INSTITUTES for MATHEMATICS

(Graz University of Technology)

&

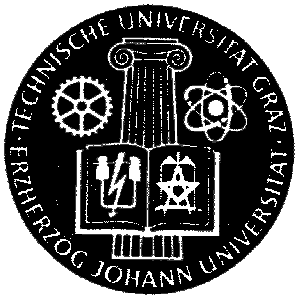
INSTITUTE for MATHEMATICS and SCIENTIFIC COMPUTING

(University of Graz)

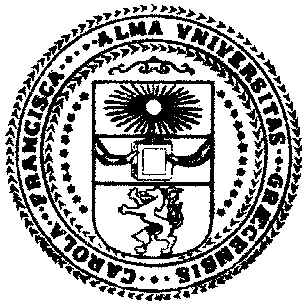
M.E.C. Mutsaers

Receding Horizon Control Techniques applied to the Cardiovascular System

Report No. 02/2007 June, 28, 2007



Institute for Mathematics,
Graz University of Technology
Steyrergasse 30
A-8010 Graz, Austria



Institute for Mathematics and
Scientific Computing,
University of Graz
Heinrichstrasse 36
A-8010 Graz, Austria

Receding horizon control techniques applied to the cardiovascular system

Traineeship report
University of Graz
Austria

M.E.C. Mutsaers

June 28, 2007

CONTENTS

1 Introduction **5**

2 Model for the cardiovascular system **7**
2.1 General formulas needed for the model . . . 7
2.2 Deriving a model for the system . . . 9
2.2.1 States related with volume changes . . . 9
2.2.2 States related with ventricle activity . . . 10
2.2.3 State related with the amount of oxygen supply to the tissues . . . 10
2.2.4 State related with the baroreceptor loop . . . 11
2.2.5 Complete model . . . 11
2.3 Simulating the model in Matlab . . . 12

3 The receding horizon controller **13**
3.1 Introduction . . . 13
3.2 General optimisation problem . . . 13
3.3 Optimisation problem for the cardiovascular model . . . 15
3.4 Gradient with respect to the state equations . . . 15
3.4.1 Calculating the gradient by hand . . . 16
3.4.2 Automatic differentiation . . . 21
3.5 Gradient with respect to the input . . . 21
3.6 Updating the model input function . . . 22
3.7 Summary of the complete algorithm . . . 26
3.8 Note on limits in integrals . . . 27
3.9 Notes on implementation in Matlab . . . 27

4 Receding horizon in closed loop situation **29**
4.1 Comparison with loop in real system . . . 29
4.2 Simulation results of closed loop system . . . 30
4.2.1 Parameters in cost function . . . 30
4.2.2 Control and horizon time intervals . . . 35
4.2.3 Optimisation results in controller . . . 37
4.3 Conclusions on simulation results . . . 38

5 Noise sensitivity of closed loop system **39**
5.1 Noise on controller output and system output . . . 39

6 Parameter estimation **43**
6.1 Defining the problem for the parameter estimation . . . 43
6.2 Steps taken during estimation . . . 44
6.2.1 Estimating a proper starting value for controller output-related parameter . . . 44
6.2.2 Estimate a value for the control horizon . . . 45
6.2.3 Estimate a value for the gain of the horizon inside the controller . . . 45

6.2.4 Estimating the values for the other parameters 47
6.2.5 Estimating parameters in controller and model 50

7 Summary, conclusions and recommendations **53**

A Parameters used in the model **55**

B Matlab files **57**

Bibliography **75**

CHAPTER 1

Introduction

This report contains the results of my traineeship project at the Karl-Franzens University of Graz. At this university in Austria, I have been participating in a research project at the department of Mathematics and Scientific Computing, which is an institute of the faculty Natural Science. In this relative small group, a lot of research is done on subjects like (functional) Analysis, Algebra, Numerical Mathematics and Optimisation. Also the research of the members contains a lot of plant-modeling and control. My supervisor in Graz is Stefan Volkwein, who is a member of the Optimal Control and Inverse Problems research group of the department. He knows a lot about optimisation and numerical mathematics, which are mathematical aspects I definitely needed during my traineeship period.

Because I have mentioned to my coach in the Netherlands (Siep Weiland) that I am interested in model- and control problems with a biomedical aspect, Mr. Volkwein introduced me into a topic which matches those interests. In the group, research has been done on modeling of the cardiovascular system of human beings. This cardiovascular system contains all blood vessels and the complete heart-activity of a human. Because not everything what happens inside the body is exactly known and defined, it is a hard task to create a model for it. Prof. Kappel, which is the head of the Mathematical department, Jerry Batzel, which is a member of the group, Prof. Daniel Schneditz, a professor at the medical university, and Prof. Hien Tran, a mathematics professor at the university of the North Carolina State University, have written a book named "Cardiovascular & Respiratory Systems: Modeling, Analysis & Control" [1], which was the starting point of my project. As the title of this book already mentions, it contains a lot of fundamental aspects of the cardiovascular system that I needed during the project.

The first chapter of the book [1] contains a model for the cardiovascular system, which is summarized in Chapter 2 of this report. In this model, the so called baro-receptor loop is stabilized using an LQR-controller. This baroreceptor loop controls the contraction rate of the heart, which has a very large influence on the complete model. This controller depends on the arterial systemic pressure in the body, which is also the case in real life! The baroreceptors, which are endings of nerves, measure the pressure and transmit this information to the medulla. Then, after some processing, the medulla controls the contraction-rate of the heart. So, also the controller used in the model should have this functionality. For creating an LQR-controller, linearization of the model is needed but this mathematical model transformation is not desired, because the non-linear aspects of the model disappear!

That is why a so called receding horizon optimal controller is introduced in this report. Chapter 3 starts with the definition of the optimisation problem that needs to be solved and needs to be implemented into a regulator. This regulator is a kind of predictive controller, which has a fixed time interval in future. That is why it is called a receding horizon controller!

First, this regulator is estimated for open-loop purposes, but in Chapter 4 also the closed loop variant is investigated. Because no linearization has been applied, those controllers should give a better performance than the LQR controller defined in the book.

In Chapter 5 noise will be added to the input and the output of the cardiovascular system. Those noise additions should not make the system behave unstable.

Because the closed loop system, using the model and the receding horizon controller, should have the same behaviour as the process in the human body, the parameters for the controller and model are

estimated using clinical measurement data. This is discussed in Chapter 6 of this report.

In the last Chapter 7, the results of the research I have done will be summarized and conclusions are drawn. Also recommendations for further research on this topic are made in this chapter.

The results of this research are not only published in this traineeshop report, but also in a paper [5]. This paper is submitted for a journal, but is already available on the SFB website of the Mathematical department of the University of Graz.

At the end of this introduction, I want to thank some people who have helped me during this traineeship project. First of all, I want to thank Stefan Volkwein for being my supervisor during the project in Graz. Without his help and coaching, I definitely wasn't able to get the results that are now written down in this report. I also want to thank Prof. Franz Kappel, Jerry Batzel and Mostafa Bachar from the Mathematical department for helping me with problems and by giving me very useful information (during e.g. lectures I have participated) which was needed to understand everything. At last, I also want to thank my coach in the Netherlands, Siep Weiland, who arranged the possibility for me to do this nice traineeship project in Graz.

CHAPTER 2

Model for the cardiovascular system

As mentioned in the introduction, this chapter contains the main formulas and aspects for the considered cardiovascular model. This model, which is discussed in [1], is used as starting point for the complete traineeship.
This chapter starts with some general formulas that describe the flows of the blood, which are used in the derivation of the model. With those formulas, the (differential) equations for the system can be described, which results in the complete model.

2.1 General formulas needed for the model

First of all, a relation between the pressure and the volume in a blood-containing compartment has to be defined. A linear relation is assumed:

$$V = cP + V_0. \tag{2.1}$$

In (2.1), c is the compliance and V is the volume of the compartment. The pressure in the compartment is denoted with P, and the unstressed volume of the compartment is denoted with V_0 (which is often equal to zero later on this report).

One of the most important things to know in the cardiovascular system is the cardiac output Q_{co}. This calculation is just a multiplication of the stroke volume V_{str} (the volume of the blood ejected by one beat of the ventricle) with the heart-rate H:

$$Q_{co} = HV_{str}. \tag{2.2}$$

Knowing the cardiac output and the volume-pressure relation is not enough for modeling the complete system. In our model, all vessels in the systemic- and pulmonary circuits are lumped together into two vessels. So, these vessels all have one equivalent compliance- (c) and resistance (R) value. The compliances are being used in the volume-pressure relation (2.1), but also those resistances of the system-vessels play a large role in the model, because they influence the blood flow F:

$$F = \frac{1}{R}\left(P_a - P_v\right). \tag{2.3}$$

The indices of the pressures in equation (2.3) provide information about what kind of vessel is mentioned. Table 2.1 gives an overview of the meanings of those two indices and the other subscripts that are used in this report. Relation (2.3) can be compared with Ohm's law. An equivalent electrical schematic can be described with a resistor, which has two different potentials on each site (a and v). The voltages on the nodes ($\mathcal{V}_a$ & $\mathcal{V}_v$) are equal to the pressures (P_a & P_v), the resistance in the vessels is equal to an electrical resistance $\mathcal{R}$ and the flow (F) is equal to the current ($\mathcal{I}$) flowing through this resistance. One can see that Ohm's law ($\mathcal{R} = \frac{\mathcal{V}_a - \mathcal{V}_v}{\mathcal{I}}$) gives the same result as equation (2.3).

