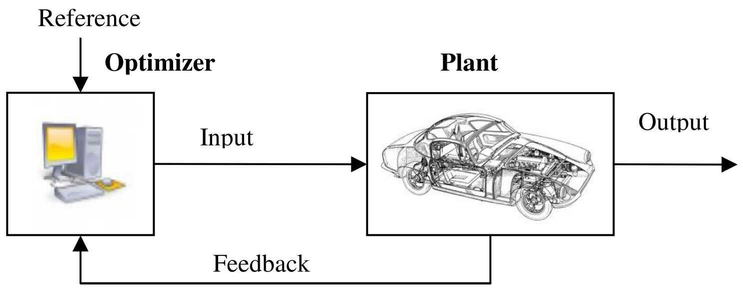


Figure 1.3: Scheme of control using LQR controler

The driver (MPC) of the third car will look around for a possibilities before starting the trip. Since he would be able to see path only a few meters ahead based on his eyesight (prediction horizon), his traveling plan will be restricted exactly by this distance (optimization over a finite time horizon). On the basis of what awaits him on the trajectory he will predict how quickly can the car go. He would thus has an optimal control plan for this trajectory. The plan will also consider all restrictions such as trajectory, speed limit, time, fuel consumption and others. Regardless of the extensive plan the driver will always use only the first input (e.g. for the first meter of the track), which will cause that the car will move forward (by one meter). Driver will then create a new plan based on present circumstances for a trajectory which can be recently seen. This way driver will create multiple trajectory plans where only the first planned input will be implemented to the system (car), which will lead him to a flexible driving performance. He will be able to respond fast enough to any possible danger on the road.

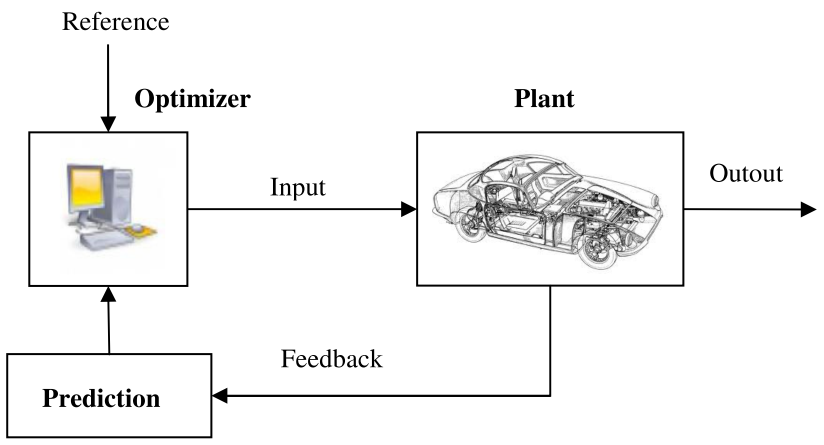


Figure 1.4: Scheme of control using MPC controler

Differences between each control approach can be characterized in the following table:

	MIMO systems	Performance	Constraints
PID	NO	NO	NO
LQR	YES	YES	NO
MPC	YES	YES	YES

Table 1.1: Differences between each control approach

1.2 Advantages of model predictive control

One of the biggest advantages of model predictive control is the effective handling of input constraints, where almost all plants are subject to such restrictions. MPC strategy thus overcomes the shortcomings of existing methods, such as the LQ (linear quadratic), respectively LQG (linear quadratic Gaussian), working on the infinite prediction horizon, which are not able to involve constraints in the optimization problem. In the practical problems controlled inputs are naturally limited in its scope (e.g. valve can be opened only to 100%), impact on system without using constrained inputs can be significant. Vital ingredients in control are represented by different safe conditions, which do not allow to exceed certain limitations of some physical variables (e.g. pressure, temperature, concentration). Moreover, MPC problem formulation allows one to include special type of constraints (e.g. soft constraints), which primarily serve for ensuring stability in case of systems with time delays or for respecting some physical limitations when non-minimum phase system has to be controlled. [1]

1.3 Mathematical formulation of MPC [2]

Optimization problem of the MPC is to minimize the objective function while we must take into account the different constraints (e.g., car control, respect the road, keeping distance from other cars, obey the maximum speed and so on). Mathematically we can formulate this problem as follows:

$$\begin{array}{lllll} min & f_0(x) & & (1.1a) & \text{Objective function} \\ \text{s.t.} & f_i(x) \leq 0 & & (1.1b) & \text{Inequality constraints} \\ & g_i(x) = 0 & & (1.1c) & \text{Equity constraints} \end{array}$$

1.3.1 Convex optimization problems

Convex optimization problem has several advantages:

- Achieving global solutions (if one attain local minimum, then it is also a global one)
- The availability of efficient solvers

Definition: The function $f(x)$ is called convex if for any two points x_1 and x_2 , $t\epsilon[0,1]$

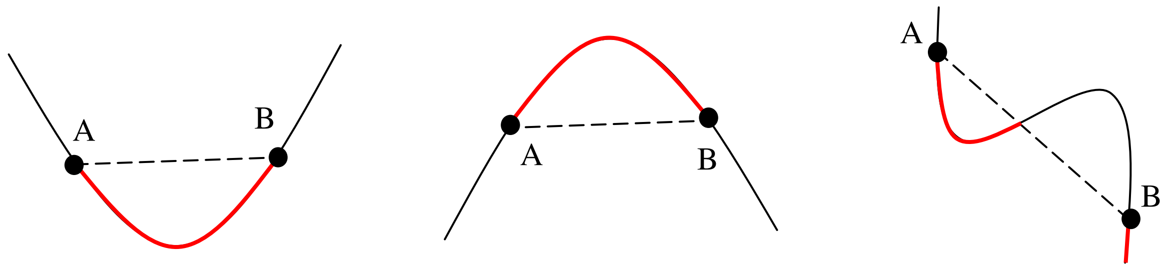
$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \tag{1.2}$$

and the function is called strictly convex if:

$$f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2) \tag{1.3}$$

For each $t\epsilon[0,1]$, $x_1 \neq x_2$.

If function $f(x)$ is (strictly) convex function, then function $-f(x)$ is (strictly) concave.



Convex function

Concave function

Nonconvex, nonconcave function

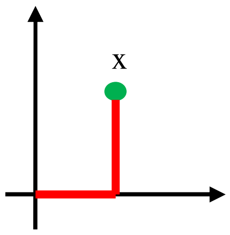
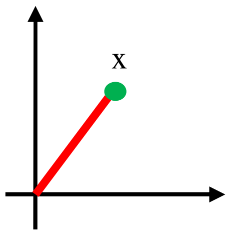
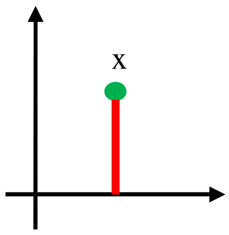
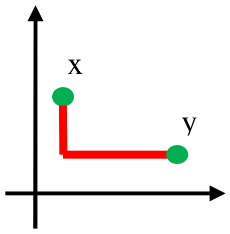
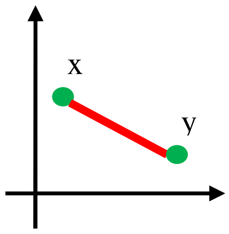
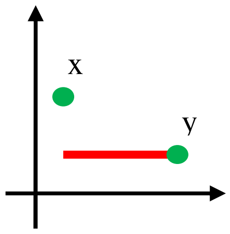
Figure 1.5: Convex and concave functions

If we pick up two points $A = f(x_1)$ and $B = f(x_2)$, one can obtain a line segment $\overline{AB}$. If the graph of the function on the assumed interval lies under the chord (tetivo), then function is convex. Naturally, if we replace the sign $\leq$ with the $\geq$ in the inequality (1.2) then fulfillment of the modified expression indicates concavity.

1.3.2 **Norms**

Norm is a function that assigns a length of all (nonzero) vectors in a vector space. They are convex functions, which we can write as P-norm l_p of the vector $x = [x_1, x_2, x_3, \ldots , x_n]^T$:

$$\|x\|_p = \left(\sum_i |x_i|^p\right)^{1/p} \tag{1.4}$$

Norm l_1 (Taxicab)	Norm l_2 (Euclidean)	Norm l_∞ (Infinity)
$\|x\|_1 = \sum_i \|x_i\|$ (1.4a)	$\|x\|_2 = \sqrt{\sum_i x_i^2}$ (1.4b)	$\|x\|_\infty = \max_i \|x_i\|$ (1.4c)
 $\|x\|_1$	 $\|x\|_2$	 $\|x\|_\infty$
 $\|x-y\|_1$	 $\|x-y\|_2$	 $\|x-y\|_\infty$

1.3.2.1 *Taxicab norm (l_1)*

Taxicab norm (14.a) belongs to piecewise linear cost function, which means that we will be using linear programing.

$$\begin{aligned} &\min \|x\|_1 \\ &s.t. \quad Ax \leq B \\ &\qquad\;\; Gx = H, \end{aligned} \tag{1.5}$$

where $x = [x_1, x_2, \dots, x_n]^T$. If minimization of 1 norm is assumed then the objective function can be transformed into the following form. As long as the first norm is defined as (1.4a), our objective function can be expressed by equation:

$$\min |x_1| + |x_2| + \ldots + |x_n| \tag{1.6}$$

In figure 1.6a is a a graphical representation for one state x_1. To compute taxicab norm in the certain point we have to construct epigraph ε_1 which will define area betwen the epigraph and the absolute value (figure 1.6*b*). Minimizing this surface until reaching point x_1 will get us the value of the Taxicab norm (figure 1.6*c*). Assume that x is a vector, ε is a vector of functions, which will substitute the maximum value. Both of them are our optimization variables.

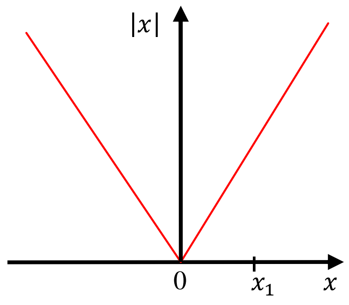


figure 1.6a

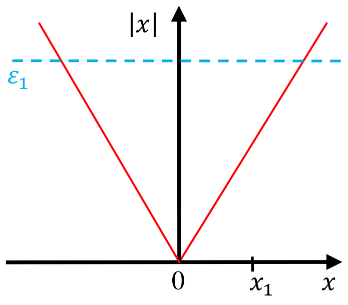


figure 1.6*b*

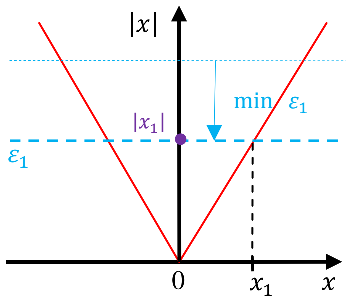


figure 1.6*c*

Based on this procedure problem (1.5) can be reformulate into:

$$\begin{aligned}
&\min \varepsilon_1 + \varepsilon_2 + . . . + \varepsilon_n \\
&s.t.\ \ Ax \leq B \\
&\qquad Gx = H \\
&\qquad -\varepsilon_1 \leq x_1 \leq \varepsilon_1 \\
&\qquad -\varepsilon_2 \leq x_2 \leq \varepsilon_2 \\
&\qquad \vdots \\
&\qquad -\varepsilon_n \leq x_n \leq \varepsilon_n
\end{aligned} \tag{1.7}$$

1.3.2.2 *Infinity norm*

Infinity norm (14.c) is similar to Taxicab norm. It also belongs to piecewise linear cost function, which means that we will be using linear programing. Difference between them is in objective function. In Taxicab norm we have considered ε as a vector of functions. In Infinity norm ε is a scalar that will represent only the worst absolute value among vector x.

$$\begin{aligned}
&\min \varepsilon \\
&s.t.\ \ Ax \leq B \\
&\qquad Gx = H \\
&\qquad -\varepsilon \leq x_1 \leq \varepsilon \\
&\qquad \vdots \\
&\qquad -\varepsilon \leq x_n \leq \varepsilon
\end{aligned} \tag{1.8}$$

1.3.2.3 *Euclidean norm*

Problem that uses the euclidean norm (14.b) will not belong to linear programing since cost is neither linear, nor piecewise linear. What is worst, it does not even belong to quadratic programing (because of the square root). The most common procedure to transform this norm into quadratic programing is to multiply it with itself. In another words we will get the squared value of this norm.

$$\|x\|_2 \Rightarrow \|x\|_2^2 \tag{1.9}$$

$$\begin{aligned} &\min \sqrt{x^T x} \\ &s.t. \;\; Ax \leq B \\ &\qquad Gx = H \end{aligned} \quad (1.10) \qquad \Rightarrow \qquad \begin{aligned} &\min x^T x \\ &s.t. \;\; Ax \leq B \\ &\qquad Gx = H \end{aligned} \quad (1.11)$$

This modified form of the norm distorts the true value so that in the interval $x\epsilon(-1;1)$ values are undervalued while in the interval $x\epsilon(-\infty;-1) \cup (1;\infty)$ values are overvalued.

1.3.3 **Constraints**

For control of real technological processes or technical systems is necessary to meet a number of limitations which are defined in advance. The role of constraints in the control design has at least three important aspects.

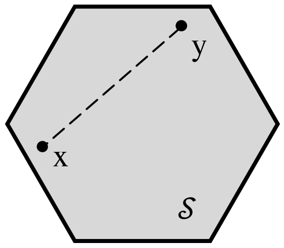
1. Using restrictions for better representation of physical systems (input saturation)
2. Using constraints to ensure stable control (constraints in the form of end-stabilizing constraints)
3. Using restrictions for tuning a controller parameters to achieve better quality control

In terms of character constraints can be divided into:

- Physical constraints (eg, we can not affine term the gear lever into a higher level than the design allows)
- Technological constraints (obeying certain speed limitations)

In practical applications, we often encounter with convex constraints, which are easily solvable. A set $\mathcal{S} \subseteq \mathbb{R}^n$ is convex if: $x, y \in \mathcal{S}, \;\; \lambda, \mu \geq 0, \;\; \lambda + \mu = 1$

$$\lambda x + \mu y \in \mathcal{S} \tag{1.12}$$



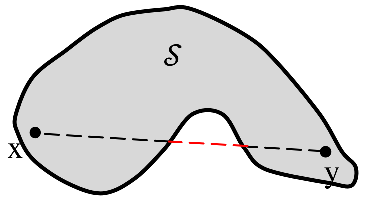
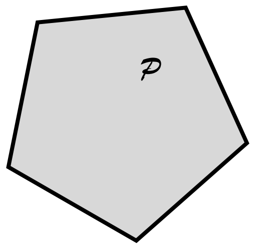
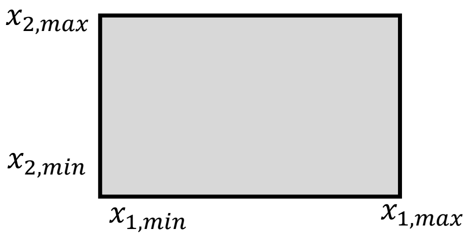
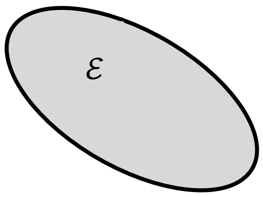
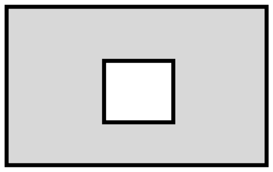


Figure 1.7: Convex and nonconvex sets

The most common examples of sets of constraints are:

• Polytopic $x \in \mathbb{R}^n$ $\quad \mathcal{P} = \{x \| Ax \leq B\}$	
• Box $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix},\ B = \begin{bmatrix} x_{1,max} \\ x_{2,max} \\ -x_{1,min} \\ -x_{2,min} \end{bmatrix}$	
• Ellipsoidal – more difficult to solve $x \in \mathbb{R}^n$ $\quad \mathcal{E} = \{x \| (x - x_0)P(x - x_0) \leq r\}$	
• Nonconvex – extremly difficult to solve	

Polytope $\mathcal{P}$ represents an area that was created as an intersection of all halfspaces (constraints in the form of inequality (1.1b)), while constraints in a form of equality (1.1c) define an area where must lie the optimum (in this case lines p, r, s, t). In figure 1.8 is ilustrated an example of all constraints, where permissible area of the problem is highlighted by a red color.

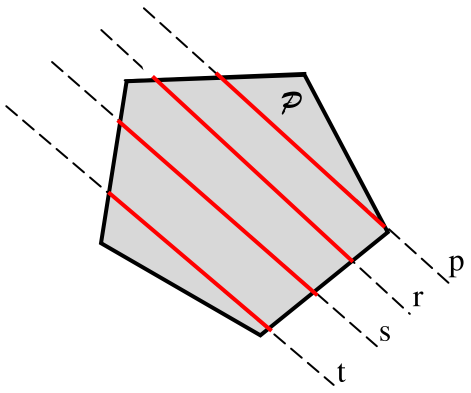


Figure 1.8: Constraints in form of equality and inequality

1.3.4 **Mathematical formulation of the problem**

Optimization problem can be defined as:

$$\min \sum_{k=0}^{N-1} \left(\|Q_x x_{t+k}\|_p + \|Q_u u_{t+k}\|_p \right) \qquad (1.13a) \qquad \text{Objective function}$$

$$s.t. \;\; x_{t+1} = Ax_t + Bu_t \qquad (1.13b) \qquad \text{Plant model}$$

$$x_t = x(t) \qquad (1.13c) \qquad \text{Initial conditions}$$

$$x_t \in \mathcal{X} \qquad (1.13d) \qquad \text{State constraints}$$

$$u_t \in \mathcal{U}, \qquad (1.13e) \qquad \text{Input constraints}$$

where

N	– Prediction horizon
P	– Norm
Q_x, Q_u	– Weighting matrices
$\mathcal{X}, \mathcal{U}$	– state and input constraints
x_{t+k}, u_{t+k}	– values of states and inputs in k-stage of prediction

1.4 Basic feature of model predictive control

Recently, the model predictive control method is being applied to the control systems with fast dynamics, hybrid systems, to the precise micro-electro-mechanical systems (MEMS). They are also applied in the development of new advanced control functions of mechatronic systems, especially in the automotive industry. These applications are allowed to create or develop new numerically efficient methods and model predictive control strategies for minimizing the practical control numerical computational load especially in real-time modes at the expense of a higher volume of auxiliary calculations carried out in real time mode. In this context, are rapidly developing methods aimed at explicit model predictive control using multi-parametric programming. [1]

1.4.1 Comparison of two standard forms of model predictive control

Classical approach based on on-line computation involves the following steps:

1. Obtain information about the current state $x(t)$ (either by direct measurement or through a Kalman filter)
2. Solve the optimization problem
3. Obtain optimal action input$u(t)$
4. Implement current value of the control value into plant
5. Repeat from step 1

Explicit model predictive controlled involves the following steps:

1. Pre-calculate the vectors of optimized problem
2. Obtain information about the current state $x(t)$ (either by direct measurement or through a Kalman filter)
3. Obtain optimal input $u(t)$
4. Implement input value into plant
5. Repeat this procedure from step 2

As the name implies, the on-line control is carried out continuously during the control of the system. This method is usually time consuming, hence the classical on-line approach was mainly implemented on slow procesess. Although the level of computer technology now forwarded much further, overloading of the computer can still be ineffective (e.g. in economic terms). In the "off-line" control we will firstly linearize the problem and so we will divide the objective function into n-PWA functions. Each region is described by a vector of objective function, which is assigned to an adequate vector for the action input. Subsequently, the system will be controlled by these pre-made vectors. This method can significantly relieve the processor at cost of increased requirements on the memory. The disadvantage of this method is mainly its flexibility relative to the changing structure of the system. But if we will be assuming a time invariant system, then this problem can be ignored.

1.4.2 **Creation of data for explicit model predictive control**

In this Section will be explained a simplified method of creation of data for explicit model predictive control. Let us consider a convex objective function$f\colon y = f(x)$,

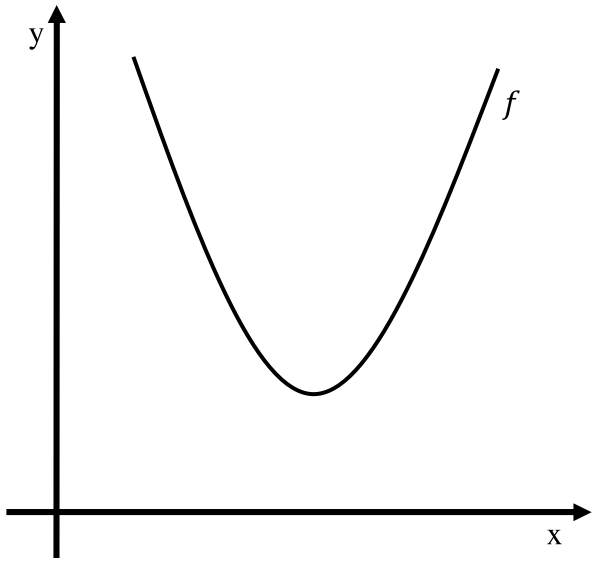


Figure 1.9: Objective function

which we will divide into n-PWA functions as follows:

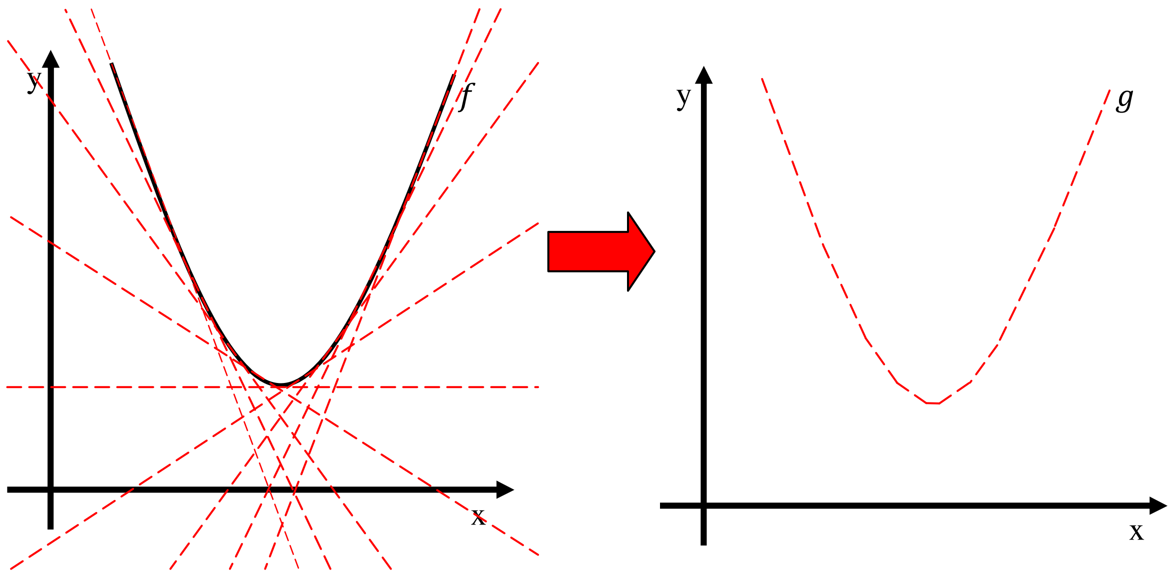


Figure 1.10: Linearization of the objective function

Number of sections affects the quality of the model in proportion, as it is a linearization. So that created piecewise affine objective function g describes our original objective function f with n-regions (sections). The function g can be defined as a linear function:

$$y = a_i^T x + b_i, \tag{1.14}$$

where $P_i \subseteq \mathbb{R}^n$, $x \in P_i$characterizes the i-th region, $a_i \in \mathbb{R}^{nx1}$, $b_i \in \mathbb{R}$, $i = \overline{1,k}$.