Subscript:	Description:
a	Arterial
v	Venous
s	Systemic
p	Pulmonary
l	Left-side of the heart
r	Right-side of the heart

Table 2.1: Subscripts used in equations of this report.

The formula (2.3), in combination with the assumption that the systemic- and pulmonary vessels are lumped together, results in the following flows for those two parts of the model:

$$F_s = \frac{1}{R_s}\left(P_{as} - P_{vs}\right),$$
$$F_p = \frac{1}{R_p}\left(P_{ap} - P_{vp}\right).$$

Now the most important flow-related formulas are mentioned, a nice overview of the complete cardiovascular system is depicted in Figure 2.1. This block-diagram shows all subscripts that are used in the equations of this report.

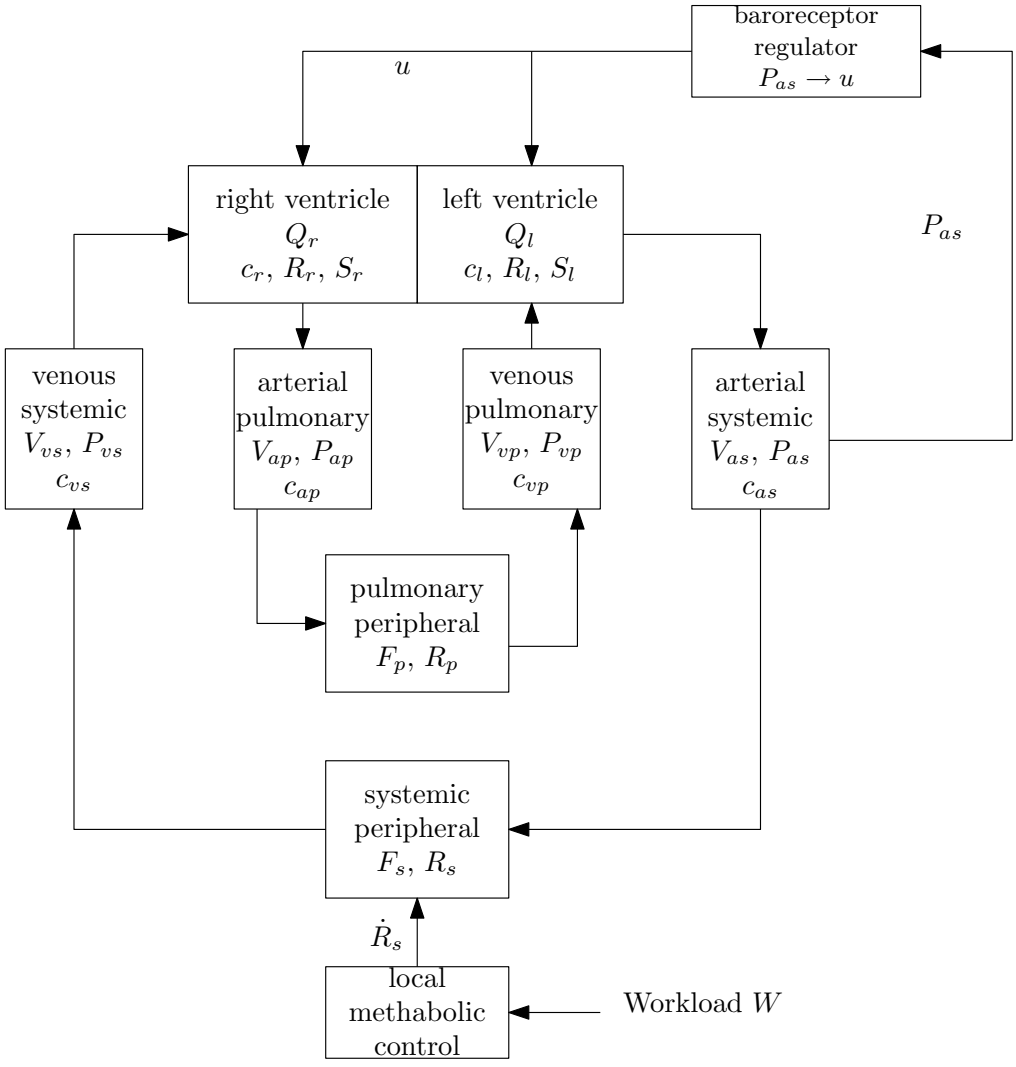


Figure 2.1: Block-diagram of the basic cardiovascular model.

2.2 Deriving a model for the system

2.2.1 States related with volume changes

One of the constant changing aspects in the complete system is the blood-volume in each compartment. This can be described as the rate of change $\dot{V} = \frac{dV}{dt}$, which is the difference between the flow into and the flow out of a compartment during a time period dt. With use of the equations (2.1) & (2.3) and with help of the block-diagram depicted in Figure 2.1, the following four differential equations can be derived:

$$\begin{aligned}
\dot{P}_{as} &= \frac{1}{c_{as}} \left\{ Q_l - \frac{1}{R_s} \left(P_{as} - P_{vs} \right) \right\}, \\
\dot{P}_{vs} &= \frac{1}{c_{vs}} \left\{ \frac{1}{R_s} \left(P_{as} - P_{vs} \right) - Q_r \right\}, \\
\dot{P}_{ap} &= \frac{1}{c_{ap}} \left\{ Q_r - \frac{1}{R_p} \left(P_{ap} - P_{vp} \right) \right\}, \\
\dot{P}_{vp} &= \frac{1}{c_{vp}} \left\{ \frac{1}{R_p} \left(P_{ap} - P_{vp} \right) - Q_l \right\}.
\end{aligned} \tag{2.4}$$

Those equations are the first four states of the model. It is possible to reduce this number of states to three, because the total volume of blood in the system should be constant:

$$V_{as} + V_{vs} + V_{ap} + V_{vp} = P_{as}c_{as} + P_{vs}c_{vs} + P_{ap}c_{ap} + P_{vp}c_{vp} \equiv V_{tot}.$$

With this equation it is possible to express a pressure, for example P_{vp}, as function of the other ones (so, $P_{vp}\left(P_{as}, P_{vs}, P_{ap}\right)$). If this substitution is applied, the value for V_{tot} has to be known. However, previous parameter estimation [1] has been shown that it is better to use all four pressures as states for the model.

The flows out of the heart (Q_l & Q_r) are parameters in the four differential equations (2.4), but in the model they are rewritten as functions of states and other parameters. Before it is possible to calculate the amount of blood flowing out of the heart during one stroke, the amount of blood flowing into the heart (during diastole) should be calculated.
The first step is the integration of (2.1) with respect to time (and taking the initial volume as V_{syst}, which is equal to the volume after the systole):

$$V(t) = V_{syst}e^{-\frac{t}{Rc}} + \left(cP_v + V_0\right)\left(1 - e^{-\frac{t}{Rc}}\right).$$

By substituting the time moment $t = t_d(H) = \frac{1}{\sqrt{H}}\left(\frac{1}{\sqrt{H}} - \kappa\right)$, which is the duration time of the diastole (with κ being a parameter), the amount of blood after the diastole-phase is equal to:

$$V_{diast} = k(H)V_{syst} + a(H)\left(cP_v + V_0\right), \tag{2.5}$$

where we have used the abbreviations $k(H) = e^{-\frac{t_d(H)}{Rc}}$ and $a(H) = 1 - k(H)$. Because of the Frank-Starling mechanism of the ventricle, also the following equation must hold:

$$V_{str} = \frac{S}{P_a}\left(V_{diast} - V_0\right), \tag{2.6}$$

where S is equal to the contractility of the ventricle muscle, and P_a is the arterial pressure which the ventricle has to eject (the lung-artery or the aorta) against.
So, by combining the equations (2.5) and (2.6) and knowing the relation (2.2), the two heart-flows can be described with:

$$\begin{aligned}
Q_l &= H\frac{c_l a_l(H) P_{vp} S_l}{a_l(H)P_{as} + k_l(H)S_l}, \\
Q_r &= H\frac{c_r a_r(H) P_{vs} S_r}{a_r(H)P_{ap} + k_r(H)S_r}.
\end{aligned} \tag{2.7}$$

With rewriting the two parameters (Q_l & Q_r) as those formulas, the number of extra parameters is reduced to only one (κ, which is in the function for $t_d(H)$) instead of two (Q_l & Q_r). (The other parameters in (2.7) are also used in (2.4), and S and H will be states later on.)

2.2.2 States related with ventricle activity

The contractility of the ventricle is limited, because $V_{str} \leq V_{diast}$. This implies an upper-bound for the value of S. The force also has to be positive, so the lower-bound is also known:

$$0 \leq S \leq \frac{a(H)\,(CP_v + V_0)}{a(H)\,(cP_v + V_0) - V_0} P_a. \tag{2.8}$$

Therefore the parameter S (which will be a state later on) should be rewritten as $f\,(S, P_a)$:

$$f\,(S, P_a) = \min\left(S, \frac{a(H)\,(CP_v + V_0)}{a(H)\,(cP_v + V_0) - V_0} P_a\right) \qquad , \qquad S \geq 0. \tag{2.9}$$

To keep the model simple, one can assume that $S_l \leq P_{as}$ and $S_r \leq P_{ap}$, which results in $f\,(S_l, P_{as}) = S_l$ and $f\,(S_r, P_{ap}) = S_r$.
The contractility of the ventricles can be described with a second order differential equation.

$$\ddot{S} + \gamma \dot{S} + \alpha S = \beta H, \tag{2.10}$$

where γ, α and β are positive constants.
This differential equation (2.10) results into four equations, that describe states in the model:

$$\begin{aligned} \dot{S}_l &= \sigma_l, \\ \dot{\sigma}_l &= \beta_l H - \alpha_l S_l - \gamma_l \sigma_l, \\ \dot{S}_r &= \sigma_r, \\ \dot{\sigma}_r &= \beta_r H - \alpha_r S_l - \gamma_r \sigma_r. \end{aligned} \tag{2.11}$$

2.2.3 State related with the amount of oxygen supply to the tissues

The amount of oxygen that is needed by the tissues is regulated with the local metabolic control system of the body. By increasing or decreasing the resistance of the (systemic) vessels, the blood flow can be controlled. This control system can be described with the following formula [7]:

$$R_s = A_{pesk} C_{v,O_2}, \tag{2.12}$$

where A_{pesk} is a positive constant and C_{v,O_2} is the concentration of O_2 in the venous blood. The metabolic rate for each tissue M_T is partly satisfied by the O_2 supply provided by the blood flow and partially by anaerobic biochemical reactions that provide an anaerobic energy flow M_b. This can be described with the following equation:

$$M_T = F_s\,(C_{a,O_2} - C_{v,O_2}) + M_b, \tag{2.13}$$

where C_{a,O_2} is the O_2 concentration in the arterial blood (which is assumed to be constant). The anaerobic energy flow M_b depends on the rate of change of C_{v,O_2}, so:

$$M_b = -K \frac{d}{dt} C_{v,O_2}, \tag{2.14}$$

where the constant $K > 0$.
By taking the derivative of (2.12) with respect to time, and by substituting the equations (2.3), (2.13) and (2.14), the following state equation can be derived:

$$\dot{R}_s = \frac{1}{K}\left\{A_{pesk}\left(\frac{P_{as} - P_{vs}}{R_s} C_{a,O_2} - M_T\right) - (P_{as} - P_{vs})\right\}. \tag{2.15}$$

The cardiovascular system needs to be modeled with a certain ergometric workload W as input, instead of the variable M_T. This can be done by writing the formula for the metabolic rate M_T as function of this workload W.

$$M_T = M_0 + \rho W,$$

where M_0 is the metabolic rate in the systemic tissue region corresponding to zero workload and ρ is a positive constant.

2.2.4 State related with the baroreceptor loop

The baroreceptor loop will not be described in so much detail as in [1] in this report, because a complete other technique is used in this report. This loop is modeled with a feedback controller which regulates the heart rate. In the real system, the baroreceptor loop uses baroreceptors (nerve endings) that measure the arterial systemic pressure (P_{as}). The signals from those baroreceptors are processed in the medulla, which indirectly controls the heart rate H. Because of its unknown behaviour this relation is just written in general as:

$$\dot{H} = u(t, \cdot). \tag{2.16}$$

In (2.16), $u(t, \cdot)$ can be described with use of an LQR controller, which is done in the book [1]. However in this research report this type of regulator will be replaced with a receding horizon optimal controller.

2.2.5 Complete model

Now, all states of the model are formulated in differential equations.
An overview of all those equations is given by (2.17).

$$\begin{aligned}
\dot{P}_{as} &= \frac{1}{c_{as}} \left\{ H \frac{c_l a_l(H) P_{vp} S_l}{a_l(H) P_{as} + k_l(H) S_l} - \frac{1}{R_s} (P_{as} - P_{vs}) \right\} \\
\dot{P}_{vs} &= \frac{1}{c_{vs}} \left\{ \frac{1}{R_s} (P_{as} - P_{vs}) - H \frac{c_r a_r(H) P_{vs} S_r}{a_r(H) P_{ap} + k_r(H) S_r} \right\} \\
\dot{P}_{ap} &= \frac{1}{c_{ap}} \left\{ H \frac{c_r a_r(H) P_{vs} S_r}{a_r(H) P_{ap} + k_r(H) S_r} - \frac{1}{R_p} (P_{ap} - P_{vp}) \right\} \\
\dot{P}_{vp} &= \frac{1}{c_{vp}} \left\{ \frac{1}{R_p} (P_{ap} - P_{vp}) - H \frac{c_l a_l(H) P_{vp} S_l}{a_l(H) P_{as} + k_l(H) S_l} \right\} \\
\dot{S}_l &= \sigma_l \\
\dot{\sigma}_l &= \beta_l H - \alpha_l S_l - \gamma_l \sigma_l \\
\dot{S}_r &= \sigma_r \\
\dot{\sigma}_r &= \beta_r H - \alpha_r S_r - \gamma_r \sigma_r \\
\dot{R}_s &= \frac{1}{K} \left\{ A_{pesk} \left(\frac{P_{as} - P_{vs}}{R_s} C_{a,O_2} - (M_0 + \rho W) \right) - (P_{as} - P_{vs}) \right\} \\
\dot{H} &= u(t, \cdot)
\end{aligned} \tag{2.17}$$

Knowing all those equations, it is (unfortunately) not possible to create a state-space system in the general form, because of non-linearities:

$$\begin{aligned}
\underline{\dot{x}} &= A\underline{x} + B\underline{u}, \\
\underline{y} &= C\underline{x} + D\underline{u}.
\end{aligned}$$

However, it is possible to define the state vector $\underline{x}$ (at a certain time moment) as:

$$\underline{x} = \{P_{as}, P_{vs}, P_{ap}, P_{vp}, S_l, \sigma_l, S_r, \sigma_r, R_s, H\}^T \in \mathbb{R}^{10}.$$

The input in our model $\underline{u}$ is the output of the controller for the baroreceptor loop u at some time moment:

$$\underline{u} = \{u\}^T \in \mathbb{R}.$$

There can be a lot of outputs for this system. However, while doing measurements, only two states are measured during time:

$$\underline{y} = \{H, P_{as}\}^T \in \mathbb{R}^2.$$

The system contains a lot of parameters, which are collected in a parameter-vector $\underline{p}$. It is possible to determine some of them a priori, but also parameter estimation will be needed.

$$\underline{p} = \{c_{as}, c_{vs}, c_{ap}, c_{vp}, c_l, c_r, R_p, R_l, R_r, \kappa, C_{a,O_2}, K, A_{pesk}, M_0, \rho, \alpha_l, \alpha_r, \beta_l, \beta_r, \gamma_l, \gamma_r, W\}^T \in \mathbb{R}^{22}$$

In general this total set of equations can be written as:

$$\underline{\dot{x}}(t) = \mathcal{F}\left(\underline{x}(t), u(t), \underline{p}\right), \text{ which can be shortened to } \underline{\dot{x}}(t) = \mathcal{F}\left(\underline{x}(t), u(t)\right). \tag{2.18}$$

2.3 Simulating the model in Matlab

Below some simulations of this model with use of the **ode45**-solver in Matlab.
Because the LQR-controller is not implemented in my Matlab m-files, the system input $u(t)$ is fixed to zero. At the change from the rest phase to the exercise phase there is a manual adjustment of the heart rate H to a value of 105. The parameters used in the simulation are taken from previous parameter estimation with the LQR-controller implemented and specified in Appendix A.

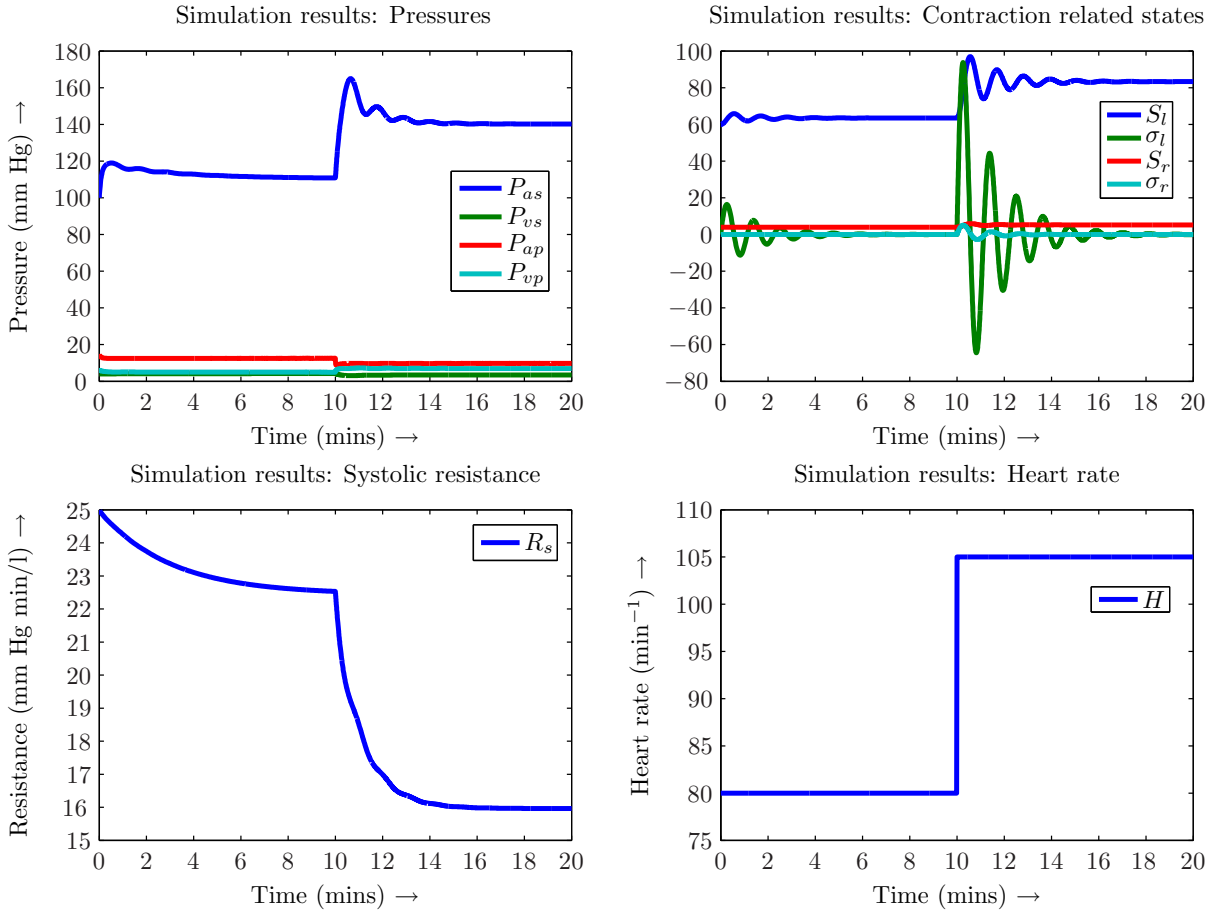


Figure 2.2: Simulations results of the system without control.