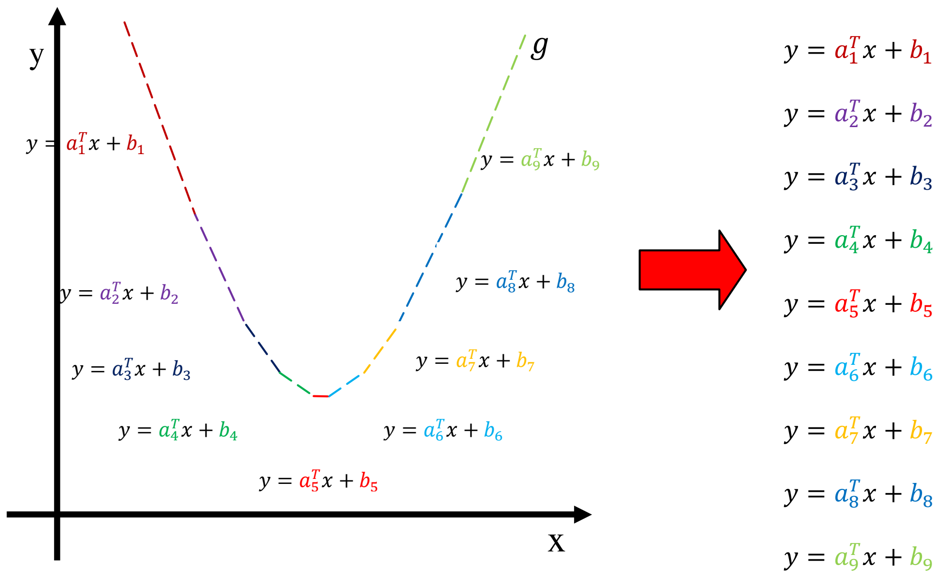


Figure 1.11: Piecewise affine objective function

Parameters thus obtained are stored in the matrix:

$$A = (a_1,\ a_2, \ldots, a_n) \tag{1.15}$$

$$B = (b_1,\ b_2, \ldots, b_n) \tag{1.16}$$

Each region has created its own associated control law, which also can be defined as a linear function:

$$u = c_i^T x + d_i, \tag{1.17}$$

where $P_i \subseteq \mathbb{R}^n$, $x \in P_i$ characterizes the i-th region, $c_i \in \mathbb{R}^{nx1}$, $d_i \in \mathbb{R}$, $i = \overline{1,k}$.

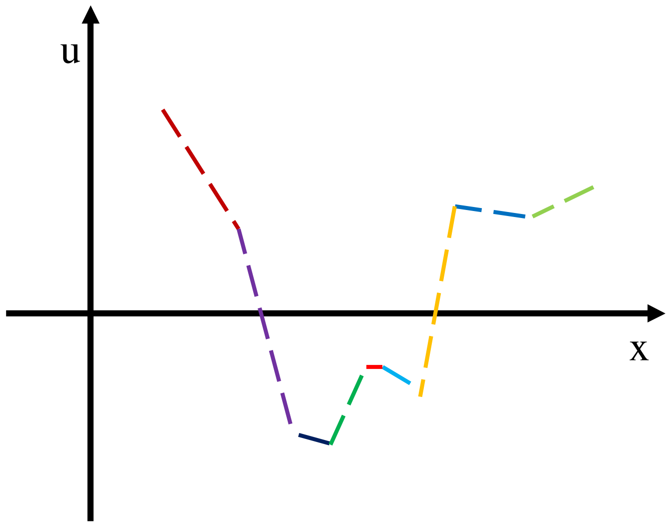


Figure 1.12: Control law

We will also save these parameters into matrices:

$$C = (c_1,\ c_2, \ldots, c_n) \tag{1.18}$$

$$D = (d_1,\ d_2, \ldots, d_n) \tag{1.19}$$

Model expressed by equations (1.14) and (1.17) is also known as piecewise affine (PWA) model and falls into the category of hybrid systems

1.4.3 Processing data of explicit model predictive control

Let us assume that we have all the necessary data of the hybrid system: A, B, C, D $(1.15, 1.16, 1.18, 1.19)$.

This system will be controlled by using the following steps:

1. With the initial state x_0 we will compute the functional values of each region$i$:

$$f_i(x_0) = a_i^T x + b_i \tag{1.20}$$

, where $a_i$and b_i are parameters of the given region i.

2. To find out in which region (k) we currently are, we need to find the maximum value among all functional values:

$$k = \max f_i(x_0) \tag{1.21}$$

3. If the current region is known, the first optimal value of the input will be computed from the corresponding equation of the affine function (for a given region):

$$u_1 = c_k^T x_0 + d_k \tag{1.22}$$

4. Implementation of the current value of the input to the system lead us to a change in state of the system, thus this procedure will be applied again from the first step.

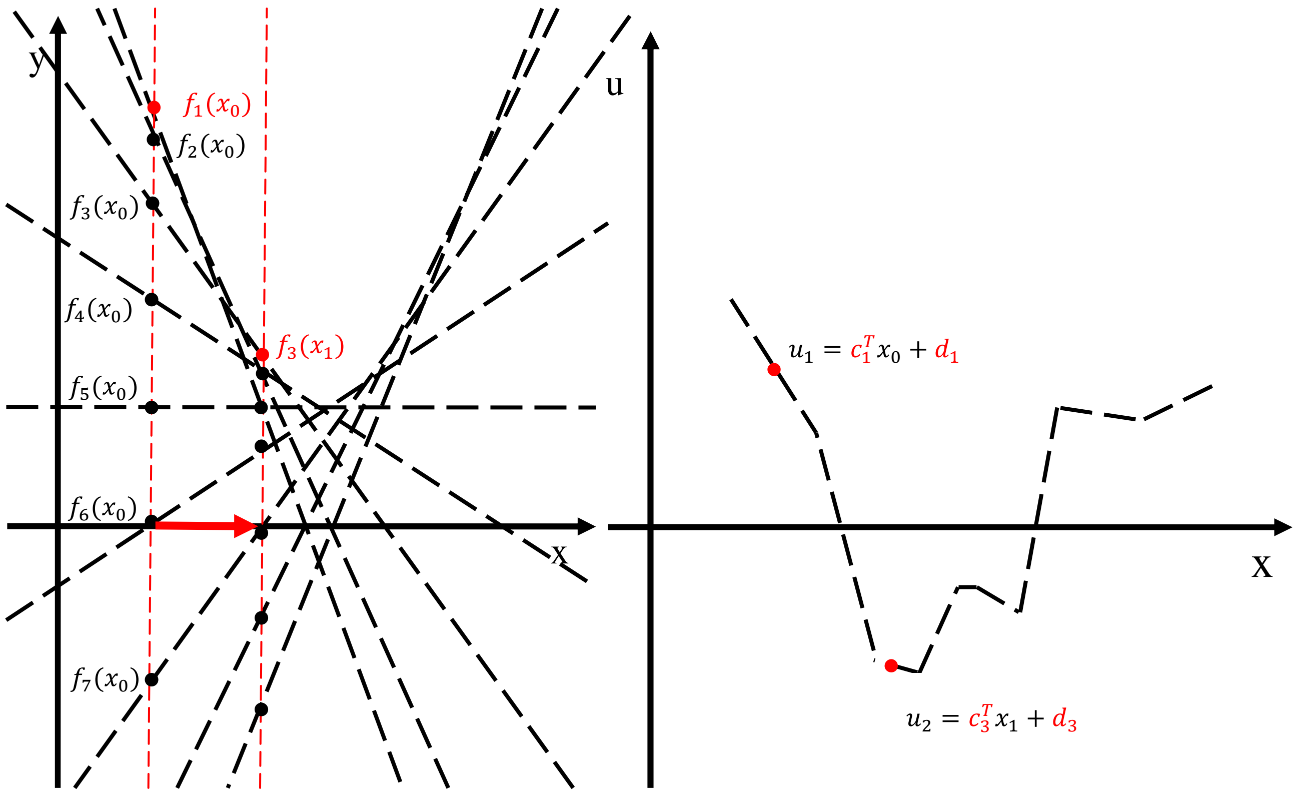


Figure 1.13: Computation of a optimal action inputs

1.5 MATLAB

MATLAB is a programming environment for scientific and technical calculations and modeling, design of algorithms, simulations, plotting of functions and data, creation of user interfaces, measurement and signal processing. MATLAB is an abbreviation of MATrix LABoratory, which corresponds to the fact that key data structures, using in calculation are matrices. It also allows to interface with programs written in other languages, including C/C + +, Java and Fortran.

For MATLAB was created numerous number of toolboxes that extend the capabilities of programming. One of them is called MPT (The Multi-Parametric Toolbox) for the design, analysis and implementation of optimal controllers for a limited (linear, nonlinear and hybrid) systems. The effectiveness of the code is guaranteed by an extensive library of algorithms from the field of computational geometry and many multiparametric optimization. [3]

2 Applicability of model model predictive control

In the first Chapter we have get familiar with model predictive control as one of the most modern control approach that is mostly implemented on industrial systems because of their large time constants. Model predictive control is rapidly expanding caused of its effective handling of input restrictions and the fact that it is predicting the future of the model. The reason why MPC is not implemented into quicker systems can by demonstrated on the following example. Let us say that our controlled system is a chemical reactor where we will control temperature of reactor by stream of cooling water. Chemical reactions can have realy fast thermal gradients that require quick response for cooling. We are talking about exothermic reactions since more intense reactions can lead us to produce great amount of heat. If the additional heat that is not required for propagation of reaction will not be cooled soon enough the system can reach certain point from where it will be unstable. The temperature will grow exponentially until explosion will eventually come. This explosion will not only cause economic and environmental losses, but may as well cause losses of lives or suing and disintegration of the entire company. So to avoid scenario from *example 1*, our controller has to ensure to compute the correct action inputs to the system within a specified period.

$$T_c \leq T_s, \tag{2.1}$$

where T_c is time required to compute optimal input and T_s is sampling time of the system.

Model predictive control is very computationally expensive because several predictions of control inputs from certain state are being performed (only the first one is used). This prediction is required for better control performance. Of course we can decrease it at the cost of losing one of the advantages of this control. From Chapter 1.4 we know that there exist two different methods of control. The first one is called on-line method and it computes whole mathematical formulation (1.13) several times for each sampling time. On the other hand the second explicit method does not compute the whole mathematical formulation, but only simple searching in the tables of piecewise affine functions. From this we can deduce that this method is more useful for us. From now on we will consider only this method.

2.1 Sampling time

Each discretized system has sampling time that represents period of time where system receives information about states, safety sensors and so on. Now we know that the controler must process all received informations and use them to determine the appropriate input to the system. Since sampling time is directly linked to the speed of the system, it can be read from it. Length of sampling time can be determined from following interval:

$$T_s \in \left(\frac{T_{90}}{5}, \frac{T_{90}}{15}\right), \tag{2.2}$$

where T_{90} is time when the step response of the output reached 90% of the steady-state value (minimal phases system) or 90% of the maximum amplitude (non-minimal phases system) what is sufficient since all characteristics of the system should be described (figure 2.1).

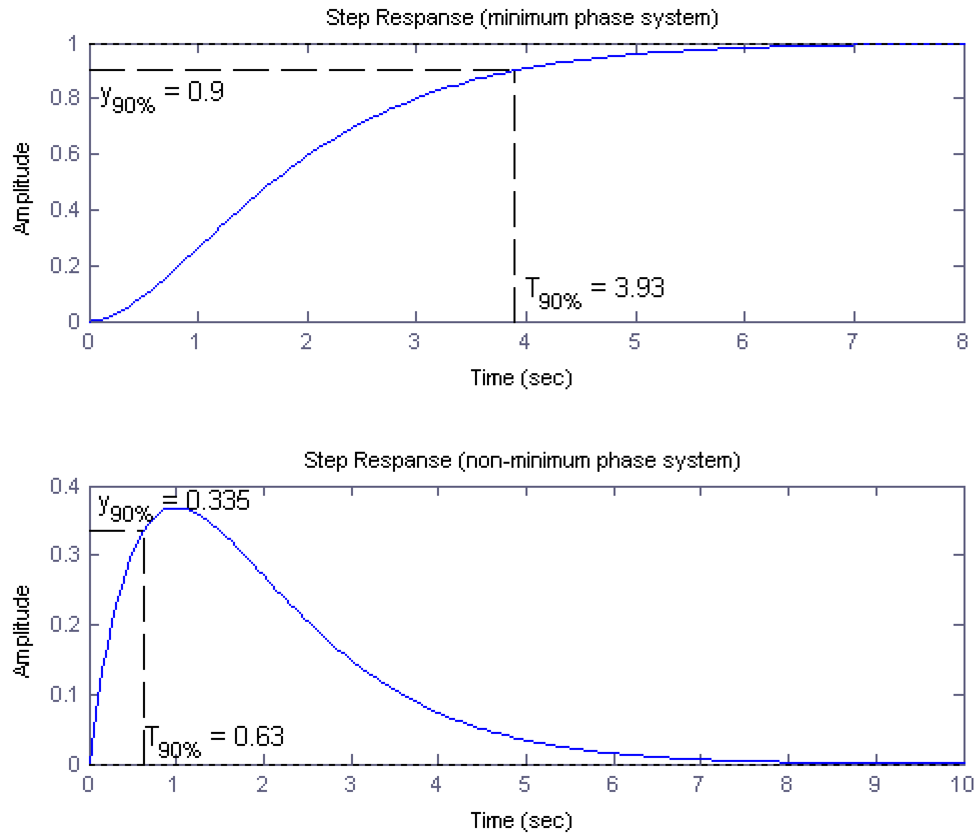


Figure 2.1: Step response of minimal and non-minimal phase system

2.2 Computation time of explicit predictive control

As it was mentioned computation time of explicit predictive control consists of searching values in the table. In this table we find matrices (1.15), (1.16), (1.18) and (1.19) via which we express equations (1.14) and (1.17). The whole method how we are computation data of predictive explicit control is described (graphicly as well) in Chapter 1.4.3. If information about the current state is obtained we compute all function values of this state for each region (1.20). By finding the maximal value among them we will associate the index of the current state (1.21). Optimal input is then simply compute from equation (1.22). From this procedure computation time can be expressed as a function of number of regions *k*:

$$T_c = f(k) \tag{2.3}$$

Problem can be seen from a diferent perspective if after linearization all regions (their intervals) would be saved with matrices (1.15), (1.16), (1.18) and (1.19). Finding the correct index of the region will be defined as assigning current state of the system to the correct interval region. We are simply asking whether the state belongs to the first, second, ..., n-th region (figure 2.2). Then, if the index is known, the optimal input will be directly calculated from the associated matrices (1.17). In this case computation time is expressed as linear function:

$$T_c = a + \sum_{i=1}^{r} t_c, \tag{2.4}$$

where a is time required to solve equation of optimal input (1.17), r represents index of the correct region and t_c is the time required to determine whether the state of the system lies in one region.

$$\begin{tikzcd}
\text{Start} \arrow[r] & R_1 \arrow[r, "x_0 \notin R_1"] \arrow[d, "x_0 \in R_1"'] & R_2 \arrow[r, "x_0 \notin R_2"] \arrow[d, "x_0 \in R_2"'] & \cdots \arrow[r, "x_0 \notin R_{k-1}"] & R_k \arrow[d, "x_0 \in R_k"'] \\
& u = c_1^T x_0 + d_1 & u = c_2^T x_0 + d_2 & & u = c_k^T x_0 + d_k
\end{tikzcd}$$

Figure 2.2: Seeking the region of validity

Since we do not know in what region we are, we have to count with the worst time. This time occurs if and only if the state will be located in the last region and therefore the algorithm will have to check every single region. Equation (2.4) will then have the form:

$$T_c = kt_c + a, \tag{2.5}$$

where k is the number of regions.

2.3 Reducing the number of regions

From equation (2.5) is obvious that computation time T_c is directly proportional to the number of regions k, therefore if we want to met expression (2.1) we need to reduce the number of regions. From Chapter 1.4.2 we know that initial data for explicit model predictive control are made of linearization of the system (figure 2.3). So from continuous function $f{:}y = f(x)$ we will get piecewise affine function g, for which applies (1.14). This function will be associated with a control law expressed by equation (1.17).

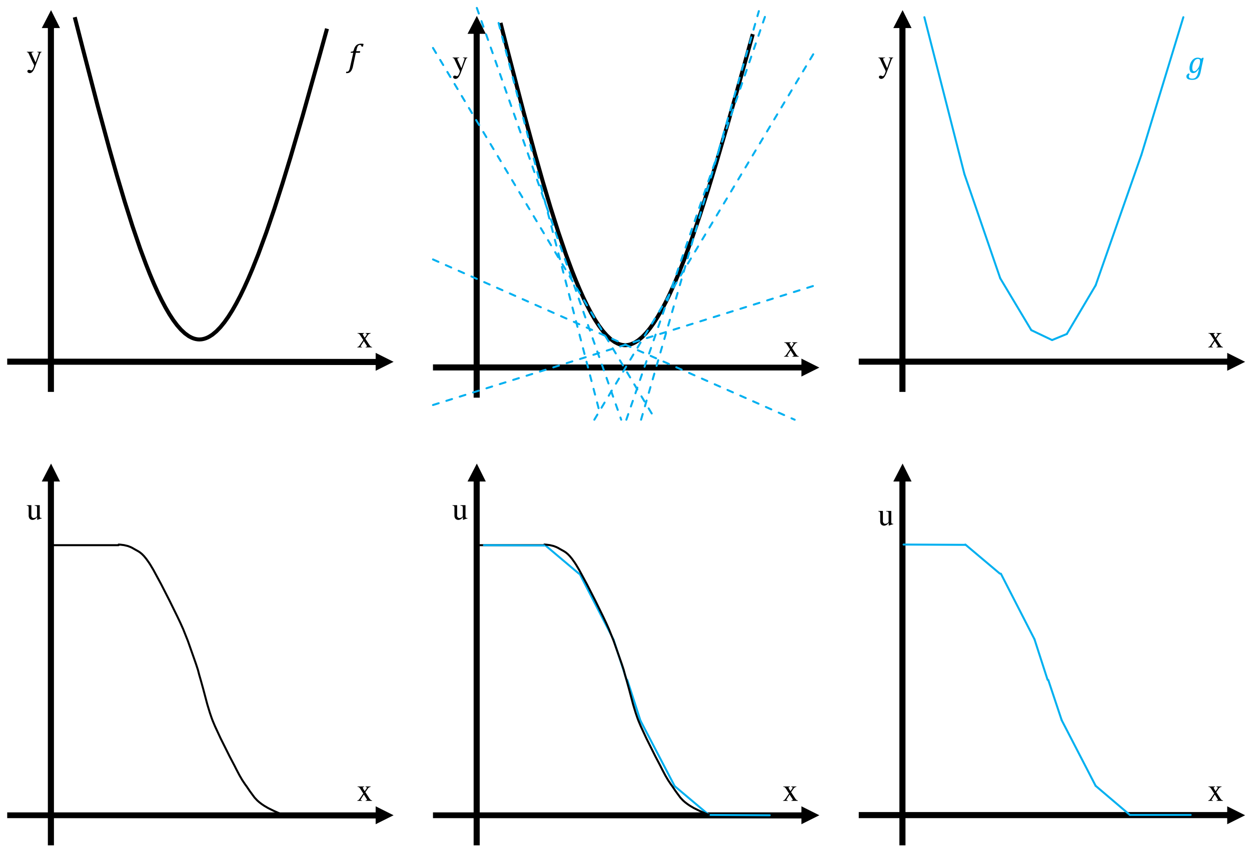


Figure 2.3: PWA functions

Now if we want to simplify whole formulation of piecewise affine function defined over ten regions at figure 2.3, we have to reduce the number of regions. Based on this we will decrease requirements for data storage and due to equation (2.5) computation time will be decreased as well. On the other hand innacuracies of linearization and of the performance (optimality) will be increased.

From the procedure, where PWA function described over ten regions g we will get another PWA function h described over two regions:

$$g\colon y^* = f^*(x) \;\longrightarrow\; h\colon \tilde{y} = \tilde{f}(x), \tag{2.6}$$

which can be seen on figure 2.4. If $g\colon y^* = f^*(x)$ is optimal PWA function, then $h\colon \tilde{y} = \tilde{f}(x)$ is suboptimal PWA function.

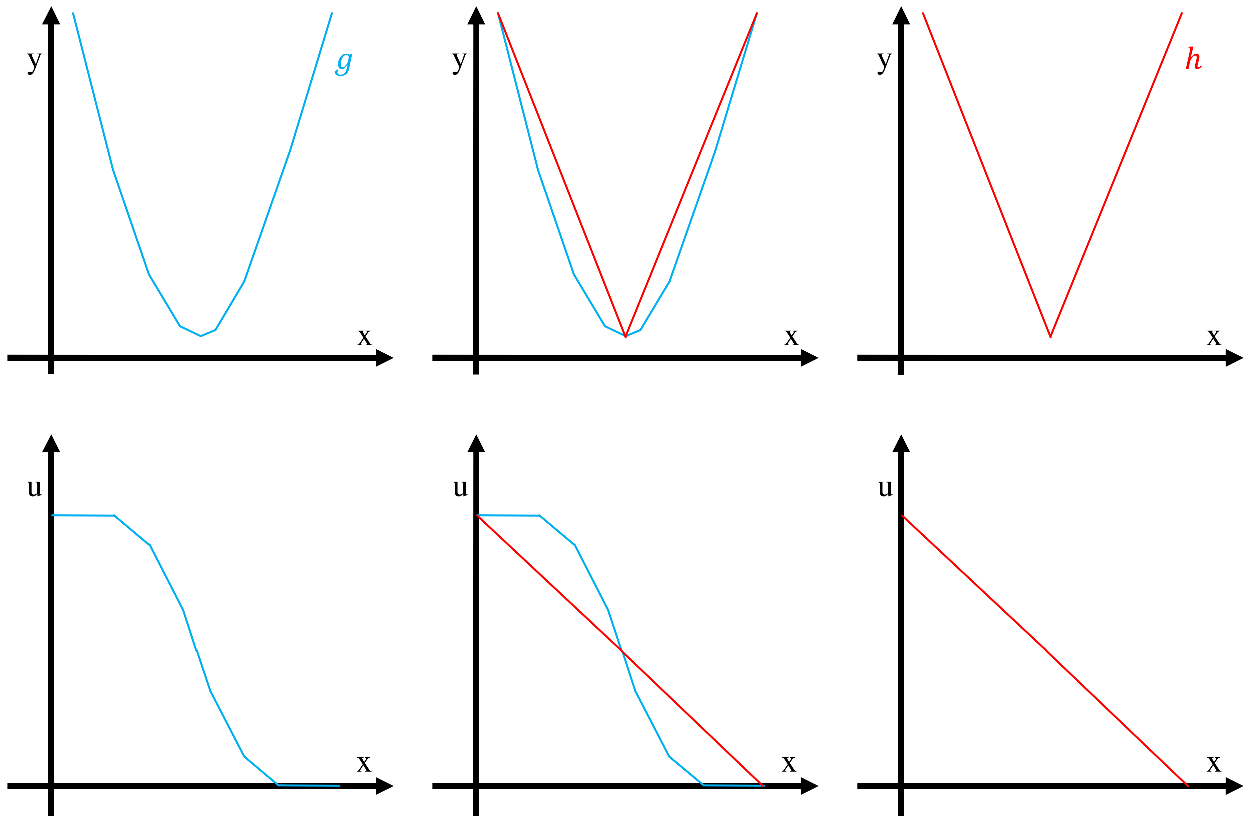


Figure 2.4: PWA function over 2 regions

3 Reconstruction of objective function

Till now we assumed to have objective function as a continuous function (figure 1.9), but in most cases it is not true. In real systems after implementing certain input (inputs), by using feedback (observer, kalmans filter ...) we will receive only information about current state (states). For this given state (states), we can determine its (their) function value. So basically by implementing several inputs (set of inputs) we will get n-coordinates of the objective function as we can see in figure 3.1. Now our goal would be to reconstruct objective function by means of these points.

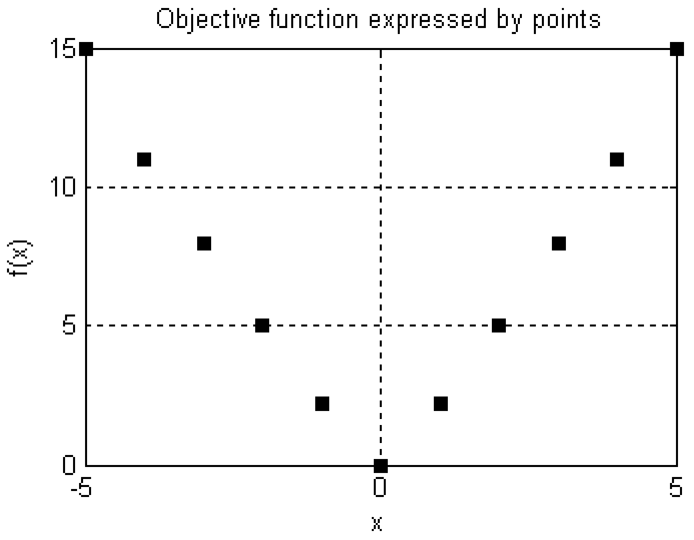


Figure 3.1: Objective function expressed by points

3.1 Direct construction of piecewise affine objective function

By using this direct method we will attempt to reconstruct a piecewise affine function directly from the given points. Basically what we want to do is to find out all slopes and affine terms of each line that will be describing our piecewise affine function. To achieve this we have to performe the following steps:

1. In the first step we will create lines between given point for each $x_i \leq x_j \; \forall i \neq j$.

$$f(x_i) = \alpha_{i,j}^T x_i + \beta_{i,j} \tag{3.1a}$$

$$f(x_j) = \alpha_{i,j}^T x_j + \beta_{i,j}, \tag{3.1b}$$

where x are coordinates of points, α is a slope, β is a affine term, i is index of the given point, j is index of each further point $j = \overline{\imath + 1, n}$.

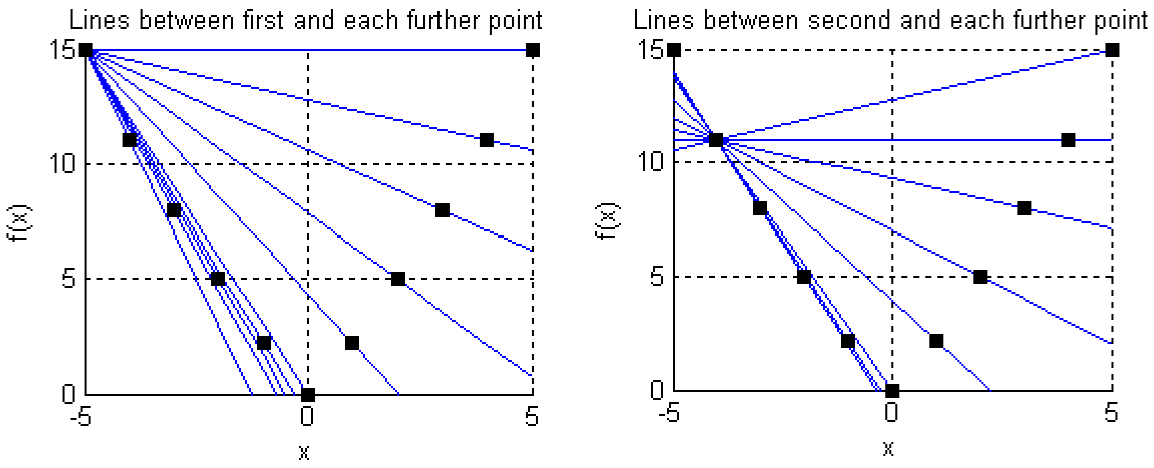


Figure 3.2: Lines leading through given point

In the figure 3.2 are two examples for first point ($i = 1, j = \overline{2,11}$) and the second point ($i = 2, j = \overline{3,11}$).

2. In the second step we will find out correct slopes and affine terms of each following pair of points (each region).

   From equations (3.1) we can express slope by:

$$\alpha_{i,j} = \frac{f(x_i) - f(x_j)}{x_i - x_j} \tag{3.2}$$

   Since we are assuming that objective function is convex, correct slope can be computed as a minimum value among all slopes:

$$\alpha_i = \min(\alpha_{i,j}) \tag{3.3}$$

   By combining equations (3.2) and (3.3), we will get the equation:

$$\alpha_i \leq \frac{f(x_i) - f(x_j)}{x_i - x_j} \tag{3.4}$$

   If slopes between two following points are known, then corresponding affine terms can be obtained from equation:

$$\beta_i = f(x_i) - \alpha_i x_i \tag{3.5}$$

In figure 3.3 are two examples of correct slope and affine term for the given point.

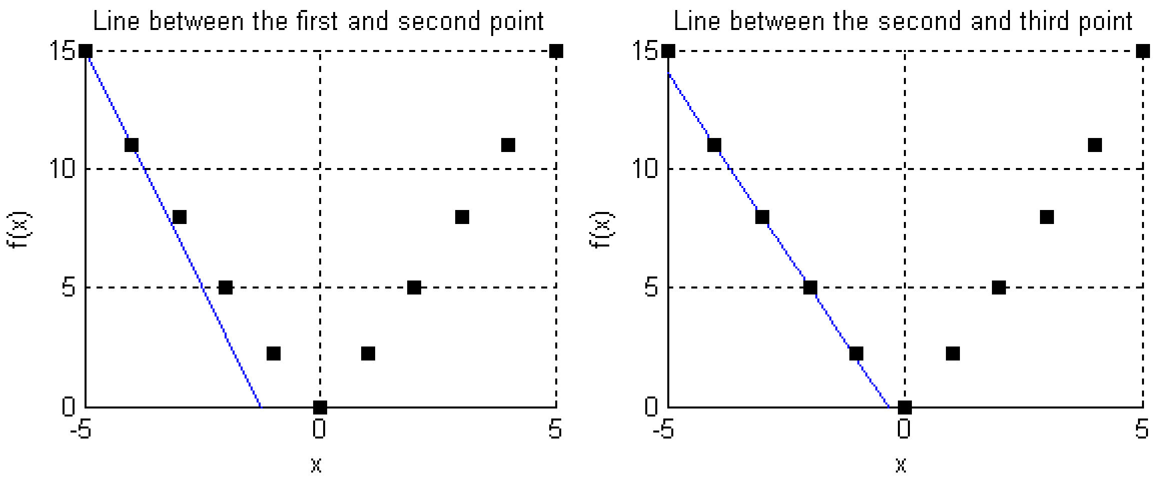


Figure 3.3: Line leading through given point

3. Since that objective function is expressed only as a set of lines (figure 3.4) and not as a line in the corresponding regions (figure 1.11), we will have to change it in this last (third) step. There exist several methods how to solve this problem. For example we can find out regions by intersections where in this case we would find coordinates of our starting points as edges of regions.

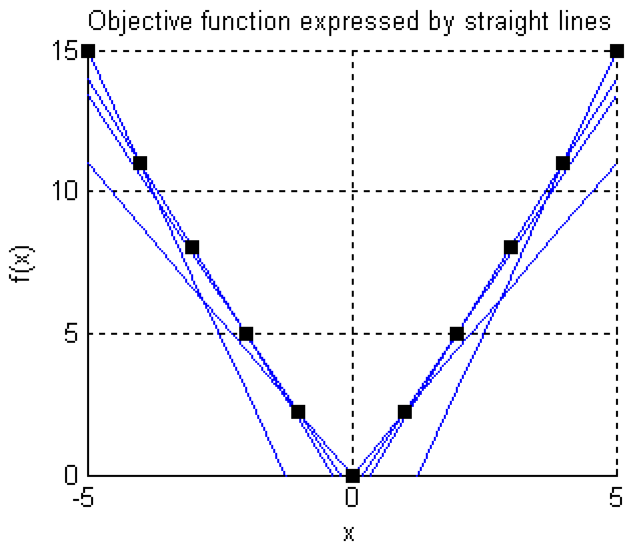


Figure 3.4: Objective function expressed by lines

The method which we are going to use is basically similar to method described in Chapter 1.4.3:

- In the begining we will find polytope X which will include all points x:

$$x \in \mathrm{X} \qquad \mathrm{X} = \left\{\mathrm{x} \middle| \begin{pmatrix}-5\\-5\end{pmatrix} \leq \mathrm{x} \leq \begin{pmatrix}5\\5\end{pmatrix}\right\} \tag{3.6}$$

- Then by separating this polytope we will create a large number of small points on which we will evaluate the function values of all lines.
- Then we will find to which line belonged this maximum value. This way we will be able to define vertices of regions.

The result of this procedure is piecewise affine objective function defined over ten regions (figure 3.5)

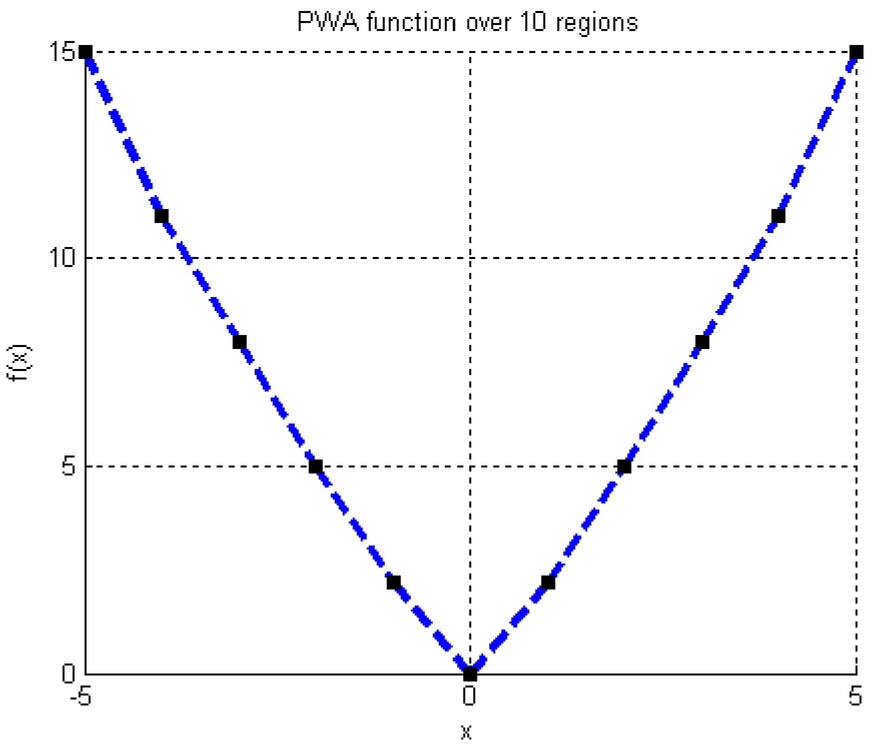


Figure 3.5: Piecewise linear function over 10 regions

3.1.1 **Formulation in MATLAB**

Consider the symmetric objective function in R^1 expressed by these points:

```
x = [-5 -4 -3 -2 -1 0 1 2 3 4 5];
y = [15 11 8 5 2.2 0 2.2 5 8 11 15];
```

If we would like to plot these points, we would get the same result as in figure 3.1. Our first two steps (to obtain figure 3.5) we will achieve by the following optimization procedure:

- First of all we will define each optimized parameter:

```
x_length = length(x);
xlength = x_length - 1;
nx = size(x,1);
%% symbolic parameters
J = sdpvar(xlength,1);
alfa = sdpvar(xlength,nx);
beta = sdpvar(xlength,1);
```

- As we consider that the objective function is symmetrical (and that we are in R^1), in constraints will be included in addition to equations (3.4) and (3.5) the term:

$$\alpha_i = \alpha_j, \tag{3.7}$$

where $i = \overline{1, n/2}$, $j = \overline{n, n/2}$

```
%% constraints
F = [];
% constraints for alpha
for i = 1:xlength
    for j = i+1:xlength + 1
        F = F + [alfa(i,:) <= (y(j)-y(i))/(x(:,j)-x(:,i))];
    end
    if nx == 1 % symmetrical obj
        F = F + [alfa(i,:) == -alfa(xlength + 1 - i,:)];
    end
end

% constraints for beta
for i = 1:xlength
    F = F + [J(i) == alfa(i,:)*x(:,i) + beta(i)];
    if nx == 1 % symmetrical obj
        F = F + [beta(i) == beta(xlength + 1 - i)];
    end
end
```

- The objective function is going to minimize distance of the given point and the function value in corresponding line in that point:

```
%% objective function
obj = 0;
for i = 1:xlength
    obj = obj + (J(i) - y(i))^2;
end
```

- If we have defined whole formulation, then we can solve the problem from which we will obtain optimal slopes and affine terms:

```
%% solve
info = solvesdp(F,obj);
if info.problem ~= 0
    error('Problem is unsolvable !!!')
end
alfa = double(alfa);
beta = double(beta);
```

The last (third) step will be achieved by using function `f = get_explicit_pwa_max(alpha', beta', X)`, where X is a polytope defined by equation (3.6) and represents a convex hull of vertices of all regions:

```
nx = size(x,1);
x_max = zeros(1,nx);
x_min = zeros(1,nx);
for i = 1:size(x,1)
    x_max(i) = max(x(i,:));
    x_min(i) = min(x(i,:));
end
X = polytope([eye(nx);-eye(nx)],[x_max'; -x_min']);

% obtain the explicit representation of f(x) as
%   f(x) = c_j*x+d_j if x \in P_j
f = get_explicit_pwa_max(alpha', beta', X);
```

By ploting the result of this function by function `plot_pwa(f)`, we would get the figure 3.5.

```
plot_pwa(f);
grid on
hold off
```

The whole algorithm can be found in the appendix as m-file `get_PWA`. In table 3.1 the required computational time is listed as a function of dimension:

Dimension	R^1	R^2	R^3
Elapsed time [s]	4.211420	4.384174	4.459774

Table 3.1: Computation time of the algorithm

3.2 Other methods of construction piecewise affine function

There are several other methods by means we are able to construct piecewise affine objective function. One of the most common techniques is based on approximation of a curve (objective function) that will minimize a sum of squared distances between each of those points and the curve. This way we will get a function in a polynomial form (figure 3.2). Then we can make the same procedure as we did in figure 2.3.

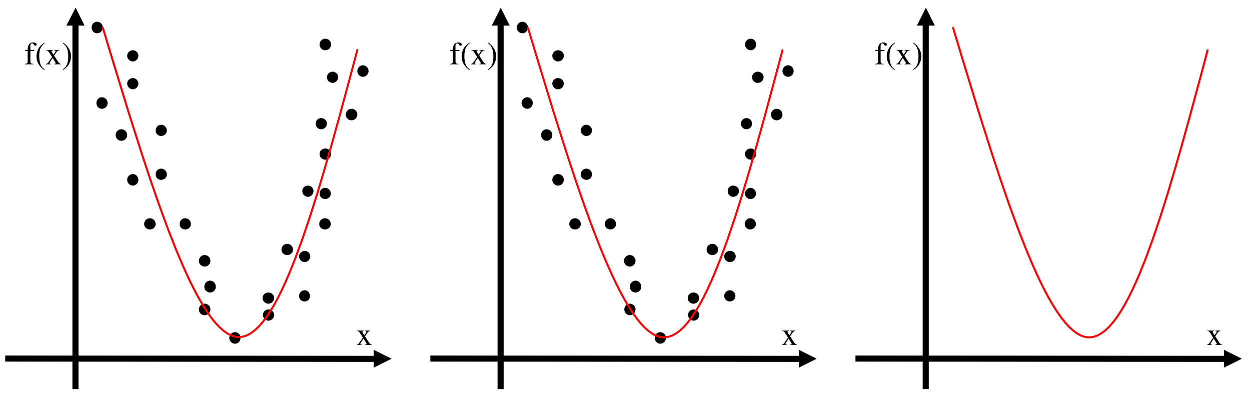


Figure 3.6: Approximating a curve using least squared method

Advantage of this method is that we can have whole objective function (that does not have to be convex) in a form of polynomial (e.g.$f: y = x^2 + 2x + 1$). But if we will construct piecewise affine function from this polynomial, inaccuracy caused by both in approximation and linearization should be higher than in the first (Chapter 3.1) direct method.

4 Boundaries

In order to reduce the number of regions (of the objective function) we have to define a stable area in which we may perform this operation. This area will be constructed through the two piecewise affine functions (boundaries) $\underline{J}$ and $\bar{J}$. In order to demonstrate these two boundaries we are going to use system `lti_1d_unstable`, which we can find in the appendix.

4.1 Lower boundary $\underline{J}$

Since the lower boundary represents the optimal objective function, therefore there can not be a better (lower placed) objective function such as this one.

$$y_i \geq y^*, \tag{4.1}$$

where $\mathrm{K} = \{\kappa_i\}$, $\kappa_i\colon y_i = \tilde{f}(x)$ is a set of suboptimal functions and $f^*\colon y^* = f(x)$ is the optimal function. This optimal function f^*we can get as a result of optimization problem described in Chapter 1.3.4 by equations 1.13. Lower boundary of the system is depicted in figure in figure 4.1.

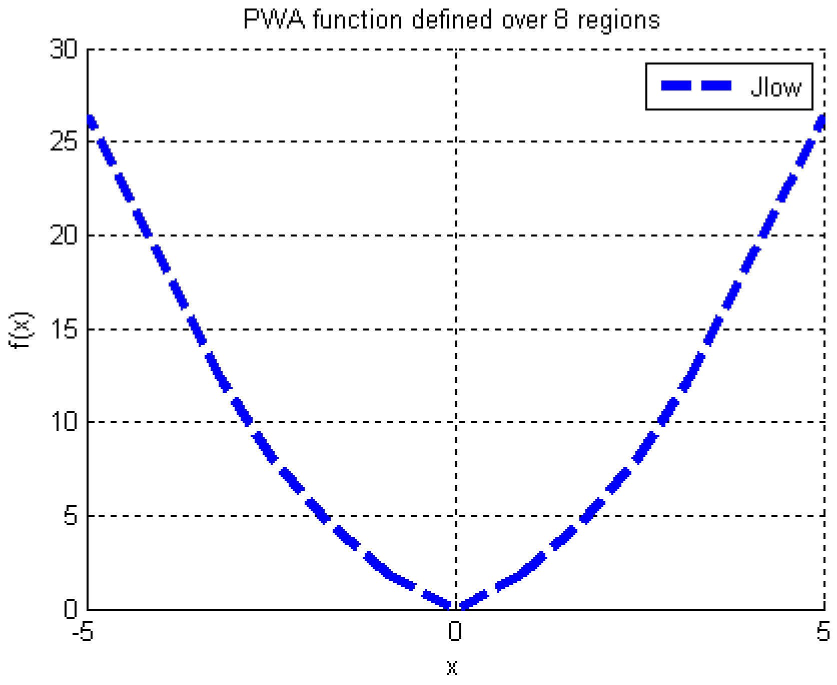


Figure 4.1: Lower boundary

4.2 Upper boundary $\bar{J}$

In this Section we will create an upper boundary for objective function. This boundary combined with lower boundary will define an admissible area in which stability will be guaranteed by Lyapunov function.