CHAPTER 3

The receding horizon controller

3.1 Introduction

As mentioned in the previous chapter, the baroreceptor loop of the cardiovascular system is stabilized using a Linear Quadratic Regulator (LQR). This type of controller needs a linear (or linearized in [1]) model that has to be controlled. Because of this (undesired) linearisation, a lot of nonlinear phenomena are not taken into account. That is the reason why another approach needs to be investigated. In this chapter, a receding horizon controller [8, 2] is introduced and implemented for this cardiovascular model. To create such a controller, the optimisation problem that needs to be solved has to be defined. When this optimisation problem is known, the cost can be calculated using the starting values for the states and the input of the cardiovascular model. To compute the gradient of the cost, the Lagrange function is defined, which results in the adjoint or dual functions. With those adjoint functions, the gradient of the cost function can be estimated, which then can be used to adjust the input of the model (done using the gradient projection method and backtracking). With this new input function the cost should be lower, which results in a better behaviour of the system! This update cycle should be applied for each short time period to get good control results. This division into small time intervals explains why the controller is called a receding horizon controller.

3.2 General optimisation problem

Before starting with the definition of the (infinite-dimensional) optimisation problem, some function spaces need to be introduced. First of all the input functions, which are in the so called U function space. Those functions can be any arbitrary finite function from $t = 0$ till $t = T$, which don't have to be differentiable with respect to time. In the general model M number of functions are used as input. Knowing this, those input functions $\underline{u}$ can be described with:

$$\underline{u} \in U = L^2\left(0, T : \mathbb{R}^M\right), \text{with the norm:} \quad \|\underline{u}\|_U = \sqrt{\langle \underline{u}, \underline{u} \rangle_U} = \sqrt{\int_0^T \|\underline{u}(t)\|^2_{\mathbb{R}^M} dt}.$$

Because the model contains differential equations, the function space for the states X can not be of the same type as the function space for the inputs. Not only the functions $\underline{x}$ but also the derivative with respect to time $\dot{\underline{x}}$ should be in the same space! When taking N state functions and using the same time interval as defined for the input space U, this results in the following description:

$$\underline{x} \in X = H^1\left(0, T : \mathbb{R}^N\right),$$

with the inner product $\langle \underline{x},\ \tilde{\underline{x}} \rangle_X = \int_0^T \underline{x}(t)^T \tilde{\underline{x}}(t) + \dot{\underline{x}}(t)^T \dot{\tilde{\underline{x}}}(t)\ dt$ for $\underline{x}, \tilde{\underline{x}} \in X$
and the induced norm $\|\underline{x}\|_X = \sqrt{\langle \underline{x},\ \underline{x} \rangle_X}$.

One can denote that at a certain time moment $t \in (0,T)$:

$$t \longmapsto \underline{u}(t) \in \mathbb{R}^M \qquad \text{and} \qquad t \longmapsto \underline{x}(t) \in \mathbb{R}^N \qquad \text{and} \qquad t \longmapsto \underline{\dot{x}}(t) \in \mathbb{R}^N.$$

The condition $\underline{x} \in H^1(0,T:\mathbb{R}^N)$ implies that $\underline{x} \in L^2\left(0,T:\mathbb{R}^N\right)$ **and** $\underline{\dot{x}} \in L^2\left(0,T:\mathbb{R}^N\right)$.

The differential equations and the cost function are depending on both the input functions $\underline{u}$ and the state functions $\underline{x}$. So, another function space Z, with functions $\underline{z} = (\underline{x}, \underline{u})$, is introduced, which is equal to:

$$\underline{z} \in Z = X \times U.$$

Knowing those spaces, it is possible to describe the general cost function $J : Z \to \mathbb{R}$, which maps the functions $(\underline{x}, \underline{u}) \mapsto J(\underline{x}, \underline{u})$ (or $(\underline{x}, \underline{u}) = \underline{z} \mapsto J(\underline{z})$).
This cost function can be calculated for each arbitrary set of functions $\underline{x}$ **and** $\underline{u}$. However, because of the constraints that give a relation between $\underline{x}$ and $\underline{u}$, the valid solutions are only related with functions $\underline{u}$. This results in the following reduced cost:

$$\hat{J}(\underline{u}) = J(\underline{x}(\underline{u}), \underline{u}), \text{ where the optimal value for the input function } \underline{u}^* \text{ has to satisfy } \nabla \hat{J}(\underline{u}^*) = 0.$$

As mentioned, the optimisation problem does not only have a cost function, but has also some constraints. This is called constrained optimisation. Those constraints are given by some differential equations (with their initial conditions):

$$\underline{\dot{x}}(t) = \mathcal{F}\left(\underline{x}(t), \underline{u}(t)\right), \quad t \in (0,T) \qquad \text{with a certain initial condition} \qquad \underline{x}(0) = \underline{x}_0,$$

where $\mathcal{F} : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^N$. Those two equations can be rewritten with use of an operator $e(\underline{x}, \underline{u}) = e(\underline{z})$ as:

$$e = (e_1, e_2) : X \times U \longrightarrow L^2\left(0,T:\mathbb{R}^N\right) \times \mathbb{R}^N = Y \tag{3.1}$$

defined as:

$$(\underline{x}, \underline{u}) \longmapsto \begin{cases} e_1(\underline{x}, \underline{u}) = \underline{\dot{x}} - \mathcal{F}(\underline{x}, \underline{u}) & \in L^2(0,T:\mathbb{R}^N), \\ e_2(\underline{x}, \underline{u}) = \underline{x}(0) - \underline{x}_0 & \in \mathbb{R}^N. \end{cases} \tag{3.2}$$

Now the constrained optimisation problem can be described as followed:

$$\min_{(\underline{x},\underline{u})} J\left(\underline{x}, \underline{u}\right) = \min_{\underline{z}} J\left(\underline{z}\right) \qquad \text{subject to} \qquad e(\underline{x}, \underline{u}) = e(\underline{z}) = 0. \tag{3.3}$$

For this problem the Lagrangian, which is utilized to derive optimality conditions, is given by:

$$\begin{aligned} L(\underline{x}, \underline{u}, \Lambda) &= J(\underline{x}, \underline{u}) + \langle e(\underline{x}, \underline{u}), \Lambda \rangle_{L^2(0,T:\mathbb{R}^N)\times\mathbb{R}^N} \\ &= J(\underline{x}, \underline{u}) + \langle e_1(\underline{x}, \underline{u}), \underline{\Lambda}_1 \rangle_{L^2(0,T:\mathbb{R}^N)} + \langle e_2(\underline{x}, \underline{u}), \underline{\Lambda}_2 \rangle_{\mathbb{R}^N}, \end{aligned} \tag{3.4}$$

where Λ contains the two Lagrange multiplier vectors $\underline{\lambda}_1(t)$ $(= \underline{\Lambda}_1 \in L^2(0,T:\mathbb{R}^N))$ and $\underline{\lambda}_2$ $(= \underline{\Lambda}_2 \in \mathbb{R}^N)$. Combining (3.2) and (3.4) results in:

$$L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) = J\left(\underline{x}, \underline{u}\right) + \int_0^T \left(\underline{\dot{x}}(t) - \mathcal{F}\left(\underline{x}(t), \underline{u}(t)\right)\right)^T \underline{\lambda}_1(t) dt + \left(\underline{x}(0) - \underline{x_0}\right)^T \underline{\lambda}_2. \tag{3.5}$$