4.2.1 Definitions and theorems [4]

Let us consider linear discrete time dynamical systems:

$$x^{+} = Ax + Bu, \tag{4.2}$$

where $x \in R^{n_x}$ is the current state, $u \in R^{n_u}$ is the current control input to the system and x^{+} is the successor state. Then if system (4.2) is controlled by the control law $u_k = \kappa(x)$, the closed loop system is defined as:

$$x^{+} = Ax + B\kappa(x) \tag{4.3}$$

- **Positively invariant set** – A set $\mathcal{X} \subseteq R^{n_x}$ is positively invariant set of system (4.3), if $Ax_k + B\kappa(x_k) \in \mathcal{X}\ \forall x \in \mathcal{X}$
- **K-class function** – A real-valued function $\alpha\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ belongs to class K if it is continuous, strictly increasing and $\alpha(0) = 0$. Where $\mathbb{R}_{\geq 0}$ is the set of non-negative reals.
- **Lyapunov function** – Let $\mathcal{X}$ be a positively invariant set for system (4.3) containing neighborhood $\mathcal{N}$ of the origin in its interior and let $\overline{\alpha}(\cdot)$, $\underline{\alpha}(\cdot)$ and $\beta(\cdot)$ be K-class functions. A non-negative function $V\colon \mathcal{X} \to \mathbb{R}_{\geq 0}$ with $V(0) = 0$ is called a Lyapunov function in $\mathcal{X}$ if:

$$\forall x \in \mathcal{X}: \qquad V(x) \geq \underline{\alpha}(\|x\|) \tag{4.4a}$$

$$\forall x \in \mathcal{N}: \qquad V(x) \leq \overline{\alpha}(\|x\|) \tag{4.4b}$$

$$\forall x \in \mathcal{X}: \qquad V(x^{+}) - V(x) \leq -\beta(\|x\|) \tag{4.4c}$$

- **Asymptotic stability** – If system (4.3) admits a Lyapunov function in $\mathcal{X}$, then the equilibrium point at the origin is asymptotically stable with region of attraction $\mathcal{X}$.

4.2.2 MPC formulation

Let us reformulate (1.13) into:

$$J^*(x) = \sum_{k=0}^{N-1} \ell(x_k, u_k) \tag{4.5a}$$

$$s.t. \quad x_{k+1} = Ax_k + Bu_k \tag{4.5b}$$

$$x_0 = x \tag{4.5c}$$

$$x_t \in \mathcal{X} \tag{4.5d}$$

$$u_t \in \mathcal{U}, \tag{4.5e}$$

where $\mathcal{X}$ and $\mathcal{U}$ are closed and convex polytopes, containing the origin, A and B are systems matrices, $J^*(x)$ is an optimal objective function and $l(x_k, u_k)$ is a k-stage cost of the objective function which can be defined as:

$$\forall x \in \mathcal{X}, \forall u \in \mathcal{U} \qquad \ell(x_k, u_k) = \|Qx_k\|_p + \|Ru_k\|_p, \tag{4.5f}$$

where Q is a positive semi-definite and R is a positive definite weight matrix and p represents a norm. In our case only Taxicab ($p = 1$) or Infinite ($p = \infty$) norm. Assuming that objective function satisfy conditions of the Lyapunov function, by using formulation (4.5) equation (4.4c) can be transformed into:

$$\forall x \in \mathcal{X} \qquad J(x_{k+1}) - J(x_k) \leq \ell(x_k, u_k) \tag{4.6}$$

4.2.3 Creating the upper boundary

Let us consider the lower boundary $\underline{J}$ (respectively optimal objective function f^*) that is a PWA function defined over n-regions. If $\underline{V} = \{[x_1, J(x_1)], \dots, [x_k, J(x_k)]\} \subset \mathbb{R}^{n_x+1}$ is a set of vertices of this function, then $\underline{F} = \{[x_{\underline{F}_1}, J(x_{\underline{F}_1})], \dots, [x_{\underline{F}_{n+1}}, J(x_{\underline{F}_{n+1}})]\} \subset \underline{V}$ is a set of the lower convex hull of V. Based on equation (4.6) a set of vertices of the upper boundary is then computed as:

$$\overline{F} = \{[x_{\underline{F}_1}, J(x_{\underline{F}_1}) + \ell(x_{\underline{F}_1}, u_{\underline{F}_1})], \dots, [x_{\underline{F}_{n+1}}, J(x_{\underline{F}_{n+1}}) + \ell(x_{\underline{F}_{n+1}}, u_{\underline{F}_{n+1}})]\}, \tag{4.7}$$

where $\forall x \in \mathcal{X}, \forall u \in \mathcal{U}$. Vertices $x_{\underline{F}} = x_{\overline{F}}$ from now on we can use only x_F and the stage cost ℓ can be expressed by (4.5f), equation (4.7) can be transformed as:

$$\forall x \in \mathcal{X}, \forall u \in \mathcal{U}, i = \overline{1, n+1} \qquad \overline{F} = \left\{\left[x_{F_i}, J(x_{F_i}) + \left\|Qx_{F_i}\right\|_p + \left\|Ru_{F_i}\right\|_p\right]\right\}, \tag{4.8}$$

where Q is a positive semi-definite and R is a positive definite weighting matrices and p represents a norm. To have all informations about the upper boundary we need to compute slopes α and affine terms β from the following equation:

$$\forall x \in \mathcal{X}, i = \overline{1, n} \qquad \begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} = \begin{pmatrix} x_{F_i}^T & 1 \\ x_{F_{i+1}}^T & 1 \end{pmatrix}^{-1} \begin{pmatrix} \varphi_i \\ \varphi_{i+1} \end{pmatrix}, \tag{4.9}$$

where n is the number of regions and $\varphi_i = J(x_{F_i}) + \left\|Qx_{F_i}\right\|_p + \left\|Ru_{F_i}\right\|_p$. In MATLAB this procedure to obtain data of the upper boundary from the informations of the lower boundary is implemented in function `prepare_data`.

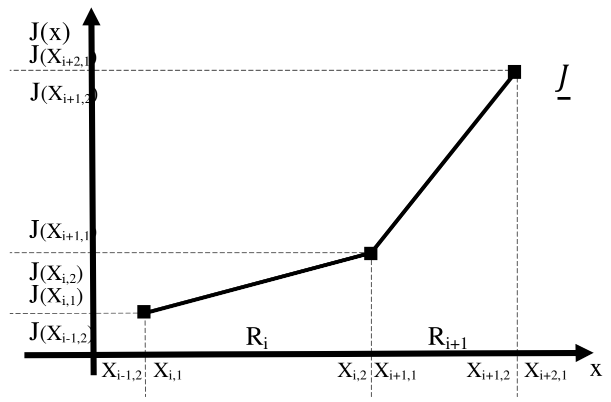


Figure 4.2a: Vertices of the lower boundary

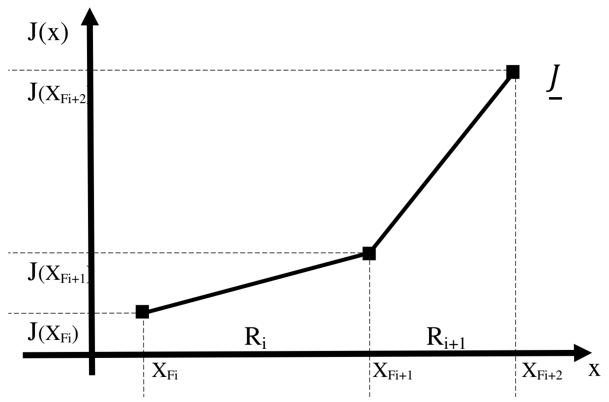


Figure 4.2b: Lower convex hull of V

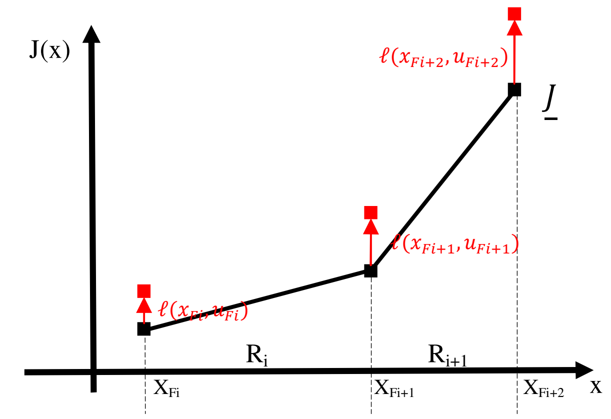


Figure 4.2c: Vertices of the upper boundary

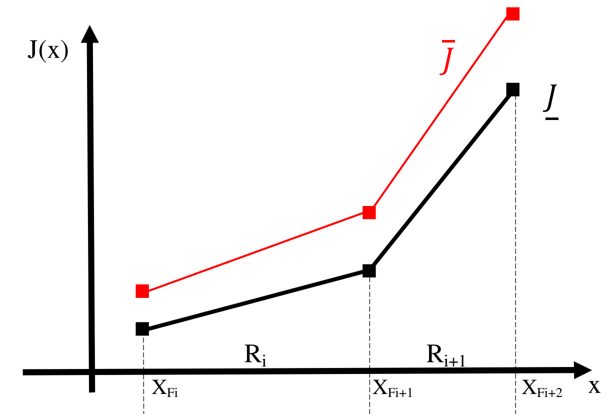


Figure 4.2d: Upper boundary

4.3 Admissible stable area

Let us assume that the lower $\underline{J}$ and the upper $\bar{J}$ boundary is well known, then the admissible area ϱ, where stability is guarantee based on Lyapunov function, is defined as:

$$\varrho = \{u | \, Jlow(x) \leq u \leq Jup(x), \forall x \in \mathcal{X}\} \tag{4.10}$$

Now when the stable area is defined, we are able to fit a new objective function $\tilde{f}$ with a smaller number of regions. We will have to keep in mind that the stability is guaranteed only if $\tilde{f}(x) \in \varrho, \forall x \in \mathcal{X}$ is satisfied.

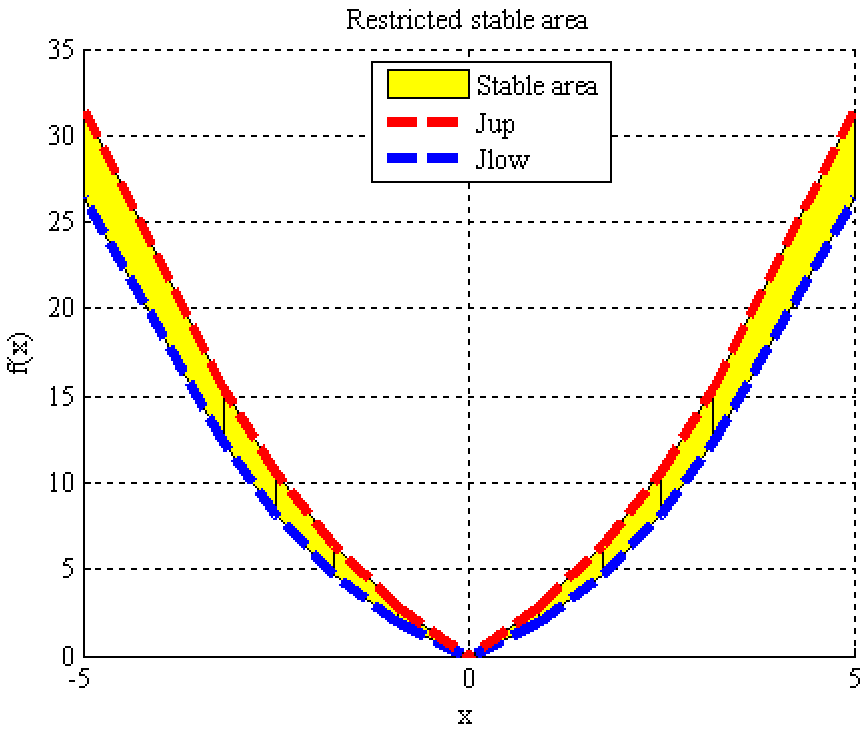


Figure 4.3: Restricted stable area

5 Fitting of the new objective function

Now what we are going to do is to fit a new objective function over fewer regions as the first objective function has had ($\tilde{R}_i \leq R_i$). Since the area where we can perform this operation is strictly restricted by two boundaries (figure 4.3), then new piecewise affine objective function has to satisfy equations:

$$\forall x: \quad f(x) \leq \bar{J}(x) \tag{5.1a}$$

$$\forall x: \quad \underline{J}(x) \leq f(x), \tag{5.1b}$$

where f is new fitted objective function, $\underline{J}$ is a lower boundary and $\bar{J}$ is a upper boundary. Since the explicit definition of a function is:

$$f(x) = \max_k(\alpha_k^T x + \beta_k), \tag{5.2}$$

therefore in order to satisfy the equation (5.1) we have to find coefficients α_i and β_i $i = \overline{1,k}$ for fixed value of k .

5.1 Objective function and the upper boundary

Assuming that data of the upper boundary is well known (defined as piecewise affine function with slopes C_U and affine terms D_U), equation (5.1a) can be further formulated as:

$$\forall i, \forall x \in R_i: \qquad \max_k(\alpha_k^T x + \beta_k) \leq C_{U,i}^T x + D_{U,i}, \tag{5.3}$$

where R_i is i-th region, α_k is k-th slope and β_k is k-th affine term of the objective function. From equation (5.3) follows that maximum value of the new fitted objective function has to be lower or equale than the function value of the upper boundary. On the right side, it would be the same as if we would say that each function value on the left side must be lower or equal than the function value on the right side. Equation (5.3) has operator less or equal ($\leq$) and as both of those functions (f, $\bar{J}$) are convex, necessary and sufficient condition are satisfied and the equation can be modified as:

$$\forall i, \forall x \in R_i, \forall k: \qquad \alpha_k^T x + \beta_k \leq C_{U,i}^T x + D_{U,i} \tag{5.4}$$

Equation (5.4) can be applied on vertices of the upper boundary (instead of whole axis scale) and the formulation will be equivalent to:

$$\forall i, \forall v_j \in vert(R_i), \forall k: \qquad \alpha_k^T v_j + \beta_k \leq C_{U,i}^T v_j + D_{U,i} \tag{5.5}$$

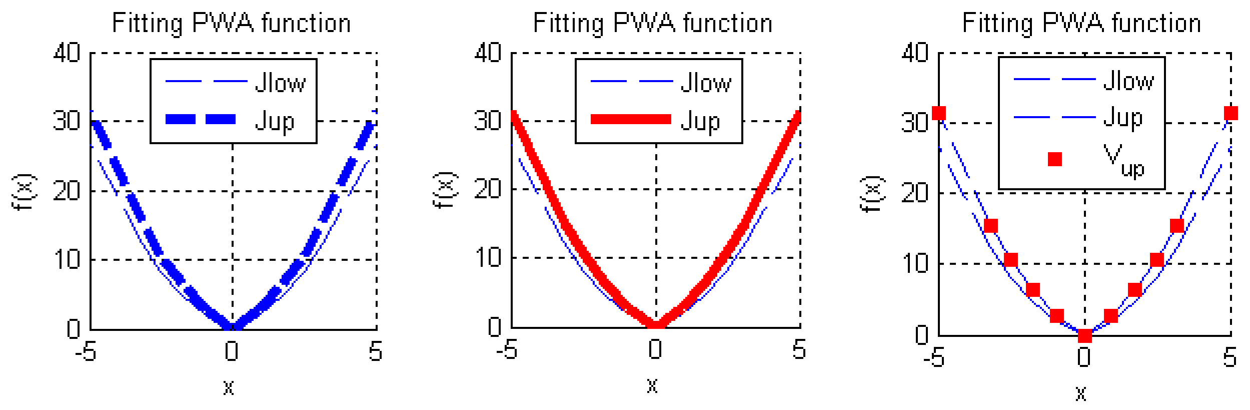


Figure 5.1a: Fitting PWA function restricted by upper boundary

Figure 5.1b: Fitting PWA function restricted by upper boundary on the whole axis x

Figure 5.1c: Fitting PWA function restricted by upper boundary on the verteces

5.2 Objective function and the lower boundary

Let us say that data of the lower boundary is well known (defined by matrix C_L for slopes and matrix D_L for affine terms). Equation (5.1b) can be then further formulated as:

$$\forall i, \forall x \in R_i: \qquad C_{L,i}^T x + D_{L,i} \leq \max_k(\alpha_k^T x + \beta_k) \tag{5.6}$$

The unknown variables are on the other side, the operator is now bigger or equal ($\geq$) than the function value of lower boundary. In this case only necessary condition is satisfied and by this reason implication is used between equations (5.5) and (5.6):

$$\forall i, \forall v_j \in vert(R_i): \qquad C_{L,i}^T v_j + D_{L,i} \leq \max_k\left(\alpha_k^T v_j + \beta_k\right) \tag{5.7}$$

In other words if equation (5.7) is met, then equation (5.6) does not have to be met. Equation (5.7) is extreamly difficult to compute, because maximum value is being used.

$$t = \max\,(x_i) \tag{5.8}$$

The result from equation (5.8) t is achieved by computation following problem:

$$\forall i: \qquad -M(1-\delta_i) \leq t - x_i \leq M(1-\delta_j) \qquad (5.9a)$$

$$\forall i, \forall j, i \neq j: \qquad x_j \leq x_i + M(1-\delta_i), \qquad (5.9b)$$

where $\sum_{l=0}^{i} \delta_l = 1$ and $\delta_l = \{0,1\}$.

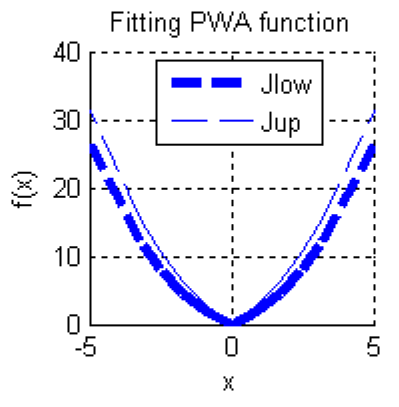


Figure 5.2a: Fitting PWA function restricted by lower boundary.

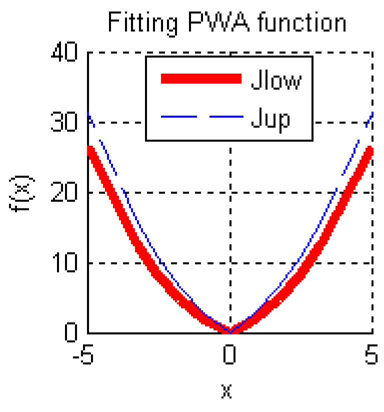


Figure 5.2b: Fitting PWA function restricted by lower boundary on the whole axis x

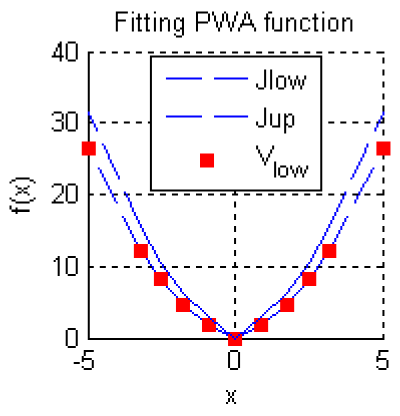


Figure 5.2c: Fitting PWA function restricted by lower boundary on the verteces

5.3 Creating a new objective function

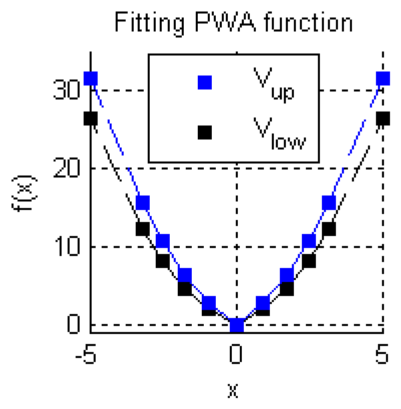


Figure 5.3a: Fitting PWA function restricted by vertices of lower and upper boundary.

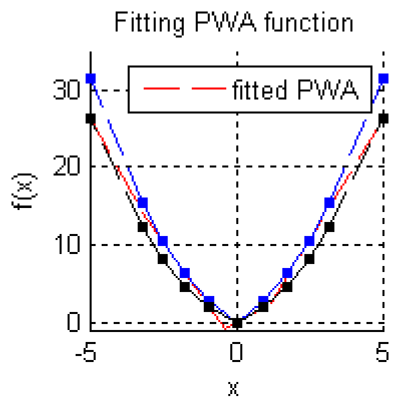


Figure 5.3b: Badly fitted PWA function

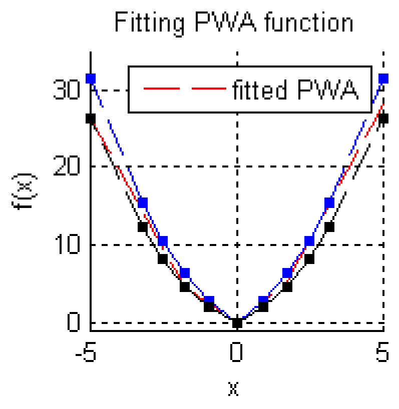


Figure 5.2c: Properly fitted PWA function.

In this Chapter we are going to creating a new objective function. Basically we are going to fit new affine lines (regions) restricted by vertices of lower and upper boundary (figure 5.3a). Based on these conditions the optimizer can fit badly piecewise affine function as we can see in the figure 5.3b. This wrong approximation is allowed by using implication between equations (5.6) and (5.7). While equation (5.7) is satisfied, another equation (5.6) is not. Properly fitted piecewise affine function is illustrated in figure 5.2c.

5.4 Implementation in MATLAB

Let us consider that data about system and its boundaries are known. To obtain new fitted objective function we will define function:

```
 [alpha,beta] = fit_PWA(Jup,Jlow,V,K),
```

where `Jup` is an upper boundary, `Jlow` is a lower boundary, `V` are verteces of regions and `K` is number of regions of the new fitted function which we have to define. Outputs of this function are slopes `alpha` and affine terms `beta`. This function is basically MATLAB representation of the equations (5.5) and (5.7). We have to keep in mind that even equation (5.7) was used, YALMIP will also compute equations (5.9).

```
function [alpha,beta] = fit_PWA(Jup,Jlow,V,K,x_up,x_low)
nx = size(V{1},2);
% obj. function
obj = 0;
% constraints
F = [];
% symbolic values
alpha = sdpvar(nx,K,'full');
beta = sdpvar(1,K);

for i = 1:length(V)
    % Jup
    for j = 1:length(V{i})
        F = F + [(V{i}(j,:)*alpha + beta) <= ...
            V{i}(j,:)*Jup.C{i} + Jup.D{i}];
    end
    % Jlow
    for j = 1:length(V{i})
        F = F + [max(V{i}(j,:)*alpha + beta) >= ...
            V{i}(j,:)*Jlow.C{i} + Jlow.D{i}];
    end
end

%% solve
info = solvesdp(F,obj);
if info.problem ~= 0
    error('Problem is unsolvable !!!')
end
alpha = double(alpha);
beta = double(beta);
```

6 Certification

As we know from Chapter 5.2, equation (5.7) is only necessary condition to expression (5.6) and so we can get bad approximation (Figure 5.3b), therefore after every fitted objective function we have to check if the equation (5.6) is also satisfied. For the fulfilling of the constraints we will use certification.

6.1 Certification of the upper boundary

Equation (5.5) is necessary and sufficient condition to expression (5.3), certification for the upper boundary is unnecessary. Anyway just to be sure we will apply this certification which can be formulated as:

$$\forall i, \forall x \in R_i \qquad \max_k(\alpha_k^T x + \beta_k) > C_{U,i}^T x + D_{U,i}, \tag{6.1}$$

where x is a optimization variable which belongs to R_i region, α_k is set of slopes and β_k is set of affine terms of the objective function while $C_{U,i}$ are slopes and $D_{U,i}$ are affine terms of the upper boundary. If exist any solution x which will satisfy equation (6.1) it means that at the point with coordinates x function value of the fitted objective function is greater then function value of the upper boundary. For this reason we have to increase the number of the upper boundary vertices by this point x. Then we will try to fit another objective function, but with greater number of vertices. Since between equations (5.3) and (5.5) is equivalence, equation (6.1) should not be ever satisfied.

6.1.1 Formulation in MATLAB

For the certification of the upper boundary (based on the equation (6.1)) we can create function:

```
[x] = cert_Jup(alpha,beta,Jup,V),
```

where alpha are slopes and beta are affine terms of the fitted objective function, Jup are piecewise affine data of the upper boundary and `V` is vector of vertices (regions). Output `x` is a point where equation (6.1) is satisfied. We have to keep in mind that if optimized problem is infeasible then in parameter *`x`* will not be saved a value. So after command *`double(x)`* zero will be received. Then based on the result `x` will be printed the answer.

```
function [x] = cert_Jup(alpha,beta,Jup,V)
nx = size(alpha,1);

for i = 1:length(V)
    x = sdpvar(nx,1);

    obj = 0;

    F = [];
    F = F + [ismember(x, Jup.R(i))];
    F = F + [max(alpha'*x + beta') - (Jup.C{i}'*x + Jup.D{i}) > 1e-5];

    info = solvesdp(F,obj);

    x = double(x);
    if info.problem == 0 && x ~= 0 , break;end
end

if x == 0
    disp('cert_Jup: Alpha and beta were certificated !!!')
else
    disp('cert_Jup: Alpha and beta were not certificated !!!')
end
```

6.2 Certification of the lower boundary

By certification of the lower boundary we are going to find out if fitted objective function is above lower boundary. This procedure is necessary because between equations (5.6) and (5.7) is applied implication (5.6 ⇒ 5.7) and so equation (5.7) is only necessary condition for equation (5.6). This can cause that fitted objective function will not be in the restricted area (figure 5.3b). Certification for the lower boundary can be formulated as:

$$\forall i, \forall x \in R_i, \forall k \qquad \alpha_k^T x + \beta_k < C_{L,i}^T x + D_{L,i}, \tag{6.2}$$

where x is optimized point which belongs to R_i region, α_k is set of slops and β_k is set of affine terms of the objective function while $C_{L,i}$ are slopes and $D_{L,i}$ are affine terms of the lower boundary. Since we want to find point with the greatest deviation from the lower boundary equation (6.2) will be transformed into following optimization problem:

$$\forall i, \forall x \in R_i \qquad \mathrm{d}^* = \min\left(\varepsilon - (C_{L,i}^T x + D_{L,i})\right) \tag{6.3a}$$

$$\varepsilon \geq \max_k \left(\alpha_k^T x + \beta_k\right), \tag{6.3b}$$

where ε is the maximum function value of the fitted objective function, x is optimized point and d^* is an optimal result of this problem. This set of equations can be further reformulated as:

$$\forall i, \forall x \in R_i \qquad \mathrm{d}^* = \min\left(\varepsilon - \left(C_{L,i}^T x + D_{L,i}\right)\right) \tag{6.4a}$$

$$\forall k \qquad \varepsilon \geq \alpha_k^T x + \beta_k \tag{6.4b}$$

If the result of this problem will be negative then objective function is badly fitted with the greatest deviation in point x. Let us apply equations (6.4) on figure 5.3b than as a result we will get figure 6.1a. In this figure we can clearly see that new objective function is badly fitted what will equation (6.4) find out by negative value of variance $\mathrm{d}^* < 0$. For this reason we have to increase the number of the lower boundary vertices by this point $x = -0{,}4$ (figure 6.1b). Then we can try to fit another objective function while this time we can be sure that equations (6.4) will not be satisfied in the point x (and in its close vicinity). This procedure should be repeated until nonnegative variance will be found. It is possible that equations (6.4) will still have negative variance which can cause that problem (5.7) illustrated in figure 5.2c will be transformed into problem (5.6) illustrated in figure 5.2b.

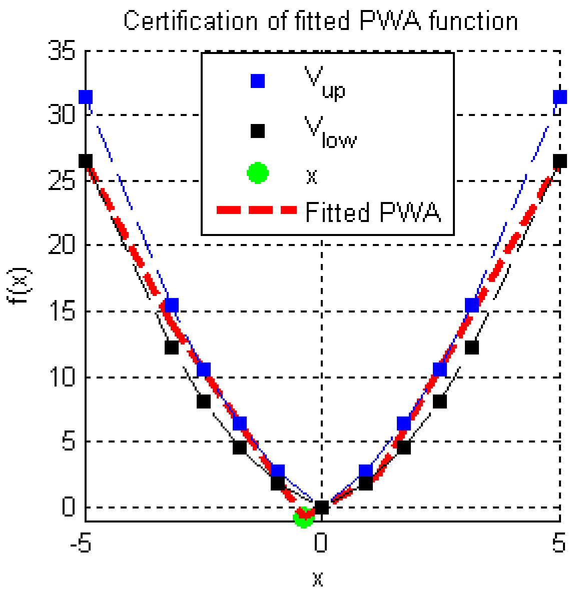


Figure 6.1a: Certification of the fitted funciton

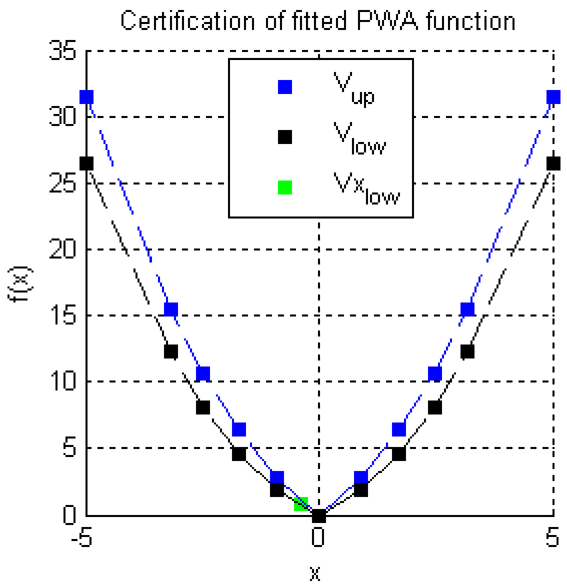


Figure 6.1b: Result of the certification

6.2.1 Formulation in MATLAB

For the certification of the lower boundary (based on equations (6.4)) we can create function:

```
[x,d] = cert_Jlow(alpha,beta,Jlow,V),
```

where alpha are slopes and beta are affine terms of the fitted objective function, Jlow are piecewise affine data of the lower boundary and `V` is vector of vertices (regions). Output `d` represents variance of the fitted objective function and lower boundary in a point with axis `x`. Based on this variance is also printed to the commandline the answer.

```
function [x,d] = cert_Jlow(alpha,beta,Jlow,V)
nx = size(alpha,1);
for k = 1:length(V)
    x = sdpvar(nx,1);
    eps = sdpvar(1,1);

    obj = eps - (Jlow.C{k}'*x + Jlow.D{k});

    F = [];
    F = F + [ismember(x, Jlow.R(k))];
    for i = 1:length(alpha)
        F = F + [eps >= alpha(:,i)'*x + beta(i)];
    end

    info = solvesdp(F,obj);
    if info.problem ~= 0, error('Problem is unsolvable !!!'),end

    x = double(x); d = double(obj);
    if d < -1e-6, break; end
end

if d >= -1e-6
    disp(sprintf('cert_Jlow: Alpha and beta were certificated !!!'))
else
    disp('cert_Jlow: Alpha and beta were not certificated !!!')
    disp(sprintf('           In point x = %d is difference = %d.',x,d))
end
```

6.3 Formulation in MATLAB

Until now we have certification of upper and lower boundary. But as it was already mentioned in both certifications if any of these function will find out x that will satisfy equation (6.1) respectively (6.2), then we have to increase number of vertices by this point x. After receive new extended vector of vertices we will fit another objective function and then certification can start again. This cycle we should perform until fitted objective function will be in the restricted area.

Because of certification of the lower boundary problem it is possible that using equations (6.4) illustrated in figure 5.2c will be transformed into problem (5.6) illustrated in figure 5.2b. For this reason we will define the maximum number of cycles. Since we are going to send extending vertices (`x_up, x_low`) to the function `fit_PWA` we have to upgrade it by adding additional inputs:

```
[alpha,beta] = fit_PWA(Jup,Jlow,V,K,x_up,x_low)
```

And by adding additional constraints:

```
for i = 1:length(V)
    % Jup extended vertices
    for j = 1:length(x_up)
        F = F + [(x_up(j,:)'*alpha + beta) <= ...
            x_up(j,:)'*Jup.C{i} + Jup.D{i}];
    end
    % Jlow extended vertices
    for j = 1:length(x_low)
        F = F + [max(x_low(j,:)'*alpha + beta) >= ...
            x_low(j,:)'*Jlow.C{i} + Jlow.D{i}];
    end
end
```

To ensure functionality of the mentioned cycles we will use the function:

```
[alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N),
```

where `Jup` and `Jlow` are piecewise afiine data for boundaries, `V` is vector of all vertices (regions), `K` is number of regions of the new fitted function, `x_up` and `x_low` vectors of extended vertices (while in the beginning there are empty) and `N` represents number of maximum cycles. Outputs from this function are slopes `alpha` and affine terms `beta` of the final fitted objective function and vectors of extended vertices `x_up` and `x_low`.

```
function [alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N)
[alpha,beta] = fit_PWA(Jup,Jlow,V,K,x_up,x_low);

[x] = cert_Jup(alpha,beta,Jup,V); % cert_Jup: f(x) <= Jup(x)
if x ~= 0
    x_up = [x_up; x];
    [alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);
End

[x,d] = cert_Jlow(alpha,beta,Jlow,V); % cert_Jlow: Jlow(x) <= f(x)
if d < -1e-6
    x_low = [x_low; x];
    if length(x_low) < N + 1
        [alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);
    elseif length(x_low) == N
```

```
        fprintf('!!! Lack of iteration steps !!!\n')
    end
else
    fprintf('Jlow(x) <= f(x) <= Jup(x) certified\n')
end
```

In the beginning new objective function will be fitted (`fit_PWA`). Then certification of the upper boundary will be applied (`cert_Jup`). Based on the result `x` this function may add another vertex and then recursively run function `Jlow_Jup` once again. If fitted objective function will be under upper boundary then certification of the upper boundry will be satisfied and we can start certificate lower boundary (`cert_Jlow`). Based on variance `d` we may add another vertex and again recursively run function `Jlow_Jup`. If both certificatins are satisfied (or number of cycles will run out) we will print the result of the certification.

To plot last fitted objective function we can use function:

```
[VV] = draw_PWA(alpha, beta, X, Jpoly),
```

where `alpha` and `beta` are slopes and affine terms of fitted objective function, `X` represents edges of axis and `Jpoly` is a structure where slopes and affine terms of both boundaries have been saved. To obtain explicit representation of fitted objective function we will use function *`get_explicit_pwa_max`*. When boundaries have been plotted by using command `plot` we can plot mentioned fitted objective function through function `plot_pwa`. In the end we will compute vector of all region verteces as the output of this main function `draw_PWA`.

```
% function [VV] = draw_PWA(alpha, beta, X, Jpoly)
% alpha - slope (f(x) = alpha*x + beta)
% beta  - affine term (f(x) = alpha*x + beta)
% X     - Polytop representing range of each axis (in matrix X)
% VV    - vertices of each new region (from new alpha and beta)
function [VV] = draw_PWA(alpha, beta, X, Jpoly)

% obtain the explicit representation of f(x) as
%   f(x) = c_j*x+d_j if x \in P_j
f = get_explicit_pwa_max(alpha, beta, X);

if gcf==1, plot(Jpoly, 'y'); end

hold on, plot_pwa(f); grid on, hold off

% output
for i = 1:length(f.R), VV{i} = extreme(f.R(i)); end
```

7 Control law

In Chapter 1.4.2 was mentioned that each piecewise affine objective function must be associated with piecewise affine control law to achieve successfully control of any process. System does not care about objective function in the certain state, it requires only optimal input from controller to move forward. In this Section we will describe how to obtain control law from objective function.

7.1 Approximate simplical control law [5]

Let $V = \{[x_1, f(x_1)], \dots, [x_k, f(x_k)]\} \subset \mathbb{R}^{n+1}$ be a set of vertices of the fitted objective function and u_i be the optimizer for vertex $[x_i, f(x_i)]$. If $F = \{[x_{F_1}, f(x_{F_1})], \dots, [x_{F_{n+1}}, f(x_{F_{n+1}})]\} \subset V$ are the vertices contained in a facet of the lower convex hull of V, then the approximate critical region defined by F is $\tilde{R}_F := conv\, \pi_x V$ and the control law is given as:

$$\forall x \in \tilde{R}_F \qquad \tilde{u}(x) := U_F X_F^{-1} \begin{pmatrix} x \\ 1 \end{pmatrix}, \tag{7.1}$$

where

$$U_F := \begin{bmatrix} u_{F_1} & \cdots & u_{F_{n+1}} \end{bmatrix}$$

$$X_F := \begin{bmatrix} x_{F_1} & \cdots & x_{F_{n+1}} \\ 1 & \cdots & 1 \end{bmatrix}$$

Equation (7.1) defines the approximate control law as the interpolation of the optimal control action given at the vertices of each approximate critical region.

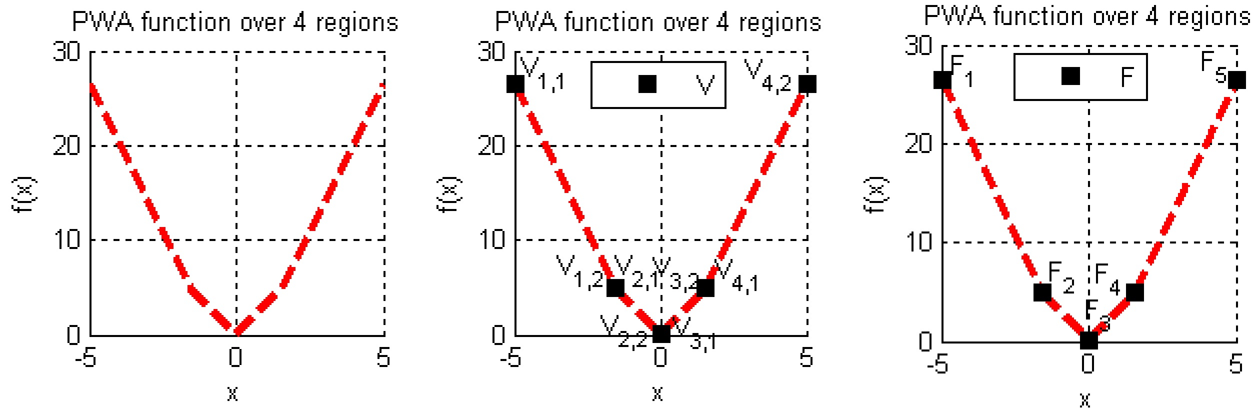


Figure 7.1a: Certificated PWA objective function

Figure 7.1b: Vertices of all regions of PWA objective functions

Figure 7.1c: Vertices made of the lower convex hull of vector V

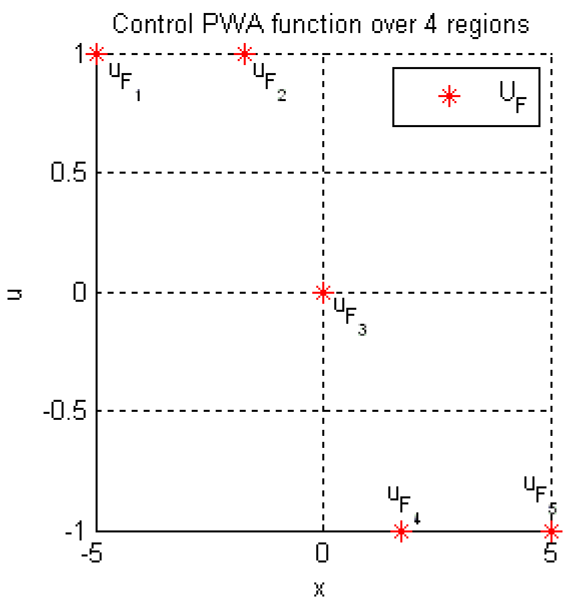


Figure 7.2a: Optimized inputs of vector V

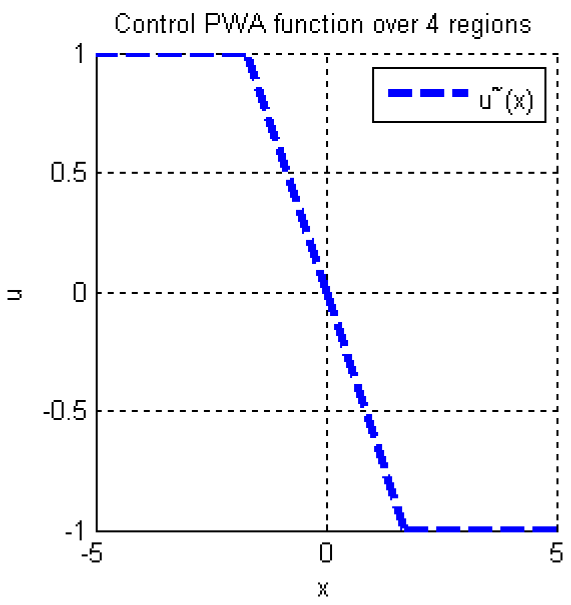


Figure 7.2b: Control law associated with obj. function

7.2 Formulation in MATLAB

First step to implement equation (7.1) will be to prepare vector of the optimized inputs U_F for verteces V. For this purpose we use the function:

```
[Vu] = get_Vu(VV,sysStruct,probStruct),
```

where `sysStruct` and `probStruct` are structures of the initial problem (in our case it is problem defined in `lti_1d_unstable`) and `VV` are vertices of each region (`VV` $= V$). Output of this function is optimized inputs (`Vu` $= U_F$) and it will be received as a result of the optimization of the particular norm. Overview of the most used norms is mentioned in Chapter 1.3.2, but in MATLAB we will use matrix formulation. Full description of each function (`norminf, norm1`) is in the appendix.

```
% Vu -> cell of the optimum inputs in all vertices
function [Vu] = get_Vu(VV,sysStruct,probStruct)
% objective function and constraints
if probStruct.norm == inf
    [Vu] = norminf(VV,sysStruct,probStruct);
elseif probStruct.norm == 1
    [Vu] = norm1(VV,sysStruct,probStruct);
else
    fprintf('!!! Norm "%d" is not allowed !!!\n',probStruct.norm);
end
```

If we take a better look to the equation (7.1) we will see that result of the multiplication $U_F X_F^{-1}$ is a matrix which represents slopes and affine terms of the control law. This matrix will be output of the following function:

```
[VU,Vu] = get_u(VV,sysStruct,probStruct),
```

where inputs are just the same as in the function `get_Vu` (already described), first output `Vu` represents vector of optimized inputs while the second one represents mentioned matrix of slopes and affine terms of the control law (VU $= U_F X_F^{-1}$).

```
% VU -> cell of the alpha and beta for inputs
%       (VU{i} = [a1_i, a2_i...;b1_i, b2_i...;])
% Vu -> cell of the optimum inputs in all region vertices  from VV
function [VU,Vu] = get_u(VV,sysStruct,probStruct)
[Vu] = get_Vu(VV,sysStruct,probStruct);

nx = size(sysStruct.A,2);      % number of states

for i = 1:length(VV)
    for j = 1:nx
        X = [VV{i}(:,j)'; ones(1,length(VV{i}(:,j)))];
        U = [Vu{i}(:,j)'];
        C = U*X^-1;
        VU{i}(:,j) = C';
    end
end
```

To plot the final control law can be again used function `draw_PWA` or alternatively can be used following function:

```
plot_PWA_u(VV,Vu),
```

where `VV` is vector of vertices of objective function and Vu is vector of vertices of the control function. If this function will meet with more than two dimensional problem, it will write warning using command `fprintf`.

```
function plot_PWA_u(VV,Vu)
nx = size(VV{1},2);

if nx == 1
    figure
    hold on
    xlabel('x')
    ylabel('u')
    title(sprintf('Control PWA function over %d regions',length(VV)))

    plot([VV{:}],[Vu{:}],'--b','LineWidth',3)
    legend('u = f(x)')
    grid

elseif nx == 2
    temp = ceil(length(VV)/7);
    color = repmat({'b','g','r','c','m','y','k'},1,temp);

    figure
    hold on
    xlabel('x1')
    ylabel('x2')
    zlabel('u')
    title(sprintf('Control PWA function over %d regions',length(alpha)))
    grid

    % for u1
    for i = 1:length(VV)
        x = VV{i}(:,1);
        y = VV{i}(:,2);
        z = Vu{i}(:,1);

        patch(x,y,z,color{i})
    end

    if size(Vu{1},2) == 2
        figure
        hold on
        xlabel('x1')
        ylabel('x2')
        zlabel('u2')
        title(sprintf('Control PWA function over %d regions',length(alpha)))
        grid

        % for u2
        for i = 1:length(VV)
            x = VV{i}(:,1);
            y = VV{i}(:,2);
            z = Vu{i}(:,2);

            patch(x,y,z,color{i})
        end
    end

else
    fprintf('!!! Cant plot if nx > 2 !!!\nNumber of states: nx = %d\n',nx)
end
```

8 Examples

In Chapter 2 we have analyzed that if we want to control any process equation (2.1) must be satistied. To achieve that we have to sometimes decrease computation time at the expense of optimality, especially when fast process has to be controlled. By decreasing the number of regions we will lower PWA data requirements for process control, but on the other hand performance will be decreased as well (figure 2.4). In this Section we will illustrate a few examples where this method will be used.