A first-order necessary optimality condition for (3.3) is written as:

$$\begin{aligned} \nabla L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right)\left(\delta\underline{x}, \delta\underline{u}, \delta\underline{\lambda}_1, \delta\underline{\lambda}_2\right) &= \nabla_{\underline{x}} L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) \delta\underline{x} + \nabla_{\underline{u}} L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) \delta\underline{u} \\ &\quad + \nabla_{\underline{\lambda}_1} L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) \delta\underline{\lambda}_1 + \nabla_{\underline{\lambda}_2} L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) \delta\underline{\lambda}_2 = 0 \end{aligned} \tag{3.6}$$

at a point $(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2) \in Z \times Y$ and for all directions $(\delta\underline{x}, \delta\underline{u}) \in X, (\delta\underline{\lambda}_1, \delta\underline{\lambda}_2) \in Y$.
Equation (3.6) should hold for all small variations in $\underline{x}$, $\underline{u}$, $\underline{\lambda}_1$ and $\underline{\lambda}_2$ (so, $\delta\underline{x}$, $\delta\underline{u}$, $\delta\underline{\lambda}_1$ and $\delta\underline{\lambda}_2$). This results in the following set of equations that needs to be solved:

$$\begin{aligned} \nabla_{\underline{x}} L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) \delta\underline{x} = 0 \qquad & \forall \delta\underline{x} \in X \\ \nabla_{\underline{u}} L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) \delta\underline{u} = 0 \qquad & \forall \delta\underline{u} \in U \\ \nabla_{\underline{\lambda}_1} L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) \delta\underline{\lambda}_1 = 0 \qquad & \forall \delta\underline{\lambda}_1 \in L^2(0,T:\mathbb{R}^N) \quad \Leftrightarrow \quad e_1(\underline{x}, \underline{u}) = 0 \\ \nabla_{\underline{\lambda}_2} L\left(\underline{x}, \underline{u}, \underline{\lambda}_1, \underline{\lambda}_2\right) \delta\underline{\lambda}_2 = 0 \qquad & \forall \delta\underline{\lambda}_2 \in \mathbb{R}^N \quad \Leftrightarrow \quad e_2(\underline{x}, \underline{u}) = 0 \end{aligned} \tag{3.7}$$

3.3 Optimisation problem for the cardiovascular model

The model described by (2.17) contains ten states, which results in $\underline{x} : (0,T) \rightarrow \mathbb{R}^{10}$ (so, $N = 10$). As initial state vector $\underline{x}(0)$ the system is in rest or in exercise, so $\underline{x}(0) = \underline{x}^{rest}$ or $\underline{x}(0) = \underline{x}^{exer}$. In general this is described with $\underline{x}(0) = \underline{x}_0$.
There is only one input (so, $M = 1$), which results in the input function $\underline{u} = u$, where $u : (0,T) \rightarrow \mathbb{R}$.
Because of those dimensions, the Lagrange multipliers can be described with:

$$\underline{\lambda}_1 : (0,T) \rightarrow \mathbb{R}^{10} \text{ and } \underline{\lambda}_2 \in \mathbb{R}^{10}.$$

Before being able to solve the optimisation problem, the cost function needs to be defined. The controller has to drive the system from the current state ($\underline{x}(t)$) to a desired equilibrium state ($\underline{x}^{eq}$). In the baroreceptor loop, the arterial systemic pressure (P_{as}) is measured by the baroreceptors. This state needs to be included in the cost function. Of course, also the input of the system (u) should not be too large. For a receding horizon controller, the error at the end should be suppressed to keep the loop stable. This can be done by adding the last term described in the next formula. Those three aspects result in the following cost function for this problem:

$$\begin{aligned} J(\underline{x},u) &= \int_0^T \left[q_{as}^2, 0, \ldots, 0\right] \left(\underline{x}(t) - \underline{x}^{eq}\right)^2 + \kappa_c^2 u(t)^2 dt + \frac{1}{2} \left[\alpha_c, 0, \ldots, 0, \right] \left(\underline{x}(T) - \underline{x}^{eq}\right)^2 \\ &= \int_0^T q_{as}^2 \left(P_{as}(t) - P_{as}^{eq}\right)^2 + \kappa_c^2 u(t)^2 dt + \frac{\alpha_c}{2} \left(P_{as}(T) - P_{as}^{eq}\right)^2. \end{aligned}$$

In this cost function, q_{as}, κ_c & α_c are weighting factors which can be chosen. If the factor q_{as} is taken large (with respect to κ), the difference between the arterial systemic pressure at the current time moment and the value in the equilibrium point will become very small, but the input u will be large then.

Knowing the cost function, it is possible to summarize the complete problem (because the differential equations are defined in (2.17) and the initial condition is also known ($\underline{x}(0) = \underline{x}_0$)):

$$\min_{\underline{x},u} J\left(\underline{x},u\right) = \int_0^T q_{as}^2 \left(P_{as}(t) - P_{as}^{eq}\right)^2 + \kappa_c^2 u(t)^2 dt + \frac{\alpha_c}{2} \left(P_{as}(T) - P_{as}^{eq}\right)^2 \tag{3.8}$$

subject to

$$\begin{aligned} &\dot{\underline{x}}(t) = \mathcal{F}\left(\underline{x}(t), u(t)\right), \quad t \in (0,T), \\ &\underline{x}(0) = \underline{x}_0. \end{aligned} \tag{3.9}$$

Knowing this optimisation problem, the Lagrange function can be formulated:

$$L\left(\underline{x},u,\underline{\lambda}_1,\underline{\lambda}_2\right) = J\left(\underline{x},u\right) + \int_0^T \left(\dot{\underline{x}}(t) - \mathcal{F}\left(\underline{x}(t),u(t)\right)\right)^T \underline{\lambda}_1(t) dt + \left(\underline{x}(0) - \underline{x}_0\right)^T \underline{\lambda}_2, \tag{3.10}$$

for $\underline{x} \in H^1(0,T:\mathbb{R}^{10}), u \in L^2(0,T:\mathbb{R}), \underline{\lambda}_1 \in L^2(0,T:\mathbb{R}^{10})$ and $\underline{\lambda}_2 \in \mathbb{R}^{10}$.
The optimality condition (3.7) should be applied to this Lagrange function, which results in two problems: The gradient of the Lagrangian with respect to the state-functions $\underline{x}$ and with respect to the input function u.

3.4 Gradient with respect to the state equations

In this section, the Lagrange function is differentiated with respect to the state functions in $\underline{x}$. In the first subsection, this will be done by hand, which takes quite some time doing this without making mistakes. Later on during the traineeship project, two members of the mathematical department (Jerry Batzel and Mostafa Bachar) explained me how to work with automatic differentiation using a toolbox in Matlab [3]. This nice way of differentiation saves a lot of time, and is shortly explained in Section 3.4.2. With this method, the solution of the derivation by hand can be verified, and it also saves time if there is a change applied in the model (so, the total derivation doesn't have to be done over by hand again).

3.4.1 Calculating the gradient by hand

In this subsection, the problem described in the first line of (3.7) needs to be solved for the cardiovascular model. Substituting this problem results in the following equation, which can be split into three parts (A,B & C).

$$\begin{aligned}\nabla_{\underline{x}} L\left(\underline{x},\underline{u},\underline{\lambda}_1,\underline{\lambda}_2\right)\delta\underline{x} = &\overbrace{\nabla_{\underline{x}}\left[J\left(\underline{x},u\right)\right]\delta\underline{x}}^{A}\\ &+\overbrace{\nabla_{\underline{x}}\left[\int_0^T\left(\dot{\underline{x}}(t)-\mathcal{F}\left(\underline{x}(t),u(t)\right)\right)^T\underline{\lambda}_1(t)dt\right]\delta\underline{x}}^{B}\\ &+\overbrace{\nabla_{\underline{x}}\left[\left(\underline{x}(0)-\underline{x}^{rest}\right)^T\underline{\lambda}_2\right]\delta\underline{x}}^{C}=0\end{aligned} \tag{3.11}$$

at a point $(\underline{x},\underline{u},\underline{\lambda}_1,\underline{\lambda}_2)\in Z\times Y$ and for any $\delta\underline{x},\in X$.

A: The gradient of the cost function:

The cost function is only related with one state (P_{as}) and the input of the system (u), so taking this derivative with respect to the vector of state functions ($\underline{x}$) results in:

$$\begin{aligned}\nabla_{\underline{x}}\,J\left(\underline{x},u\right)\delta\underline{x} &= \sum_{i=1}^{10}\nabla_{x_i}\,J\left(\underline{x},u\right)\delta x_i = \nabla_{P_{as}}J\left(\underline{x},u\right)\delta P_{as}\\ &= \int_0^T 2q_{as}^2\left(P_{as}(t)-P_{as}^{eq}\right)\delta P_{as}(t)dt+\alpha_c\left(P_{as}(T)-P_{as}^{eq}\right)\delta P_{as}(T)\\ &= \int_0^T 2q_{as}^2\left(Q(\underline{x}(t)-\underline{x}^{eq})\right)^T\delta\underline{x}(t)dt+\alpha_c\left(Q\left(\underline{x}(T)-\underline{x}^{eq}\right)\right)^T\delta\underline{x}(T)\end{aligned}$$

where $Q=$diag$(\underline{q})\in\mathbb{R}^{10\times10}$ and $\underline{q}=[1,0,\ldots,0]^T\in\mathbb{R}^{10}$.