8.1 Example 1 (lti_1d_stable)

Let us have an example of linear time-invaiant, one dimensional stable problem defined as `lti_1d_stable`. The result of the call `lti_1d_stable` can be seen in the figure 8.1a. Our next step will be to find upper boundary. Lower boundary is already known since it is represented by optimal solution of this problem (figure 8.1a). Both boundaries are plotted in figure 8.1b, where the restricted (stable) area is being highlighted. Now if we look at this figure 8.1b more closely we should see that this whole problem can be defined even over two regions ($K = 2$) and as this problem looks realy simple five iteration steps should be enough ($N = 5$). This way we can reduce data required to store slopes and affine terms to a quarter (and approximately computation time as well (2.5)). To get this new simplified objective function we will use function `Jlow_Jup`, where certification is included. If new problem will by certificated or if we will run out of iteration steps, final objective function will be plotted (figure 8.1c). As a last step to control this system with new suboptimal objective function we need to construct a control law. This procedure is described in Chapter 7 and so we should know that if we want to get slopes and affine terms of the control law we need to call function `get_u` and the function `plot_PWA_u` (alternatively function `draw_PWA`) will plot it (figure 8.1e). From Chapter 2 we know that by decreasing number of regions we will also decrease computation time required to assign correct input to the system based on current state but at the expense of loosing performance. This loss can be seen while comparing optimal and suboptimal control laws (figure 8.1f).

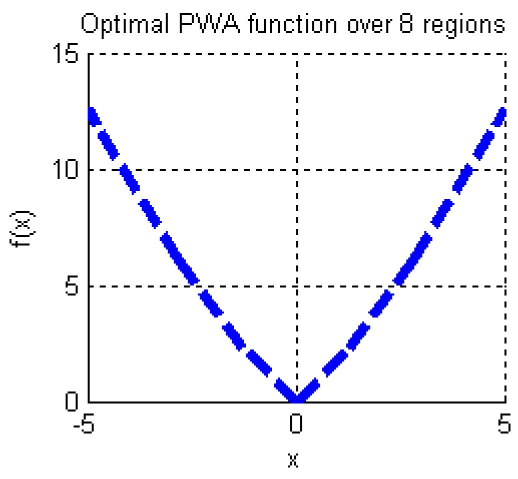


Figure 8.1a: Optimal PWA objective function

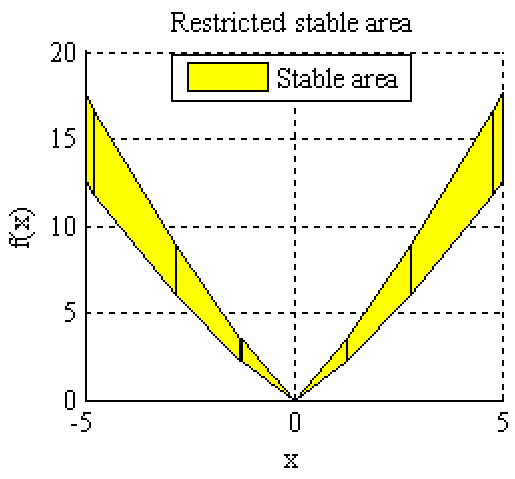


Figure 8.1b: Restricted stable area

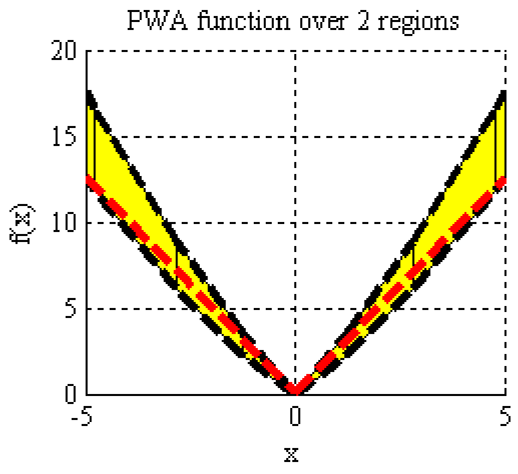


Figure 8.1c: Fitted PWA objective function

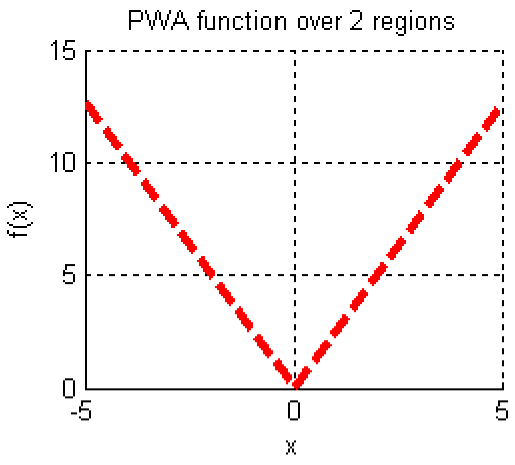


Figure 8.1d: Suboptimal PWA objective function

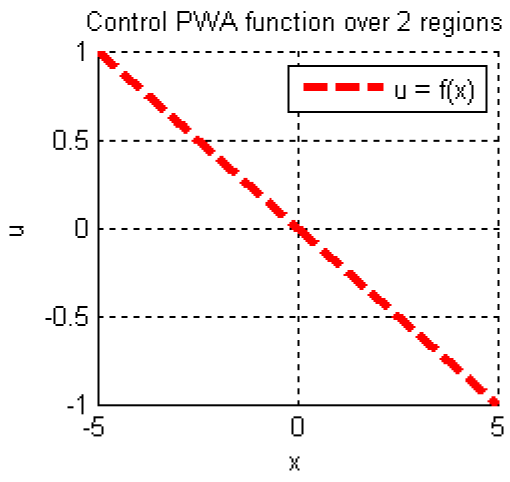


Figure 8.1e: Control PWA function

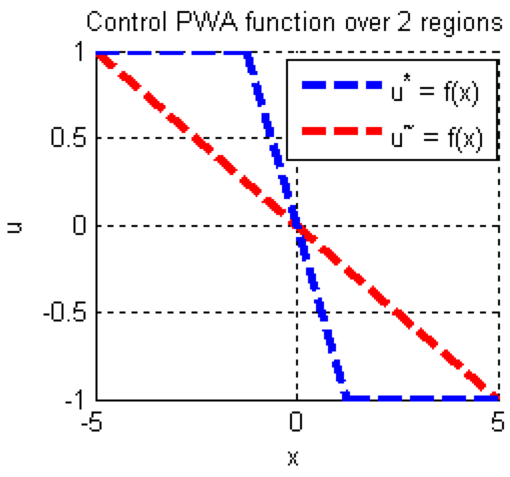


Figure 8.1f: Comparison of the control laws

8.1.1 Formulation in MATLAB

System `lti_1d_stable` is defined as:

```
clear sysStruct probStruct

sysStruct.A = 0.8;
sysStruct.B = 1;
sysStruct.C = 1;
sysStruct.D = 0;
sysStruct.umax = 1;
sysStruct.umin = -1;
sysStruct.xmax = 5;
sysStruct.xmin = -5;

probStruct.Q = 1;
probStruct.R = 1;
probStruct.norm = 1;
probStruct.N = 5;

nx = mpt_sysStructInfo(sysStruct);

ctrl = mpt_control(sysStruct, probStruct)
```

To create a new suboptimal control law of the system mentioned above we need to call following functions:

```
close all
clear all
clc

lti_1d_stable;
prepare_data;

N = 5; % number of iteration steps
K = 2;  % number of approx regions
x_up = []; % empty extending vertices for the upper boundary
x_low = []; % empty extending vertices for the lower boundary

[alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);

VV = draw_PWA(alpha, beta, X, Jpoly);

[VU,Vu] = get_u(VV,sysStruct,probStruct);

plot_PWA_u(VV,Vu);
```

8.2 Example 2 (lti_1d_unstable)

Let us have an example of linear time-invariant, one dimensional unstable problem defined as `lti_1d_unstable`. By calling m-file with this name as this system we will get an explicit form of this problem which can be graphically seen in the figure 8.2a. When the lower boundary will be found we can highlight stable area. For this purpose we have prepared another m-file `prepare_data` (figure 8.1b). From figure 8.1b we will try to guess the minimal number of regions which can be this system defined ($K = 4$). Since this problem is a little bit more difficult as example 1 the number of iterations will be doubled ($N = 10$). To get a new simplified objective function we will use function Jlow_Jup in which certification is included (figure 8.2d). The only thing what is left to do is to extract control law from the objective function. This step is achieved by function `get_u` and then after using another function `plot_PWA_u` we will plot it (figure 8.2e). In the end we can compare optimal and suboptimal control laws to see the lost of the performance caused by this method (figure 8.2f).

Since optimal objective function is symmetrical we can try to even simplify this computation. If we split objective function into two halfs ($h_1 \in \langle -5,0 \rangle, h_2 \in \langle 0,5 \rangle$), then the result of the first half h_1 will be also result for the other half, but with opposite sign of the slopes. This way fitting and the certification should by simplified by a half. To prove this statement we can take a look at the table 8.1. Time-inequality of these two methods is caused by MATLAB, which had in the case of the asymmetries to use more extended vertices.

Operation	Algorithm that does not use symmetry	Algorithm that use symmetry
Prepare explicit data	5.1 [s]	5.1 [s]
Fit a new objective function	9.2 [s]	1.1 [s]
Number of extended vertices	3	0
Plot objective function	0.2[s]	0.2 [s]
Extract control law	0.7 [s]	0.6 [s]
Plot control law	0.1[s]	0.1 [s]
Total time	**15.3 [s]**	**7.1 [s]**

Table 8.1: Comparing time efficiency by using symmetry

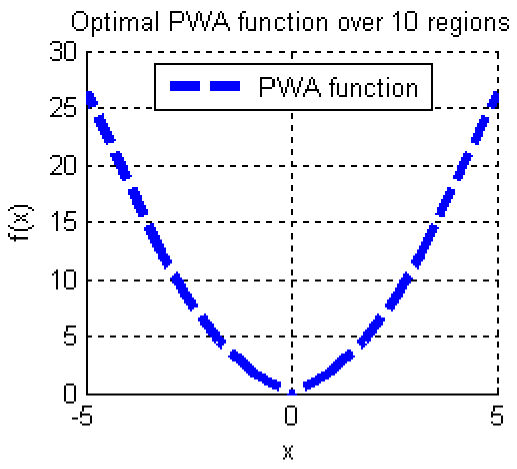


Figure 8.2a: Optimal PWA objective function

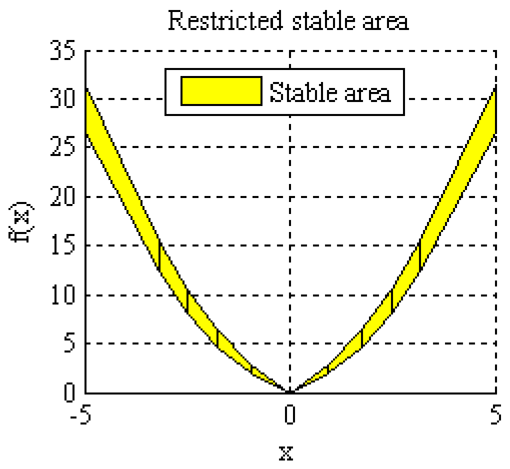


Figure 8.2b: Restricted stable area

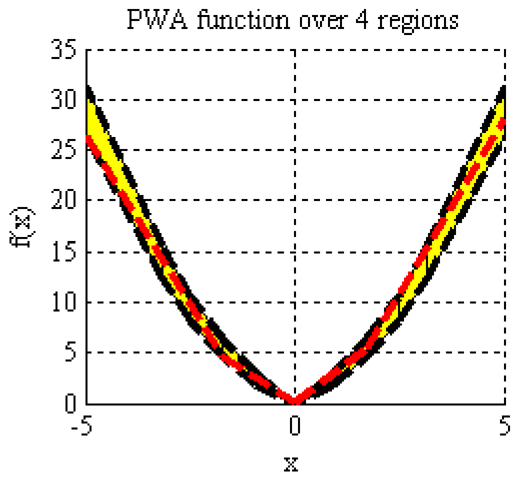


Figure 8.2c: Fitted PWA objective function

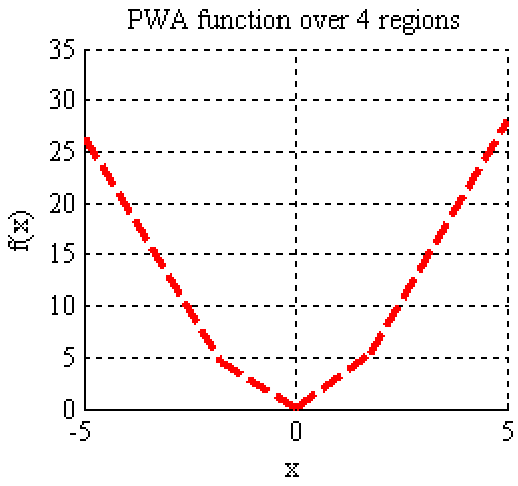


Figure 8.2d: Suboptimal PWA objective function

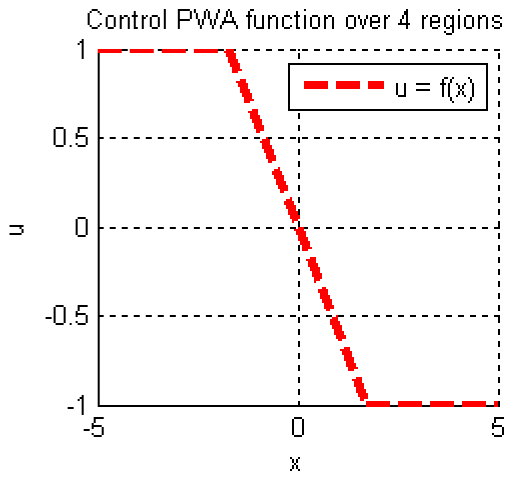


Figure 8.2e: Control PWA function

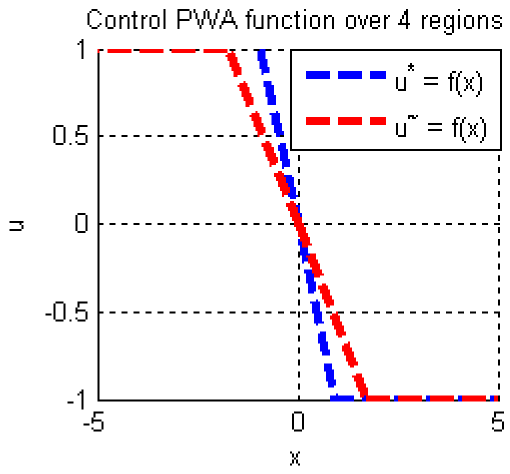


Figure 8.2f: Comparison of the control laws

8.2.1 Formulation in MATLAB

System `lti_1d_unstable` is defined as:

```
clear sysStruct probStruct

sysStruct.A = 1.1;
sysStruct.B = 1;
sysStruct.C = 1;
sysStruct.D = 0;
sysStruct.umax = 1;
sysStruct.umin = -1;
sysStruct.xmax = 5;
sysStruct.xmin = -5;

probStruct.Q = 1;
probStruct.R = 1;
probStruct.norm = 1;
probStruct.N = 5;

nx = mpt_sysStructInfo(sysStruct);
ctrl = mpt_control(sysStruct, probStruct)
```

To create a new suboptimal control law of the system we need to call following functions (function `reindex_pwa` is attached in the appendix):

```
close all, clear all, clc

lti_1d_unstable;
prepare_data;

N = 10; % number of iteration steps
K = 4;  % number of approx regions
x_up = []; x_low = []; % empty extending vertices for boundaries

if nx == 1 % nx is a dimension of the problem
    [Jlow,Jup,V,smtr] = reindex_pwa(Jlow,Jup,V);
    if smtr == 1 && mod(K,2) == 0, K = K/2; V = V(1:end/2);end
end

[alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);

if nx == 1 && smtr == 1 && mod(K,2) == 0
    temp = length(V);
    for i = 1:temp, V{temp+i} = -V{temp-i+1}; end
    alpha = [alpha, -alpha];
    beta = [beta, beta];
end

VV = draw_PWA(alpha, beta, X, Jpoly);
[VU,Vu] = get_u(VV,sysStruct,probStruct);
plot_PWA_u(VV,Vu);
```

8.3 Summarization of examples

Reducing the memory requirements of explicit model predictive control was only a supporting role, since our main goal is to reduce computation time of this control. By using results from each example we can find the improvement of time-consuming calculation of the required optimal input based on current state. In following graphs we will see compared time requirements of optimal (f^*, u^*) and suboptimal $(\tilde{f}, \tilde{u})$ problems.

8.3.1 Summarization of example 1

In this work are mentiond two procedures how we find the input to the system (Chapter 1.4.3 and Chapter 2.2), therefore in order to illustrate computation improvement of the result from the example 1 we will proceed both of them. While in figures 8.3 is compared time required to obtain input to the system in each region, in figure 8.4 we can see the control difference of optimal and suboptimal control laws.

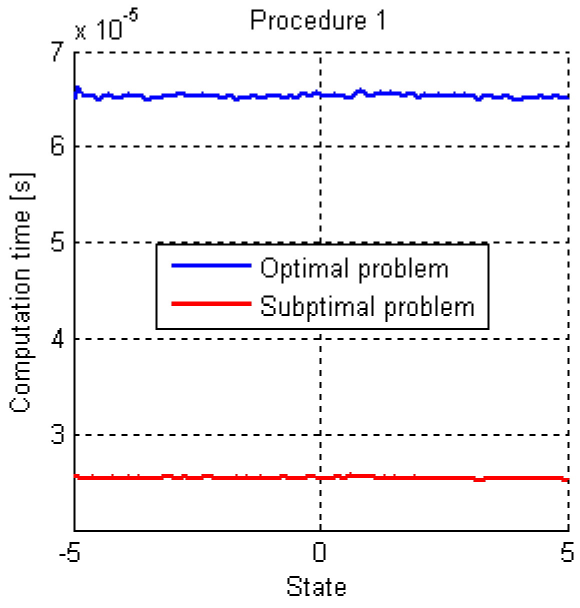


Figure 8.3a: Compared time required to obtain input to the system by procedure from Chapter 1.4.3

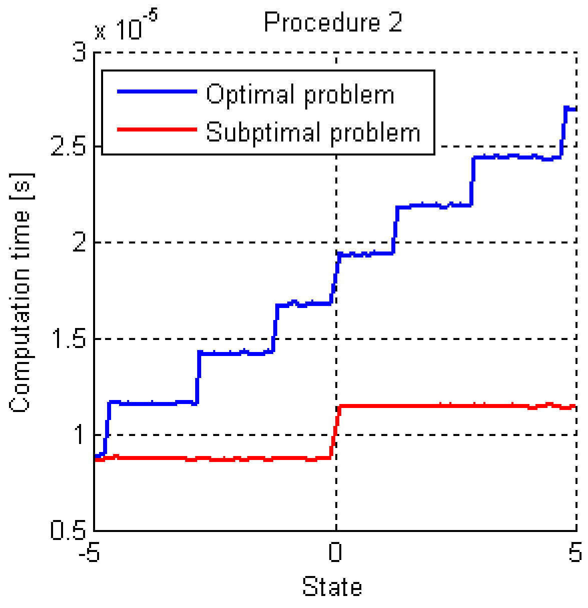


Figure 8.3b: Compared time required to obtain input to the system by procedure from Chapter 2.2

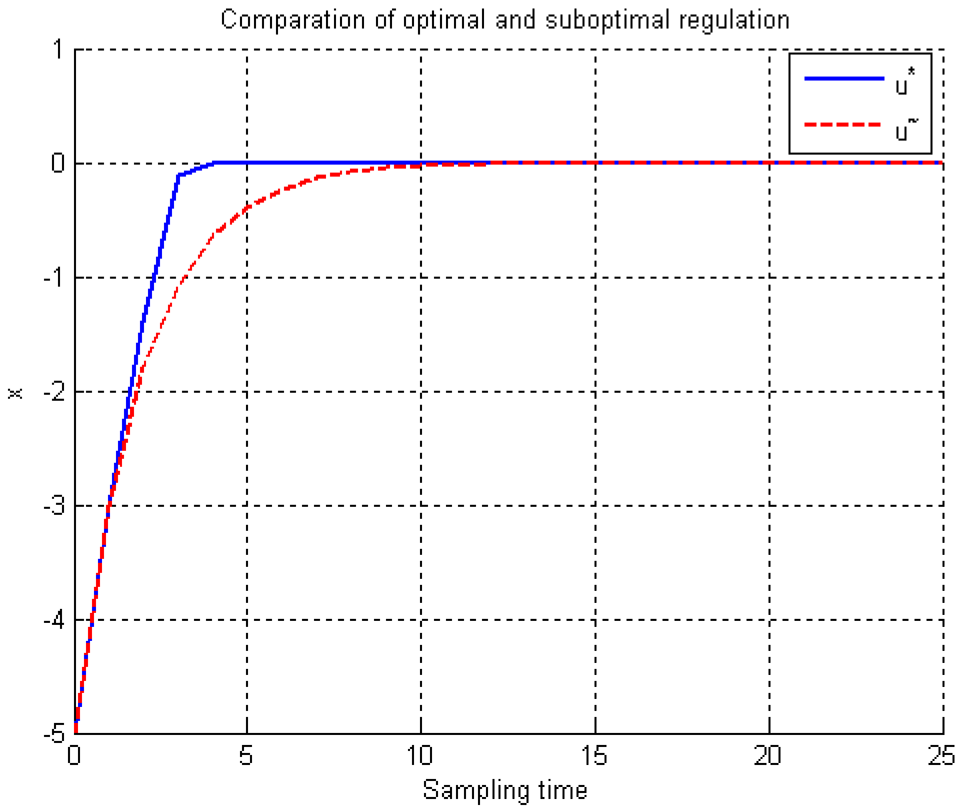


Figure 8.4: Control difference of optimal and suboptimal control laws

8.3.1.1 *Formulation in MATLAB*

Both time comparison procedures can be simple implemented in MATLAB using following commands:

```
close all
clear all
clc

lti_1d_stable;
prepare_data;
N = 10; % number of iteration steps
K = 2;  % number of approx regions
x_up = []; x_low = []; % empty extending vertices

[alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);

[Jlow,Jup,V,smtr] = reindex_pwa(Jlow,Jup,V);
VV = draw_PWA(alpha, beta, X, Jpoly);

[VU,Vu] = get_u(VV,sysStruct,probStruct);
[VUU,Vuu] = get_u(V,sysStruct,probStruct);
```

```
for i = 1:length(alpha)
    J_alpha{i} = alpha(:,i);
    J_beta{i} = beta(:,i);
end

x = [-5:.1:-.1 .1:.1:5];
for j = 1:5
    time_m1 = []; time_m2 = [];
    for i = 1:length(x)
        if x(i) == 0, continue;end
        % Procedure 1
        [time1,u_opt] = get_time_m1(Jlow.C,Jlow.D,V,VUU,x(i));
        [time2,u_sub] = get_time_m1(J_alpha,J_beta,VV,VU,x(i));
        time_m1 = [time_m1 [time1;time2]];

        % Procedure 2
        [time1,u_opt] = get_time_m2(V,VUU,x(i));
        [time2,u_sub] = get_time_m2(VV,VU,x(i));
        time_m2 = [time_m2 [time1;time2]];
    end
end
figure, hold on
title('Procedure 1'),xlabel('State'),ylabel('Computation time [s]')
plot(x,time_m1(1,:),'b')
plot(x,time_m1(2,:),'r')
legend('Optimal problem','Subptimal problem')

figure, hold on
title('Procedure 2'),xlabel('State'),ylabel('Computation time [s]')
plot(x,time_m2(1,:),'b')
plot(x,time_m2(2,:),'r')
legend('Optimal problem','Subptimal problem'),
```

where function `get_time_m1` is defined as:

```
function [time,u] = get_time_m2(VV,VU,x)
tic,
for i = 1:length(VV)
    minVV = min(VV{i}); maxVV = max(VV{i});
    if x <= maxVV && x >= minVV, Ri = i; break; end
end
u = VU{Ri}'*[x;1]; % action input
time = toc;
```

and function `get_time_m2` is defined as:

```
function [time,u] = get_time_m1(alpha,beta,VV,VU,x)
tic,
for i = 1: length(VV)
    Fvalue(i) = x'*alpha{i} + beta{i};
end
MaxF = max(Fvalue);  % maximal function value
Ri = find(Fvalue == MaxF); % index of region
u = VU{Ri}'*[x;1]; % action input
time = toc;
```