B: The gradient of the term with the integral:

This part of the gradient is a bit more difficult then the previous one, because it contains a set of differential equations (which contain all state functions $\underline{x}$) that needs to be differentiated. First the given part (B) of the total gradient function (3.11) can be rewritten as:

$$\begin{aligned}\nabla_{\underline{x}}\left[\int_0^T\left(\dot{\underline{x}}(t)-\mathcal{F}\left(\underline{x}(t),u(t)\right)\right)^T\underline{\lambda}_1(t)dt\right]\delta\underline{x} &= \int_0^T\left(\delta\dot{\underline{x}}(t)-\mathcal{F}_{\underline{x}}(\underline{x}(t),u(t))\delta\underline{x}\right)^T\underline{\lambda}_1(t)dt\\ &= \underbrace{\int_0^T\delta\dot{\underline{x}}(t)^T\underline{\lambda}_1(t)dt}_{(\Gamma)}-\int_0^T\left(\mathcal{F}_{\underline{x}}(\underline{x}(t),u(t))\delta\underline{x}(t)\right)^T\underline{\lambda}_1(t)dt.\end{aligned} \tag{3.12}$$

The part (Γ) in the above mentioned equation contains the derivative (with respect to time) of the small deviation in the state functions. However differential equations for the functions $\underline{\lambda}_1$ are required, because those make it possible to solve the first-order system in $\underline{x},u,\underline{\lambda}_1,\underline{\lambda}_2$. So, (Γ) has to be rewritten with use of partial integration:

$$\int_0^T\delta\dot{\underline{x}}(t)^T\underline{\lambda}_1(t)dt=-\int_0^T\underbrace{\left(\delta\underline{x}(t)^T\dot{\underline{\lambda}}_1(t)\right)}_{\dot{\underline{\lambda}}_1(t)^T\delta\underline{x}(t)}dt+\delta\underline{x}(T)^T\underline{\lambda}_1(T)-\delta\underline{x}(0)^T\underline{\lambda}_1(0)$$

This equation is substituted back into the previous solution of the partial gradient (3.12), which results in:

$$\nabla_{\underline{x}}\left[\dots\right]\delta\underline{x} = -\int_0^T \left(\dot{\underline{\lambda}}_1(t)^T\delta\underline{x}(t) + \underline{\lambda}_1(t)^T\left(\mathcal{F}_{\underline{x}}(\underline{x}(t),u(t))\delta\underline{x}(t)\right)\right)dt + \underline{\lambda}_1(T)^T\delta\underline{x}(T) - \underline{\lambda}_1(0)^T\delta\underline{x}(0) \quad (3.13)$$

In (3.13), the derivative of the set of differential equations ($\mathcal{F}$) with respect to the state functions ($\underline{x}$) needs to be calculated. One can notice that it results in hard terms that need to be differentiated with respect to the state functions. The fraction in the first four differential equations has always the same form, so this part is (in general) rewritten as:

$$\theta\left(a(t_d),P_1,P_2,S\right) = H\frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S},$$

$$\text{where}\quad a(t_d) = 1-k(t_d) = 1-e^{-\frac{t_d(H)}{Rc}} \quad\text{and}\quad t_d(H) = \frac{1}{\sqrt{H}}\left(\frac{1}{\sqrt{H}}-\kappa\right)$$

The function $\theta\left(a(t_d),P_1,P_2,S\right)$ should be differentiated with respect to P_1, P_2, S & H:

$$\begin{aligned}
\frac{\partial}{\partial P_1}\theta\left(a(t_d),P_1,P_2,S\right) &= \frac{\partial}{\partial P_1}\left[H\frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S}\right] = H\frac{ca(t_d)S}{a(t_d)P_2+k(t_d)S}\\
&= Hca(t_d)\frac{S}{a(t_d)P_2+k(t_d)S},\\
\frac{\partial}{\partial P_2}\theta\left(a(t_d),P_1,P_2,S\right) &= \frac{\partial}{\partial P_2}\left[H\frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S}\right] = -H\frac{ca(t_d)^2P_1S}{\left(a(t_d)P_2+k(t_d)S\right)^2}\\
&= -Hca(t_d)\frac{a(t_d)P_1S}{\left(a(t_d)P_2+k(t_d)S\right)^2},\\
\frac{\partial}{\partial S}\theta\left(a(t_d),P_1,P_2,S\right) &= \frac{\partial}{\partial S}\left[H\frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S}\right] = \frac{\left(a(t_d)P_2+k(t_d)S\right)\cdot\left(Hca(t_d)P_1\right)-\left(Hca(t_d)P_1S\right)\cdot k(t_d)}{\left(a(t_d)P_2+k(t_d)S\right)^2}\\
&= Hca(t_d)\frac{a(t_d)P_1P_2}{\left(a(t_d)P_2+k(t_d)S\right)^2},\\
\frac{\partial}{\partial H}\theta\left(a(t_d),P_1,P_2,S\right) &= \frac{\partial}{\partial H}\left[H\frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S}\right] = \frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S} + H\frac{\partial}{\partial H}\left[\frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S}\right]\\
&= \frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S} + H\frac{\partial}{\partial H}\underbrace{\left[\frac{ca(t_d)P_1S}{a(t_d)P_2+(1-a(t_d))S}\right]}_{\Phi(a(t_d))},
\end{aligned}$$

$$\text{where}\quad \frac{\partial\Phi(a(t_d))}{\partial H} = \frac{\partial\Phi(a(t_d))}{\partial a(t_d)}\cdot\frac{\partial a(t_d)}{\partial H} \quad\text{and because } a(t_d) \text{ is a function of } t_d(H), \text{ this is:}$$

$$= \underbrace{\frac{\partial\Phi(a(t_d))}{\partial a(t_d)}}_{(I)}\cdot\underbrace{\frac{\partial a(t_d(H))}{\partial t_d(H)}}_{(II)}\cdot\underbrace{\frac{\partial t_d(H)}{\partial H}}_{(III)}$$

$$(I)\quad :\quad \frac{\partial\Phi(a(t_d))}{\partial a(t_d)} = \frac{cP_1S^2}{\left(a(t_d)P_2+(1-a(t_d))S\right)^2}$$

$$(II)\quad :\quad \frac{\partial a(t_d(H))}{\partial t_d(H)} = \frac{1}{Rc}e^{-\frac{t_d(H)}{Rc}}$$

$$(III)\quad :\quad \frac{\partial t_d(H)}{\partial H} = \frac{\kappa}{2}H^{-\frac{3}{2}} - H^{-2}$$

$$\text{so,}\quad \frac{\partial\Phi(a(t_d))}{\partial H} = \frac{cP_1S^2}{Rc\left(a(t_d)P_2+(1-a(t_d))S\right)^2}\left(\frac{\kappa}{2}H^{-\frac{3}{2}}-H^{-2}\right)e^{-\frac{t_d(H)}{Rc}} \quad\text{which results in:}$$

$$\frac{\partial}{\partial H}\theta\left(a(t_d),P_1,P_2,S\right) = \frac{ca(t_d)P_1S}{a(t_d)P_2+k(t_d)S} + \frac{cP_1S^2}{Rc\left(a(t_d)P_2+k(t_d)S\right)^2}\left(\frac{\kappa}{2}H^{-\frac{1}{2}}-H^{-1}\right)e^{-\frac{t_d(H)}{Rc}}.$$

Now some hard terms (especially the one with respect to H) for the derivatives of the fraction in the set of differential equations are defined, the derivation of the total set can start. (Note: in the formulas below, $\underline{x} \in \mathbb{R}^{10}$ and $u \in \mathbb{R}$, because $\mathcal{F}(\underline{x}, u, \underline{p})$ is a function evaluated at some time t)

Derivation with respect to P_{as}***:***

$$\frac{\partial}{\partial P_{as}}\mathcal{F}(\underline{x},u) = \begin{bmatrix} \frac{1}{c_{as}}\left\{-Hc_l a_l(t_d)\frac{a_l(t_d)P_{vp}S_l}{\left(a_l(t_d)P_{as}+k_l(t_d)S_l\right)^2}-\frac{1}{R_s}\right\} \\ \frac{1}{R_s c_{vs}} \\ 0 \\ \frac{1}{c_{vp}}\left\{Hc_l a_l(t_d)\frac{a_l(t_d)P_{vp}S_l}{\left(a_l(t_d)P_{as}+k_l(t_d)S_l\right)^2}\right\} \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{K}\left\{A_{pesk}\frac{C_{a,O_2}}{R_s}-1\right\} \\ 0 \end{bmatrix}$$

Derivation with respect to P_{vs}***:***

$$\frac{\partial}{\partial P_{vs}}\mathcal{F}(\underline{x},u) = \begin{bmatrix} \frac{1}{R_s c_{as}} \\ \frac{1}{c_{vs}}\left\{-\frac{1}{R_s}-Hc_r a_r(t_d)\frac{S_r}{a_r(t_d)P_{ap}+k_r(t_d)S_r}\right\} \\ \frac{1}{c_{ap}}Hc_r a_r(t_d)\frac{S_r}{a_r(t_d)P_{ap}+k_r(t_d)S_r} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -\frac{1}{K}\left\{A_{pesk}\frac{C_{a,O_2}}{R_s}-1\right\} \\ 0 \end{bmatrix}$$

Derivation with respect to P_{ap}***:***

$$\frac{\partial}{\partial P_{ap}}\mathcal{F}(\underline{x},u) = \begin{bmatrix} 0 \\ \frac{1}{c_{vs}}Hc_r a_r(t_d)\frac{a_r(t_d)P_{vs}S_r}{\left(a_r(t_d)P_{ap}+k_r(t_d)S_r\right)^2} \\ \frac{1}{c_{ap}}\left\{-Hc_r a_r(t_d)\frac{a_r(t_d)P_{vs}S_r}{\left(a_r(t_d)P_{ap}+k_r(t_d)S_r\right)^2}-\frac{1}{R_p}\right\} \\ \frac{1}{R_p c_{vp}} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Derivation with respect to P_{vp}:

$$\frac{\partial}{\partial P_{vp}}\mathcal{F}(\underline{x},u)=\begin{bmatrix}\frac{1}{c_{as}}Hc_la_l(t_d)\frac{S_l}{a_l(t_d)P_{as}+k_l(t_d)S_l}\\0\\\frac{1}{R_pc_{ap}}\\\frac{1}{c_{vp}}\left\{-\frac{1}{R_p}-Hc_la_l(t_d)\frac{S_l}{a_l(t_d)P_{as}+k_l(t_d)S_l}\right\}\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