Control difference of optimal and suboptimal control laws is implemented in MATLAB by:

```
close all
clear all
clc

lti_1d_stable;
prepare_data;
N = 10; % number of iteration steps
K = 2;  % number of approx regions
x_up = []; x_low = []; % empty extending vertices
[Jlow,Jup,V,smtr] = reindex_pwa(Jlow,Jup,V);

[alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);

VV = draw_PWA(alpha, beta, X, Jpoly);

[VU,Vu] = get_u(VV,sysStruct,probStruct);

time = 25;

% optimal control law
X = [];
x = -5;
for i=1:time
    X = [X x]; x = sysStruct.A*x+sysStruct.B*ctrl(x);
end

% suboptimal control law
Y = [];
x = -5;
for i = j:time
    Y = [Y x];

    % finding u
    for i = 1:length(VV)
        VV_min = min(VV{i});
        VV_max = max(VV{i});
        if (x >= VV_min) && (x <= VV_max)
            u = VU{i}(1)*x + VU{i}(2);
        end
    end

    x = sysStruct.A*x + sysStruct.B*u;
end

figure
hold on
plot(0:time,[X 0],'b','Linewidth',2)
plot(0:time,[Y 0],'--r','Linewidth',2)
axis([0 time -5 1])
title('Comparation of optimal and suboptimal regulation')
xlabel('Sampling time'),ylabel('x'),legend('u^*','u^~'),grid
```

8.3.2 **Summarization of example 2**

In order to summarize results from second example we will use the same functions as in Chapter 8.3.1, with exceptions that `lti_1d_unstable` and `K = 4` has been used.

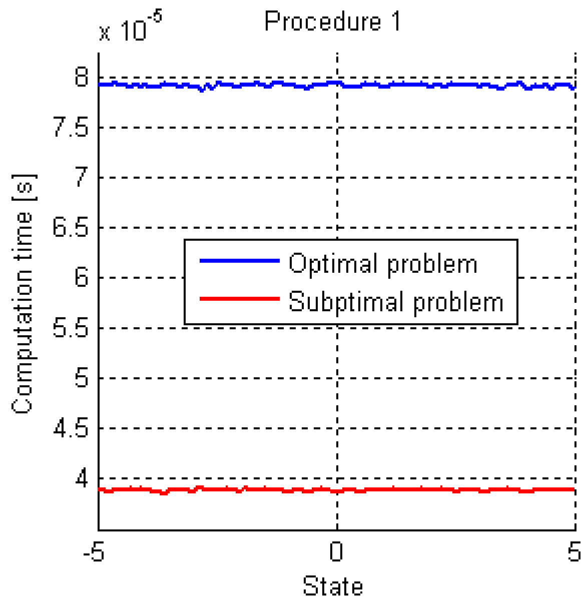


Figure 8.5a: Compared time required to obtain input to the system by procedure from Chapter 1.4.3

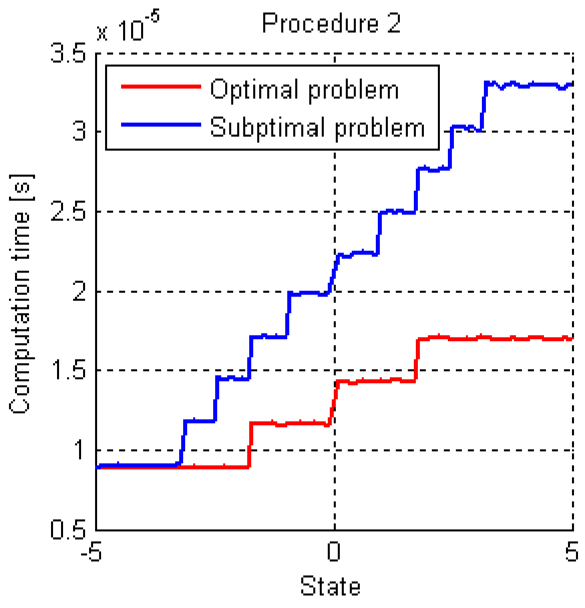


Figure 8.5b: Compared time required to obtain input to the system by procedure from Chapter 2.2

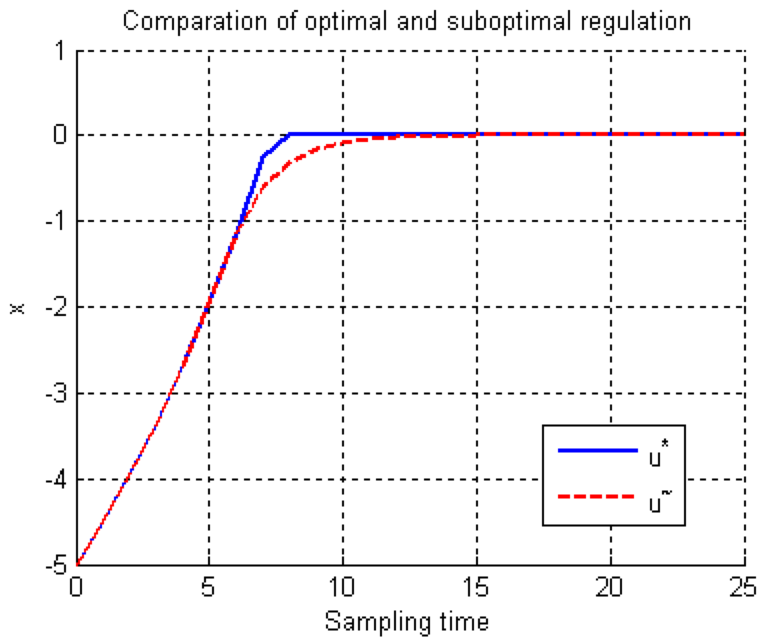


Figure 8.6: Control difference of optimal and suboptimal control laws

Conclusion

Predictive control is one of the most modern control approach which differs from other conventional methods by its ability to predict the development of states and the fact that constraints are directly incorporated into the optimization problem. On the other hand this complexity represents the main drawback of this approach, because to solve such complicated optimization problem requires large computational demands. In this work we have tried to propose a method which allowed implementation of this control approach on systems with fast dynamics.

In the opening Chapters, we have got familiar with model predictive control (its strengths, options, mathematical formulations and so on) as well as the reasons why it is necessary to propose use of the method which is able to reduce the demands for explicit data storage but at the expense of losing performance (optimality). In sequel Chapter we have been searching for an appropriate solution which has been dealing with all necessary requirements of this method and thus to achieve suboptimal regulator, which would simultaneously guarantee the stability of the system in his whole range. In order to define a stable area where a new simplified objective function could be fitted, we have used two boundaries. First (lower) boundary was basically the optimal objective function, while the other one (upper boundary) was created by moving of the system at the limit of stability. In this stable area we were able to approximate a new simplified objective function defined over fewer region. Since only necessary condition has been used in the lower boundary, correct fitting was not guaranteed (as we could saw in the figures). In order to insure that the necessary condition has met the sufficient condition we have defined the certification. Right after the newly created function has fulfilled the certification we was able to extract a control law from it and thus to obtain a suboptimal regulator. In order to proof the functionality of this method we have implemented it into two examples and the results graphically illustrated.

In this work we have proposed a method which is able to reduce the computation time and a storage capacity of the explicit data by a great portion, while stability is still guaranteed. Based on this reduction the control of the system can be faster, but we have to keep in mind that at the expense of losing the performance (optimality).

Resumé

Úvod

Prediktívne riadenie patrí k najmodernejším prístupom riadenia, ktoré sa odlišuje od ostatných konvenčných metód riadenia tým, že sekvencia optimálnych vstupov do systému sa vypočíta prostredníctvom optimalizácie daného problému na konečnom predikčnom horizonte vzhľadom na obmedzenia a za predpokladu, že model procesu, ako aj stav v danom kroku riadenia sú známe. V skutočnosti tento fakt predstavuje aj najväčšiu nevýhodu tohto spôsobu riadenia, pretože pre výpočet takejto optimalizácie je potrebná výkonná výpočtová technika a v neposlednej rade aj efektívny softvér (solver). Z tohto dôvodu sa prediktívne riadenie implementovalo prevažne do procesov s pomalou dynamikou, akými sú napríklad priemyselné procesy, nakoľko optimálny akčný zásah do daného procesu musí byť vypočítaný do doby odobrania ďalšej vzorky (vzorkovacieho času). V posledných rokoch bol však zaznamenaný významný pokrok v oblasti zdokonaľovania a vývoja výpočtových prostriedkov vrátene nových výkonných numerických metód a z toho hľadiska sa aj tento nedostatok postupne vytráca.

Explicitné prediktívne riadenie

Poznáme dva prístupy implementácie prediktívneho riadenia. Prvým z nich je klasický prístup rovnako označovaný ako on-line riadenie. Druhým prístupom je explicitné riadenie, ktoré je taktiež nazývané ako off-line riadenie. Základný rozdiel medzi týmito dvoma spôsobmi riadenia je v prístupe vypočítania optimálneho vstupu do systému. Pri on-line prístupe sa optimálny akčný zásah vypočíta neustálou optimalizáciou problému (1.13), čo vedie k výraznému zaťaženiu výpočtovej techniky. Z tohto dôvodu sa tento klasický prístup riadenia mohol implementovať iba do systémov s pomalou dynamikou. Pri explicitnom riadení sa optimalizácia vykoná iba raz a to ešte predtým, ako začneme riadiť daný systém (odkiaľ vychádza aj názov off-line). Touto optimalizáciou si daný problém vyjadríme ako hybridný systém, ktorého budeme môcť riadiť pomocou rovníc (1.14, 1.17). Teda ako systém, ktorého účelová funkcia ako aj zákon riadenia budú rozdelené do viacerých po častiach afinných úsekov (regiónov). Údaje o každom jednom úseku, teda smernice a posunutie, sú uložené do tabuľky, ktorá bude obsahovať matice (1.15), (1.16), (1.18) a (1.19). Akčný zásah sa získa prostredníctvom určenia indexu aktívneho regiónu pre aktuálny stav a to na základe jednoduchých matematických výpočtov. Následne sa z tabuľky získajú údaje o zákone riadenia pre daný región a určí sa správny akčný zásah. Už teda nebude prebiehať neustala optimalizácia daného problému, čím sa výrazne zníži náročnosť na výpočtovú techniku. Na druhej strane, vzhľadom na rozsiahlosť tabuľky údajov, sa zvýši pamäťová náročnosť.

Zníženie implementačnej náročnosti explicitného prediktívneho riadenia

Z teórie vieme, že pri riadení diskrétnych systémov (ktoré sú všetky systémy riadené počítačom) sa musí čas, potrebný na výpočet optimálneho akčného zásahu, zmestiť práve do jednej doby vzorkovania (2.1), pričom vhodná doba vzorkovania pre daný proces by mala spadať do určitého intervalu (2.2). Explicitné prediktívne riadenie je prístup, pri ktorom sa optimálny vstup do riadeného procesu vypočíta prostredníctvom hľadania príslušného regiónu v tabuľke údajov (obrázok 2.2). Preto môžeme povedať, že čas potrebný na výpočet optimálneho vstupu bude priamo úmerný počtu regiónov (2.5). Preto vieme, že ak budeme chcieť implementovať explicitné prediktívne riadenie do ľubovoľného systému (s určitou periódou vzorkovania), budeme musieť znížiť počet regiónov daného problému až do takej miery, aby bol výpočtový čas akčného zásahu menší (nanajvýš rovný) ako doba vzorkovania. Rovnako prostredníctvom znižovania počtu regiónov budeme znižovať aj pamäťové nároky, ktoré sú potrebné k implementácii daného explicitného prediktívneho riadenia.

Metóda, ktorá bola navrhnutá v tejto práci, sa snaží znížiť počet regiónov a to takým spôsobom, že prvotnú (optimálnu) účelovú funkciu sa pokúsi aproximovať novou (suboptimálnou) účelovou funkciou, ktorá bude zadefinovaná prostredníctvom menšieho počtu regiónov. Avšak takáto aproximácia nemôže byť vykonaná na ľubovoľnom mieste. Preto je potrebné si najskôr zostrojiť hranice stability. Spodnú hranicu tvorí práve naša optimálna účelová funkcia, nakoľko lepšie riadenie ako je to optimálne už neexistuje. Hornú hranicu zostrojíme tak, že daný systém posunieme až na hranicu stability. Prípustná oblasť, kde bude garantovaná stabilita daného systému, vznikne vymedzením práve týchto dvoch hraníc (obrázok 4.3), v ktorej budeme môcť aproximovať novú (suboptimálnu) účelovú funkciu. Pre spätnú kontrolu, či takto zostrojená funkcia bude podmnožinou stabilného priestoru, sme si zostrojili dva certifikáty. Až keď daná funkcia prejde certifikáciou, budeme môcť priradiť k nej suboptimálny zákon riadenia (obrázok 7.2).

Záver

V diplomovej práci sme sa zaoberala problematikou znižovania implementačnej náročnosti explicitného prediktívneho riadenia. Vysvetlili sme si základné prednosti prediktívneho riadenia a následne sme si aj predstavili dva odlišné prístupy pri jeho implementácii. Navrhli sme metódu, ktorá na základe redukcie počtu regiónov dokázala znížiť implementačnú náročnosť explicitného prediktívneho riadenia, ale to za cenu zníženia kvality riadenia (suboptimality). Funkčnosť tejto metódy sme dokázali v poslednej kapitole a to na dvoch príkladoch, pričom výsledky porovnania optimálneho a suboptimálneho riadenia boli ilustrované prostredníctvom grafov.

References

[1] Ing. Adrián Karas,PhD., prof. Ing. Boris Rohaµ-Ilkiv,CSc., doc. Ing. Cyril Belavý,CSc.: Praktické aspekty prediktívneho riadenia. STU Bratislava 2007. ISBN 978-80-89316-06-9

[2] Michal Kvasnica, Ing., PhD. - Model Predictive Control (MPC), Part 1: Introduction

[3] M. Kvasnica, P. Grieder, and M. Baotic´. Multi-Parametric Toolbox (MPT), 2004. Available from http://control.ee.ethz.ch/~mpt/

[4] Alexander Domahidi, Melanie N. Zeilinger, Manfred Morari and Colin N. Jones: Learning a Feasible and Stabilizing Explicit Model Predictive Control.Law by Robust Optimization.

[5] C.N. Jones, M. Barić and M. Morari: Multiparametric Linear Programming with Applications to Control. European Journal of Control (2007)13:152–170

Appendix A – Main program

```
close all
clear all
clc

lti_1d_stable;
% lti_1d_unstable;
% lti_2d;

prepare_data;
N = 10; % number of iteration steps
K = 2;  % number of approx regions
x_up = []; x_low = []; % empty extending vertices

nx = size(sysStruct.A,2);   % number of states

if nx == 1 % nx is a dimension of the problem
    [Jlow,Jup,V,smtr] = reindex_pwa(Jlow,Jup,V);
    if smtr == 1 && mod(K,2) == 0, K = K/2; V = V(1:end/2);end
end

[alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);

if nx == 1 && smtr == 1 && mod(K,2) == 0
    temp = length(V);
    for i = 1:temp
        V{temp+i} = -V{temp-i+1};
    end
    alpha = [alpha, -alpha];
    beta = [beta, beta];
end

% VV = draw_PWA_1D(alpha,beta,Jlow,Jup,V);
VV = draw_PWA(alpha, beta, X, Jpoly);

[VU,Vu] = get_u(VV,sysStruct,probStruct);

plot_PWA_u(VV,Vu);
```

Appendix B – Systems

lti_1d_stable

```
clear sysStruct probStruct

sysStruct.A = 0.8;
sysStruct.B = 1;
sysStruct.C = 1;
sysStruct.D = 0;
sysStruct.umax = 1;
sysStruct.umin = -1;
sysStruct.xmax = 5;
sysStruct.xmin = -5;

probStruct.Q = 1;
probStruct.R = 1;
probStruct.norm = 1;
probStruct.N = 5;

nx = mpt_sysStructInfo(sysStruct);
ctrl = mpt_control(sysStruct, probStruct)
```

lti_1d_unstable

```
clear sysStruct probStruct

sysStruct.A = 1.1;
sysStruct.B = 1;
sysStruct.C = 1;
sysStruct.D = 0;
sysStruct.umax = 1;
sysStruct.umin = -1;
sysStruct.xmax = 5;
sysStruct.xmin = -5;

probStruct.Q = 1;
probStruct.R = 1;
probStruct.norm = 1;
probStruct.N = 5;

nx = mpt_sysStructInfo(sysStruct);
ctrl = mpt_control(sysStruct, probStruct)
```

Appendix C – prepare_data

```
% we want to find a_k, b_k, k = 1,...,K of the convex PWA function
% f(x) := max a_k*x+b_k such that
%   \forall x: Jlow(x) <= max a_k*x+b_k <= Jup(k)
%   "K" is minimized (as a heurstics we minimize ||a_k||_1, ||b_k||_1)
% since Jlow() and Jup() are PWA functions, the first constraint reads
%   \forall i, x \in R_i: Clow_i*x+Dlow_i <= max a_k*x+b_k <= Cup_i*x+Dup_i
%
% in our case, Jlow(x) = V(x), where V(x) is the optimal cost function of a
% given explicit MPC solution (pretend it's a Lyapunov function for now),
% and Jup(x) = Jlow(x) + ||Qx||
%
% Jlow() and Jup() are convex PWA functions:
%   Jlow(x) := Clow_i*x+Dlow_i, if x \in R_i
%    Jup(x) := Cup_i*x+Dup_i, if x \in R_i

% to construct epigraphs, we need to find the maximum of a PWA function
% over its domain. for that, we need vertices of the domain
X = union(ctrl.Pfinal); VX = extreme(X);

% we are going to approximate the cost of a given explicit MPC solution
Clow = ctrl.Bi; for i = 1:length(Clow), Clow{i} = Clow{i}'; end
Dlow = ctrl.Ci;
R = ctrl.Pn;
V = pelemfun(@extreme, R);
nR = length(R);

% get Jup(x) = Jlow(x) + ||Qx||
[Cup, Dup] = J_add_norm(Clow, Dlow, R, ctrl.probStruct.Q);

% obtain polytopic representation of the epigraphs:
% * PJ is the epigraph of Jlow(x)
% * PJN is the epigraph of Jup(x)
% * Jpoly is the polytopic representation of the difference of the two
%   functions
[PJ, PJN, Jpoly, Jmax] = get_J_epigraph(R, X, Clow, Dlow, Cup, Dup);

close all
plot(Jpoly, 'y');

Jlow.R = R;
Jlow.C = Clow;
Jlow.D = Dlow;
Jup.R = R;
Jup.C = Cup;
Jup.D = Dup;
```

J_add_norm

```
function [c, d] = J_add_norm(c, d, R, Q)

nx = dimension(R(1));
V = pelemfun(@extreme, R);

for k = 1:length(c)
    W = V{k}';
    nv = size(W, 2);
    J = c{k}'*W + repmat(d{k}, 1, nv);
    for i = 1:nv
        J(i) = J(i) + sub_norm(W(:, i), Q, R(k));
    end
    q = [W' ones(nv, 1)]\J';
    c{k} = q(1:nx); d{k} = q(end);
end


%--------------------------------------------
function y = sub_norm(x, Q, R)

x0 = chebyball(R);
nx = length(x0);
M = eye(nx);
for i = 1:nx
    if x0(i) < 0
        M(i, i) = -1;
    end
end
y = sum(Q*(M*x));
```

get_J_epigraph

```
function [PJ, PJN, J, Jmax] = get_J_epigraph(R, B, Clow, Dlow, Cup, Dup)
nx = dimension(R(1));

% obtain maximal value of Jup(x) over domain "B"
V = extreme(B);
Jmax = 0;
for i = 1:length(Cup)
    for j = 1:size(V, 1)
        Jmax = max(Jmax, Cup{i}'*V(j, :)' + Dup{i});
    end
end

fprintf('Computing epigraph of Jlow(x)...\n');
% epigraph of the value function, i.e.
%  { [x; e] | e >= Jlow(x) }
x = sdpvar(nx, 1);
e = sdpvar(1, 1);
yPJ = [ ismember(x, B); e <= Jmax ];
for k = 1:length(Clow)
```

```
    yPJ = yPJ + [ e >= Clow{k}'*x + Dlow{k} ];
end
PJ = union(polytope(yPJ));

fprintf('Computing epigraph of Jup(x)...\n');
% epigraph of the shifted value function, i.e.
%  { [x; e] | e >= J(x) + ||x|| }
yPJN = [ ismember(x, B); e <= Jmax ];
for k = 1:length(Cup)
    yPJN = yPJN + [ e >= Cup{k}'*x + Dup{k} ];
end
PJN = union(polytope(yPJN));

fprintf('Computing polytopes of Jup(x)-Jlow(x)...\n');
% polytopic representation of J(x)+||x|| - J(x)
J = polytope;
V = pelemfun(@polytope, R);
nR = length(R);
x = sdpvar(nx, 1);
e = sdpvar(1, 1);
for i = 1:nR
    if i==1 || i == nR || mod(i, 10)==0
        fprintf('%d / %d\n', i, nR);
    end
    Jlow = Clow{i}'*x+Dlow{i};
    Jup = Cup{i}'*x+Dup{i};
    yJ = [ ismember(x, R(i)); Jlow <= e <= Jup ];
    J = [J polytope(yJ)];
end
```