Derivation with respect to S_l:

$$\frac{\partial}{\partial S_l}\mathcal{F}(\underline{x},u)=\begin{bmatrix}\frac{1}{c_{as}}Hc_la_l(t_d)\frac{a_l(t_d)P_{vp}P_{as}}{\left(a_l(t_d)P_{as}+k_l(t_d)S_l\right)^2}\\0\\0\\-\frac{1}{c_{vp}}Hc_la_l(t_d)\frac{a_l(t_d)P_{vp}P_{as}}{\left(a_l(t_d)P_{as}+k_l(t_d)S_l\right)^2}\\0\\-\alpha_l\\0\\0\\0\\0\end{bmatrix}$$

Derivation with respect to σ_l:

$$\frac{\partial}{\partial \sigma_l}\mathcal{F}(\underline{x},u)=[0,0,0,0,1,-\gamma_l,0,0,0,0]^T$$

Derivation with respect to S_r:

$$\frac{\partial}{\partial S_r}\mathcal{F}(\underline{x},u)=\begin{bmatrix}0\\-\frac{1}{c_{vs}}Hc_ra_r(t_d)\frac{a_r(t_d)P_{vs}P_{ap}}{\left(a_r(t_d)P_{ap}+k_r(t_d)S_r\right)^2}\\\frac{1}{c_{ap}}Hc_ra_r(t_d)\frac{a_r(t_d)P_{vs}P_{ap}}{\left(a_r(t_d)P_{ap}+k_r(t_d)S_r\right)^2}\\0\\0\\0\\0\\-\alpha_r\\0\\0\end{bmatrix}$$

Derivation with respect to σ_r:

$$\frac{\partial}{\partial \sigma_r}\mathcal{F}(\underline{x},u)=[0,0,0,0,0,0,1,-\gamma_r,0,0]^T$$

Derivation with respect to R_s:

$$\frac{\partial}{\partial R_s}\mathcal{F}(\underline{x},u)=\begin{bmatrix}\frac{P_{as}-P_{vs}}{R_s^2c_{as}}\\ -\frac{P_{as}-P_{vs}}{R_s^2c_{vs}}\\ \frac{P_{ap}-P_{vp}}{R_s^2c_{ap}}\\ -\frac{P_{ap}-P_{vp}}{R_s^2c_{vp}}\\ 0\\ 0\\ 0\\ 0\\ -\frac{A_{pesk}C_{a,O_2}(P_{as}-P_{vs})}{R_s^2K}\\ 0\end{bmatrix}$$

Derivation with respect to H:

$$\frac{\partial}{\partial H}\mathcal{F}(\underline{x},u)=\begin{bmatrix}\frac{1}{c_{as}}\left\{\frac{c_la_l(t_d)P_{vp}S_l}{a_l(t_d)P_{as}+k_l(t_d)S_l}+\frac{c_lP_{vp}S_l^2}{R_lc_l\left(a_l(t_d)P_{as}+k_l(t_d)S_l\right)^2}\left(\frac{\kappa}{2}H^{-\frac{1}{2}}-H^{-1}\right)e^{-\frac{t_d(H)}{R_lc_l}}\right\}\\ -\frac{1}{c_{vs}}\left\{\frac{c_ra_r(t_d)P_{vs}S_r}{a_r(t_d)P_{ap}+k_r(t_d)S_r}+\frac{c_rP_{vs}S_r^2}{R_rc_r\left(a_r(t_d)P_{ap}+k_r(t_d)S_r\right)^2}\left(\frac{\kappa}{2}H^{-\frac{1}{2}}-H^{-1}\right)e^{-\frac{t_d(H)}{R_rc_r}}\right\}\\ \frac{1}{c_{ap}}\left\{\frac{c_ra_r(t_d)P_{vs}S_r}{a_r(t_d)P_{ap}+k_r(t_d)S_r}+\frac{c_rP_{vs}S_r^2}{R_rc_r\left(a_r(t_d)P_{ap}+k_r(t_d)S_r\right)^2}\left(\frac{\kappa}{2}H^{-\frac{1}{2}}-H^{-1}\right)e^{-\frac{t_d(H)}{R_rc_r}}\right\}\\ -\frac{1}{c_{vp}}\left\{\frac{c_la_l(t_d)P_{vp}S_l}{a_l(t_d)P_{as}+k_l(t_d)S_l}+\frac{c_lP_{vp}S^2}{R_lc_l\left(a_l(t_d)P_{as}+k_l(t_d)S_l\right)^2}\left(\frac{\kappa}{2}H^{-\frac{1}{2}}-H^{-1}\right)e^{-\frac{t_d(H)}{R_lc_l}}\right\}\\ 0\\ \beta_l\\ 0\\ \beta_r\\ 0\\ 0\end{bmatrix}$$

Now all derivatives to each state function in $\underline{x}$ are calculated, the total matrix $\mathcal{F}_{\underline{x}}$ contains all those vectors:

$$\mathcal{F}_{\underline{x}}=\Big[\frac{\partial}{\partial P_{as}}\mathcal{F}(\underline{x},u,\underline{p}),\frac{\partial}{\partial P_{vs}}\mathcal{F}(\underline{x},u,\underline{p}),\frac{\partial}{\partial P_{ap}}\mathcal{F}(\underline{x},u,\underline{p}),\frac{\partial}{\partial P_{vp}}\mathcal{F}(\underline{x},u,\underline{p}),\frac{\partial}{\partial S_l}\mathcal{F}(\underline{x},u,\underline{p}),\frac{\partial}{\partial \sigma_l}\mathcal{F}(\underline{x},u,\underline{p}),$$
$$\frac{\partial}{\partial S_r}\mathcal{F}(\underline{x},u,\underline{p}),\frac{\partial}{\partial \sigma_r}\mathcal{F}(\underline{x},u,\underline{p}),\frac{\partial}{\partial R_s}\mathcal{F}(\underline{x},u,\underline{p}),\frac{\partial}{\partial H}\mathcal{F}(\underline{x},u,\underline{p})\Big]\in\mathbb{R}^{10\times 10}$$

C: The gradient of the last term in (3.11):

The last term of the optimality condition equation is:

$$\nabla_{\underline{x}}\left[\left(\underline{x}(0)-\underline{x}_0\right)^T\underline{\lambda}_2\right]\delta\underline{x}=\delta\underline{x}(0)^T\underline{\lambda}_2=\underline{\lambda}_2^T\delta\underline{x}(0)$$

Combining part A, B & C:

So, combining those three parts mentioned in the previous sections, the following optimisation condition must hold:

$$\begin{aligned}&\int_0^T\left[2q_{as}^2\left(Q(\underline{x}(t)-\underline{x}^{eq})\right)^T\delta\underline{x}(t)\right]dt-\int_0^T\left(\dot{\underline{\lambda}}_1(t)^T\delta\underline{x}(t)+\underline{\lambda}_1(t)^T\left(\mathcal{F}_{\underline{x}}(\underline{x}(t),u(t),\underline{p})\delta\underline{x}(t)\right)\right)dt\\&+\left(\underline{\lambda}_1(T)+\alpha Q\left(\underline{x}(T)-\underline{x}^{eq}\right)\right)^T\delta\underline{x}(T)-\left[\underline{\lambda}_1(0)^T-\underline{\lambda}_2^T\right]\delta\underline{x}(0)=0\end{aligned}\tag{3.14}$$

This equation must hold for every $\delta\underline{x} \in X$, so in particular for variations $\delta\underline{x}$ satisfying $\delta\underline{x}(0) = \delta\underline{x}(T) = 0$. Formula (3.14) is then reduced to:

$$\int_0^T \left[2q_{as}^2 \left(Q(\underline{x}(t) - \underline{x}^{rest})\right)^T \delta\underline{x}(t)\right] dt - \int_0^T \left(\dot{\underline{\lambda}}_1(t)^T \delta\underline{x}(t) + \underline{\lambda}_1(t)^T \left(\mathcal{F}_{\underline{x}}(\underline{x}(t), u(t))\delta\underline{x}(t)\right)\right) dt = 0$$

which results in the equation:

$$\begin{aligned}\int_0^T \left[2q_{as}^2 \left(Q(\underline{x}(t) - \underline{x}^{eq})\right)^T \delta\underline{x}(t)\right] dt &= \int_0^T \left(\dot{\underline{\lambda}}_1(t)^T \delta\underline{x}(t) + \underline{\lambda}_1(t)^T \left(\mathcal{F}_{\underline{x}}(\underline{x}(t), u(t))\delta\underline{x}(t)\right)\right) dt \\ &= \int_0^T \left(\dot{\underline{\lambda}}_1(t)^T + \underline{\lambda}_1(t)^T \mathcal{F}_{\underline{x}}(\underline{x}(t), u(t))\right) \delta\underline{x}(t) dt.\end{aligned} \tag{3.15}$$

Hence,

$$2q_{as}^2 \left(Q(\underline{x} - \underline{x}^{eq})\right)^T = \dot{\underline{\lambda}}_1^T + \underline{\lambda}_1^T \mathcal{F}_{\underline{x}}(\underline{x}, u)$$

and:

$$2q_{as}^2 Q(\underline{x} - \underline{x}^{eq}) = \dot{\underline{\lambda}}_1 + \mathcal{F}_{\underline{x}}(\underline{x}, u)^T \underline{\lambda}_1.$$