Appendix D – reindex_pwa

```
% re-index each region from left to right by axis x1
% check for symmetry (if smtr == 1 => obj function is symmetrical)
function [Jlow,Jup,V,smtr] = reindex_pwa(Jlow,Jup,V)
% find correct positions of each region
temp = zeros(1,length(Jlow.R));
positions = zeros(1,length(Jlow.R));

if length(Jlow.C{:,1}) == 1
    for i = 1:length(Jlow.R)
        temp(i) = max(extreme(Jlow.R(i)));
    end
else
    for i = 1:length(Jlow.R)
        tempp = extreme(Jlow.R(i));
        temp(i) = max(tempp(1,:));
    end
end

temp2 = sort(temp); k = 1;
while k <= length(temp2)
    tempp2 = find(temp == temp2(k));
    if length(tempp2) > 1
        for j = 1:length(tempp2)
            positions(k+j-1) = tempp2(j);
        end
        k = k + length(tempp2);
    else
        positions(k) = tempp2; k = k + 1;
    end
end

% re-index each region
Jlow.R = Jlow.R(positions);
Jlow.C = Jlow.C(positions);
Jlow.D = Jlow.D(positions);

Jup.R = Jup.R(positions);
Jup.C = Jup.C(positions);
Jup.D = Jup.D(positions);

V = V(positions);
%% checking for symmetry (if smtr == 1 => obj is symmetrical)
k = length(Jup.R);
smtr = 1;

for i = 1:k/2
    if sum(abs(Jup.C{i} + Jup.C{k+1-i}) > 1e-9) > 0, smtr = 0; break; end
    if sum(abs(Jup.D{i} - Jup.D{k+1-i}) > 1e-9) > 0, smtr = 0; break; end
    if sum(abs(Jlow.C{i} + Jlow.C{k+1-i}) > 1e-9) > 0, smtr = 0; break; end
    if sum(abs(Jlow.D{i} - Jlow.D{k+1-i}) > 1e-9) > 0, smtr = 0; break; end
end
```

Appendix E – Jlow_Jup

```
function [alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N)
[alpha,beta] = fit_PWA(Jup,Jlow,V,K,x_up,x_low);

% cert_Jup: f(x) <= Jup(x)
[x] = cert_Jup(alpha,beta,Jup,V);
if x ~= 0
    x_up = [x_up; x];
    [alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);
end

% cert_Jlow: Jlow(x) <= f(x)
[x,d] = cert_Jlow(alpha,beta,Jlow,V);
disp('======================================')
alpha
beta
x_up
x_low
disp('======================================')

if d < -1e-6
    x_low = [x_low; x];
    if length(x_low) < N + 1
        [alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);
    elseif length(x_low) == N
        fprintf('!!! Lack of iteration steps !!!\n')
    end
else
    fprintf('Jlow(x) <= f(x) <= Jup(x) certified\n')
end
```

Appendix F – fit_PWA

```
function [alpha,beta] = fit_PWA(Jup,Jlow,V,K,x_up,x_low)
nx = size(V{1},2);

% obj. function
obj = 0;

% constraints
F = [];

% symbolic values
alpha = sdpvar(nx,K,'full');
beta = sdpvar(1,K);

for i = 1:length(V)
    % Jup
    for j = 1:length(V{i})
        F = F + [(V{i}(j,:)*alpha + beta) <= ...
            V{i}(j,:)*Jup.C{i} + Jup.D{i}];
    end
    for j = 1:length(x_up)
        F = F + [(x_up(j,:)'*alpha + beta) <= ...
            x_up(j,:)'*Jup.C{i} + Jup.D{i}];
    end
    % Jlow
    for j = 1:length(V{i})
        F = F + [max(V{i}(j,:)*alpha + beta) >= ...
            V{i}(j,:)*Jlow.C{i} + Jlow.D{i}];
    end
    for j = 1:length(x_low)
        F = F + [max(x_low(j,:)'*alpha + beta) >= ...
            x_low(j,:)'*Jlow.C{i} + Jlow.D{i}];
    end
end

%% solve
info = solvesdp(F,obj);
if info.problem ~= 0
    error('Problem is unsolvable !!!')
end
alpha = double(alpha);
beta = double(beta);
```

Appendix G – Certifications

cert_Jup

```
function [x] = cert_Jup(alpha,beta,Jup,V)
nx = size(alpha,1);

for i = 1:length(V)
    x = sdpvar(nx,1);

    obj = 0; F = [];
    F = F + [ismember(x, Jup.R(i))];
    F = F + [max(alpha'*x + beta') - (Jup.C{i}'*x + Jup.D{i}) > 1e-5];

    info = solvesdp(F,obj);

    x = double(x);
    if info.problem == 0 && x ~= 0 , break;end
end

if x == 0, disp('cert_Jup: Alpha and beta were certificated !!!')
else disp('cert_Jup: Alpha and beta were not certificated !!!')
end
```

cert_Jlow

```
function [x,d] = cert_Jlow(alpha,beta,Jlow,V)
nx = size(alpha,1);
for k = 1:length(V)
    x = sdpvar(nx,1); eps = sdpvar(1,1);

    obj = eps - (Jlow.C{k}'*x + Jlow.D{k});

    F = [];
    F = F + [ismember(x, Jlow.R(k))];
    for i = 1:length(alpha)
        F = F + [eps >= alpha(:,i)'*x + beta(i)];
    end

    info = solvesdp(F,obj);
    if info.problem ~= 0, error('Problem is unsolvable !!!'),end

    x = double(x); d = double(obj);
    if d < -1e-6, break; end
end

if d >= -1e-6, disp('cert_Jlow: Alpha and beta were certificated !!!')
else
    disp('cert_Jlow: Alpha and beta were not certificated !!!')
    fprintf('           In point x = %d is difference = %d. \n',x,d)
end
```

Appendix H – draw_PWA

```
% function [VV] = draw_PWA(alpha, beta, X, Jpoly)
% alpha - slope (f(x) = alpha*x + beta)
% beta  - affine term (f(x) = alpha*x + beta)
% X     - Polytop representing range of each axis (in matrix X)
% VV    - vertices of each new region (from new alpha and beta)
function [VV] = draw_PWA(alpha, beta, X, Jpoly)

% obtain the explicit representation of f(x) as
%   f(x) = c_j*x+d_j if x \in P_j
f = get_explicit_pwa_max(alpha, beta, X);

if gcf==1, plot(Jpoly, 'y'); end

hold on, plot_pwa(f); grid on, hold off

% output
for i = 1:length(f.R), VV{i} = extreme(f.R(i)); end
```

get_pwa_max

```
function fs = get_pwa_max(a, b, X)

nx = dimension(X);
J = sdpvar(1, 1);
x = sdpvar(nx, 1);
F = [ J <= 1e4; ismember(x, X) ];
for k = 1:size(a, 2)
    F = F + [ J >= a(:, k)'*x + b(k) ];
end
sol = solvemp(F, J, sdpsettings, x, J);
fpwa = mpt_mpsol2ctrl(sol, 1);
R = fpwa.Pn;
C = fpwa.Bi; for i = 1:length(C), C{i} = C{i}'; end
D = fpwa.Ci;

fs.R = R;
fs.C = C;
fs.D = D;
```

Appendix I – get_u

```
% VU -> cell of the alpha and beta for inputs
%      (VU{i} = [a1_i, a2_i...;b1_i, b2_i...;])
% Vu -> cell of the optimum inputs in all region vertices  from VV
function [VU,Vu] = get_u(VV,sysStruct,probStruct)
[Vu] = get_Vu(VV,sysStruct,probStruct);

nx = size(sysStruct.A,2);     % number of states
nu = size(sysStruct.B,2);     % number of inputs

for i = 1:length(VV)
    for j = 1:nx
        X = [VV{i}(:,j)'; ones(1,length(VV{i}(:,j)))];
        U = [Vu{i}(:,j)'];
        C = U*X^-1;
        VU{i}(:,j) = C';
    end
end
```

get_Vu

```
% Vu -> cell of the optimum inputs in all vertices
function [Vu] = get_Vu(VV,sysStruct,probStruct)
% objective function and constraints
if probStruct.norm == inf
    [Vu] = norminf(VV,sysStruct,probStruct);
elseif probStruct.norm == 1
    [Vu] = norm1(VV,sysStruct,probStruct);
else
    fprintf('!!! Norm "%d" is not allowed !!!\n',probStruct.norm);
end
```

norminf

```
% min  sum(Ex) + sum(Eu)
% s.t.    X == tA*X+tB*U+tE*xt
%      tH*X <= tK
%      tL*U <= tM
%      -tEx <= tQ*X <= tEx
%      -tEu <= tR*U <= tEu
function [Vu] = norminf(VV,sysStruct,probStruct)
% number of states, number of inputs
nx = size(sysStruct.A,2);   % number of states
nu = size(sysStruct.B,2);   % number of inputs
N = probStruct.N;           % prediction horizon

% matrices for the objective function and the constraints for N = 1
H = [eye(nx);-eye(nx)];
L = [eye(nu);-eye(nu)];
K = [sysStruct.xmax; -sysStruct.xmin];
M = [sysStruct.umax ; -sysStruct.umin];
```

```
% matrices for the objective function and the constraints for N
tQ = kron(eye(N),probStruct.Q);
tR = kron(eye(N),probStruct.R);
tH = kron(eye(N),H);
tL = kron(eye(N),L);
tK = kron(ones(N,1),K);
tM = kron(ones(N,1),M);
tE1 = eye(nx);
tE2 = kron(ones(N-1,1),zeros(nx));
tE = [tE1;tE2];
tA1 = kron(ones(1,N),zeros(nx));
tA2 = kron(eye(N-1),sysStruct.A);
tA3 = kron(ones(N-1,1),zeros(nx));
tA = [tA1;tA2 tA3];
tB1 = kron(ones(1,N),zeros(nx,nu));
tB2 = kron(eye(N-1),sysStruct.B);
tB3 = kron(ones(N-1,1),zeros(nx,nu));
tB = [tB1;tB2 tB3];

% optimization variables
U = sdpvar(N*nu,1);
X = sdpvar(N*nx,1);
Eu = sdpvar(N,1);
Ex = sdpvar(N,1);
tEx = kron(Ex, ones(nx,1));
tEu = kron(Eu, ones(nu,1));

% simulation
for i = 1:length(VV)
    u0opt = [];
    for j = 1:2
        % objective function and constraints
        obj = sum(Ex) + sum(Eu);
        F = [X == tA*X+tB*U+tE*VV{i}(j,:); ...
            tH*X <= tK; ...
            tL*U <= tM; ...
            -tEx <= tQ*X <= tEx; ...
            -tEu <= tR*U <= tEu];
        info = solvesdp(F,obj);
        if info.problem ~= 0, error('Problem is infeasible'), end
        u0opt = [u0opt double(U(1:nu))];
    end
    Vu{i} = u0opt';
end
```

norm1

```
% min  c'*Z
% s.t. AA*Z <= BB
%      GG*Z = HH
function [Vu] = norm1(VV,sysStruct,probStruct)
% number of states, number of inputs
nx = size(sysStruct.A,2);
nu = size(sysStruct.B,2);
N = probStruct.N;
```

```
% matrices for the objective function and the constraints for N = 1
H = [eye(nx);-eye(nx)];
L = [eye(nu);-eye(nu)];
K = [sysStruct.xmax; -sysStruct.xmin];
M = [sysStruct.umax ; -sysStruct.umin];

% matrices for the objective function and the constraints for N
tQ = kron(eye(N),probStruct.Q);
tR = kron(eye(N),probStruct.R);
tH = kron(eye(N),H);
tL = kron(eye(N),L);
tK = kron(ones(N,1),K);
tM = kron(ones(N,1),M);
tE1 = eye(nx);
tE2 = kron(ones(N-1,1),zeros(nx));
tE = [tE1;tE2];
tA1 = kron(ones(1,N),zeros(nx));
tA2 = kron(eye(N-1),sysStruct.A);
tA3 = kron(ones(N-1,1),zeros(nx));
tA = [tA1;tA2 tA3];
tB1 = kron(ones(1,N),zeros(nx,nu));
tB2 = kron(eye(N-1),sysStruct.B);
tB3 = kron(ones(N-1,1),zeros(nx,nu));
tB = [tB1;tB2 tB3];

% optimization variables
U = sdpvar(N*nu,1);
X = sdpvar(N*nx,1);
Eu = sdpvar(N*nu,1);
Ex = sdpvar(N*nx,1);

% objective and constraints
obj = 0; F = [];

ZnuH = zeros(size(tH,1),N*nu);
ZnxH = zeros(size(tH,1),N*nx);
ZnxL = zeros(size(tL,1),N*nx);
ZnuQ = zeros(size(tQ,1),N*nu);
ZnuL = zeros(size(tL,1),N*nu);
ZnxR = zeros(size(tR,1),N*nx);

AA = [ZnuH tH ZnuH ZnxH; ...
    tL ZnxL ZnuL ZnxL; ...
    ZnuQ tQ ZnuQ -eye(N*nx); ...
    ZnuQ -tQ ZnuQ -eye(N*nx); ...
    tR ZnxR -eye(N*nu) ZnxR; ...
    -tR ZnxR -eye(N*nu) ZnxR];

BB = [tK;tM;zeros(size(AA,1) - size([tK;tM],1),1)];
GG = [-tB (eye(N*nx)-tA) zeros(N*nx,N*nu) zeros(N*nx)];
% HH = tE*xt;
%
% F = [AA*[U;X;Eu;Ex] <= BB; ...
%     GG*[U;X;Eu;Ex] == HH];
% simulation
for i = 1:length(VV)
    u0opt = [];
    for j = 1:2
        HH = tE*VV{i}(j,:);

        F = [AA*[U;X;Eu;Ex] <= BB; ...
            GG*[U;X;Eu;Ex] == HH];

        info = solvesdp(F,obj);
        if info.problem ~= 0, error('Problem is Infeasible'),end
        u0opt = [u0opt double(U(1:nu))];
    end
    Vu{i} = u0opt';
end
```

Appendix J – plot_PWA_u

```
function plot_PWA_u(VV,Vu)
nx = size(VV{1},2);

if nx == 1
    figure
    hold on

    plot([VV{:}],[Vu{:}],'--b','LineWidth',3)

    xlabel('x'), ylabel('u'), legend('u = f(x)'), grid
    title(sprintf('Control PWA function over %d regions',length(VV)))

elseif nx == 2
    temp = ceil(length(VV)/7);
    color = repmat({'b','g','r','c','m','y','k'},1,temp);

    figure
    grid, hold on
    xlabel('x1'), ylabel('x2'), zlabel('u')
    title(sprintf('Control PWA function over %d regions',length(alpha)))

    % for u1
    for i = 1:length(VV)
        x = VV{i}(:,1);
        y = VV{i}(:,2);
        z = Vu{i}(:,1);

        patch(x,y,z,color{i})
    end

    if size(Vu{1},2) == 2
        figure
        grid, hold on
        xlabel('x1'), ylabel('x2'), zlabel('u2')
        title(sprintf('Control PWA function over %d regions',length(alpha)))

        % for u2
        for i = 1:length(VV)
            x = VV{i}(:,1);
            y = VV{i}(:,2);
            z = Vu{i}(:,2);

            patch(x,y,z,color{i})
        end
    end

else
    fprintf('!!! Cant plot if nx > 2 !!!\nNumber of states: nx = %d\n',nx)
end
```

Appendix K – get_PWA

```
close all
clear all
clc

%%
x = [-5 -4 -3 -2 -1 0 1 2 3 4 5];
y = [15 11 8 5 2.2 0 2.2 5 8 11 15];

% x = [-5 -4 -3 -2 -1 0 1 2 3 4 5; ...
%     -5 -4 -3 -2 -1 0 1 2 3 4 5];
% y = [15 11 8 5 2.2 0 2.2 5 8 11 15];

[alpha,beta] = Points_into_PWA(x,y);
%%
nx = size(x,1);

% if nx == 1, plot_1D_PWA(alpha, beta, x,y),end

x_max = zeros(1,nx);
x_min = zeros(1,nx);
for i = 1:size(x,1)
    x_max(i) = max(x(i,:));
    x_min(i) = min(x(i,:));
end
X = polytope([eye(nx);-eye(nx)],[x_max'; -x_min']);

% obtain the explicit representation of f(x) as
%   f(x) = c_j*x+d_j if x \in P_j
f = get_explicit_pwa_max(alpha', beta', X);

plot_pwa(f);
grid on
hold off
```

Appendix L – Points_into_PWA

```
function [alfa,beta] = Points_into_PWA(x,y)
x_length = length(x);
y_length = length(y);
if x_length ~= y_length error('! Length x ~= y !'),end
%% symbolic parameters
xlength = x_length - 1;
nx = size(x,1);

J = sdpvar(xlength,1);
alfa = sdpvar(xlength,nx);
beta = sdpvar(xlength,1);
%% objective function
obj = 0;
for i = 1:xlength
    obj = obj + (J(i) - y(i))^2;
end
%% constraints
F = [];
for i = 1:xlength
    for j = i+1:xlength + 1
        F = F + [alfa(i,:) <= (y(j)-y(i))/(x(:,j)-x(:,i))];
    end
    if nx == 1 % symmetrical obj
        F = F + [alfa(i,:) == -alfa(xlength + 1 - i,:)];
    end
end

for i = 1:xlength
    F = F + [J(i) == alfa(i,:)*x(:,i) + beta(i)];
    if nx == 1 % symmetrical obj
        F = F + [beta(i) == beta(xlength + 1 - i)];
    end
end
%% solve
info = solvesdp(F,obj);
if info.problem ~= 0
    error('Problem is unsolvable !!!')
end
alfa = double(alfa);
beta = double(beta);
```

Appendix M – Commands and functions used in sumarization

Comparison of time requirements

```
close all
clear all
clc

lti_1d_stable;
prepare_data;
N = 10; % number of iteration steps
K = 2;  % number of approx regions
x_up = []; x_low = []; % empty extending vertices

[alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);
[Jlow,Jup,V,smtr] = reindex_pwa(Jlow,Jup,V);

VV = draw_PWA(alpha, beta, X, Jpoly);

[VU,Vu] = get_u(VV,sysStruct,probStruct);
[VUU,Vuu] = get_u(V,sysStruct,probStruct);

for i = 1:length(alpha)
    J_alpha{i} = alpha(:,i); J_beta{i} = beta(:,i);
end

x = [-5:.1:-.1 .1:.1:5];
for j = 1:5
    time_m1 = [];
    time_m2 = [];
    for i = 1:length(x)
        if x(i) == 0, continue;end
        % Procedure 1
        [time1,u_opt] = get_time_m1(Jlow.C,Jlow.D,V,VUU,x(i));
        [time2,u_sub] = get_time_m1(J_alpha,J_beta,VV,VU,x(i));
        time_m1 = [time_m1 [time1;time2]];

        % Procedure 2
        [time1,u_opt] = get_time_m2(V,VUU,x(i));
        [time2,u_sub] = get_time_m2(VV,VU,x(i));
        time_m2 = [time_m2 [time1;time2]];
    end
end
figure, hold on
title('Procedure 1'),xlabel('State'),ylabel('Computation time [s]')
plot(x,time_m1(1,:),'b'), plot(x,time_m1(2,:),'r')
legend('Optimal problem','Subptimal problem')

figure, hold on
title('Procedure 2'),xlabel('State'),ylabel('Computation time [s]')
plot(x,time_m2(1,:),'b'),plot(x,time_m2(2,:),'r')
legend('Optimal problem','Subptimal problem')
```

get_time_m1

```
function [time,u] = get_time_m1(alpha,beta,VV,VU,x)
tic,
for i = 1: length(VV)
    Fvalue(i) = x'*alpha{i} + beta{i};
end
MaxF = max(Fvalue);  % maximal function value
Ri = find(Fvalue == MaxF); % index of region
u = VU{Ri}'*[x;1]; % action input
time = toc;
```

get_time_m2

```
function [time,u] = get_time_m2(VV,VU,x)
tic,
for i = 1:length(VV)
    minVV = min(VV{i}); maxVV = max(VV{i});
    if x <= maxVV && x >= minVV, Ri = i; break; end
end
u = VU{Ri}'*[x;1]; % action input
time = toc;
close all
clear all
clc
```

Comparition of control laws

```
lti_1d_stable;
prepare_data;
N = 10; % number of iteration steps
K = 2;  % number of approx regions
x_up = []; x_low = []; % empty extending vertices
[Jlow,Jup,V,smtr] = reindex_pwa(Jlow,Jup,V);

[alpha,beta,x_up,x_low] = Jlow_Jup(Jup,Jlow,V,K,x_up,x_low,N);

VV = draw_PWA(alpha, beta, X, Jpoly);

[VU,Vu] = get_u(VV,sysStruct,probStruct);

time = 25;

% optimal control law
X = [];
x = -5;
for i=1:time
    X = [X x];
    x = sysStruct.A*x+sysStruct.B*ctrl(x);
End

% suboptimal control law
Y = [];
x = -5;
```

```
for j = 1:time
    Y = [Y x];

    % finding u
    for i = 1:length(VV)
        VV_min = min(VV{i});
        VV_max = max(VV{i});
        if (x >= VV_min) && (x <= VV_max)
            u = VU{i}(1)*x + VU{i}(2);
        end
    end

    x = sysStruct.A*x + sysStruct.B*u;
end

figure
hold on
plot(0:time,[X 0],'b','Linewidth',2)
plot(0:time,[Y 0],'--r','Linewidth',2)
axis([0 time -5 1])
title('Comparation of optimal and suboptimal regulation')
xlabel('Sampling time'),ylabel('x'),legend('u^*','u^~'),grid
```