This results into the differential equation for $\underline{\lambda}_1(t)$:

$$\dot{\underline{\lambda}}_1(t) + \mathcal{F}_{\underline{x}}(\underline{x}(t), u(t))^T \underline{\lambda}_1(t) - 2q_{as}^2 Q(\underline{x}(t) - \underline{x}^{eq}) = \underline{0}, \quad t \in (0, T). \tag{3.16}$$

It is also possible that the small deviations in the state functions ($\delta\underline{x}$) at time $t = T$ and time $t = 0$ are not equal to zero. When applying those conditions to (3.14), this results in the conditions that:

$$\begin{aligned}&\underline{\lambda}_1(T) = -\alpha_c Q\left(\underline{x}(T) - \underline{x}^{eq}\right), \\ &\underline{\lambda}_1(0) = \underline{\lambda}_2.\end{aligned}$$

3.4.2 Automatic differentiation

The basics of automatic differentiation are published in [9]. Automatic differentiation is based on the basic mathematical operations (like sin(),sqrt(), etc, ...) which are known by every mathematical computer program. All possible (complicated) functions can be rewritten as a composition of those basic functions. A very simple example is $y = 25 * \sin(\cos(x))$, which can be written as $y = 25 * a$, with $a = \sin(b)$ and $b = \cos(x)$. All those basic mathematical operations are known by the program and also their derivatives are available. Automatic differentiation makes use of this by applying the chain rule. The derivative of the complicated function then can be rewritten as a multiplication of simple derivatives (in our example this will result in: $\frac{\delta y}{\delta x} = \frac{\delta y}{\delta a} \cdot \frac{\delta a}{\delta b} \cdot \frac{\delta b}{\delta x}$). That is how automatic differentiation works. A package that can be used is Mad, a mathematical automatic differentiation toolbox for Matlab, produced by the company TOMLAB Optimization [3]. The University of Graz has a license for this toolbox, so I have used this toolbox to verify the derivation of the $\mathcal{F}_{\underline{x}}$ matrix, which is described earlier in the previous subsection. The code to do this is quite simple and also attached in the appendix with Matlab files used during the traineeship. This toolbox is also good for estimating sensitivities of parameters in the model.

3.5 Gradient with respect to the input

Not only the gradient with respect to the states $\underline{x}$ has to be equal to zero, but also the gradient with respect to the input vector $\underline{u}$, which is only one input in our model, has to be zero:

$$\begin{aligned}\nabla_u L\left(\underline{x}, u, \underline{\lambda}_1 \underline{\lambda}_2\right) \delta u &= \int_0^T 2\kappa_c^2 u(t)\delta u(t) dt + \int_0^T -\underline{\beta}^T \underline{\lambda}_1(t)\delta u(t) dt \\ &= \int_0^T \left(2\kappa_c^2 u(t) - \underline{\beta}^T \underline{\lambda}_1(t)\right) \delta u(t) dt = 0\end{aligned} \tag{3.17}$$

where $\underline{\beta} = [0, \ldots, 0, 1]^T \in \mathbb{R}^{10}$.

Now the following theorem has been proved:

Theorem: *The first-order necessary optimisation conditions for problem (3.8) are given by: The state equations:*

$$\begin{aligned} &\dot{\underline{x}}(t) = \mathcal{F}\left(\underline{x}(t), u(t)\right), \quad t \in (0, T), \\ &\underline{x}(0) = \underline{x}_0, \end{aligned} \tag{3.18}$$

the adjoint equations:

$$\begin{aligned} &\dot{\underline{\lambda}}_1(t) + \mathcal{F}_{\underline{x}}(\underline{x}(t), u(t))^T \underline{\lambda}_1(t) - 2q_{as}^2 Q(\underline{x}(t) - \underline{x}^{eq}) = \underline{0}, \quad t \in (0, T) \\ &\underline{\lambda}_1(T) = -\alpha_c Q\left(\underline{x}(T) - \underline{x}^{eq}\right) \\ &\underline{\lambda}_1(0) = \underline{\lambda}_2, \end{aligned} \tag{3.19}$$

and the optimality condition:

$$2\kappa_c^2 u(t) = \underline{\beta}^T \underline{\lambda}_1(t), \quad t \in [0, T]. \tag{3.20}$$

3.6 Updating the model input function

Before being able to define a function which updates the input function u, so that the reduced cost $\hat{J}(u)$ and its gradient with respect to u, $\nabla_u \hat{J}(u)$, are reduced, some aspects of functional analysis need to be discussed.

Some functional analysis

The goal of the optimisation is to minimize the reduced cost $\hat{J}(u)$ which is equal to $J(\underline{x}(u), u)$. This can be achieved if the necessary optimality condition of first order holds: $\nabla_u \hat{J}(u) = 0$. This gradient can be rewritten in terms of inner products:

$$\langle \nabla_u \hat{J}(u)\ ,\ \delta u \rangle_U = \langle \nabla_{\underline{x}} J(\underline{x}(u), u)\ ,\ \underline{x}'(u)\delta u \rangle_X + \langle \nabla_u J(\underline{x}(u), u)\ ,\ \delta u \rangle_U \quad \text{for } \delta u \in U.$$

In this formula, the term $\underline{x}'(u)\delta u$ is unknown, but by using a smart substitution this can be solved.
In (3.1) the e operator is introduced, which is equal to zero if $e(\underline{x}(u), u)$. First, let's recap the behaviour of the operator e:

$$\begin{aligned} e &: X \times U \to Y, \\ \nabla e(\underline{x}, u) &: X \times U \to Y, \\ \nabla_u e(\underline{x}, u) &: U \to Y, \\ \nabla_{\underline{x}} e(\underline{x}, u) &: X \to Y. \end{aligned}$$

Later on we shall make use of the inverse of the operator $\nabla_{\underline{x}} e(\underline{x}, u)$. We suppose that this operator is invertible for any $(\underline{x}, u) \in Z$. This inverse operator does map in the inverse direction, so for example: $\nabla_{\underline{x}} e(\underline{x}(u), u)^{-1} : Y \to X$.

Also the adjoint operator is needed in this subsection of the report. The adjoint $A^* : W \to V$ of an operator $A : V \to W$ is defined by:

$$\langle Av\ ,\ w \rangle_W = \langle v\ ,\ A^* w \rangle_V, \quad \forall v \in V \text{ and } \forall w \in W.$$

When taking the gradient of the operator $\hat{e}(u) = e(\underline{x}(u), u)$, this results in the following equation, which also contains the unknown term $\delta\underline{x} = \underline{x}'(u)\delta u$:

$$\underbrace{\nabla_u \hat{e}(u) \underbrace{\delta u}_{\in U}}_{\in Y} = \underbrace{\nabla_x e(\underline{x}(u), u) \underbrace{\underline{x}'(u)\delta u}_{\in X}}_{\in Y} + \underbrace{\nabla_u e(\underline{x}(u), u) \underbrace{\delta u}_{\in U}}_{\in Y} = 0.$$

If $\nabla_{\underline{x}} e(\underline{x}(u), u)$ is invertible this can be rewritten as:

$$\underbrace{\underbrace{\underline{x}'(u)\delta u}_{\in X} = -\nabla_{\underline{x}} e(\underline{x}(u), u)^{-1} \underbrace{\nabla_u e(\underline{x}(u), u) \underbrace{\delta u}_{\in U}}_{\in Y}}_{\in X}.$$

So, it is correct that $\underline{x}'(u)\delta u$ is in the space X, because that's also shown in the inner product of equation (3.17). Now a substitution for this term can be applied, which results into:

$$\begin{aligned}
\langle \nabla_u \hat{J}(u)\ ,\ \delta u\rangle_U &= \langle \nabla_{\underline{x}} J(\underline{x}(u), u)\ ,\ -\nabla_{\underline{x}} e(\underline{x}(u), u)^{-1} \nabla_u e(\underline{x}(u), u)\delta u\rangle_X + \langle \nabla_u J(\underline{x}(u), u)\ ,\ \delta u\rangle_U \\
&= -\langle \nabla_{\underline{x}} e(\underline{x}(u), u)^{-*} \nabla_{\underline{x}} J(\underline{x}(u), u)\ ,\ \nabla_u e(\underline{x}(u), u)\delta u\rangle_Y + \langle \nabla_u J(\underline{x}(u), u)\ ,\ \delta u\rangle_U \\
&= -\langle \nabla_u e(\underline{x}(u), u)^* \nabla_{\underline{x}} e(\underline{x}(u), u)^{-*} \nabla_{\underline{x}} J(\underline{x}(u), u)\ ,\ \delta u\rangle_U + \langle \nabla_u J(\underline{x}(u), u)\ ,\ \delta u\rangle_U.
\end{aligned}$$

The definition of the Lagrange multipliers $\underline{\Lambda} = (\underline{\lambda}_1, \underline{\lambda}_2)$

$$\underline{\Lambda} = -\nabla_{\underline{x}} e(\underline{x}(u), u)^{-*} \nabla_{\underline{x}} J(\underline{x}(u), u) \in Y, \tag{3.21}$$

results in:

$$\langle \nabla_u \hat{J}(u)\ ,\ \delta u\rangle_U = \langle \nabla_u e(\underline{x}(u), u)^* \underline{\Lambda} + \nabla_u J(\underline{x}(u), u)\ ,\ \delta u\rangle_U.$$

One can notice that this equation is the same as the one in (3.17):

$$\begin{aligned}
\langle \nabla_u L(\underline{x}, u, \underline{\Lambda})\ ,\ \delta u\rangle_U &= \langle \nabla_u J(\underline{x}, u)\ ,\ \delta u\rangle_U + \langle \nabla_u e(\underline{x}, u)\delta u\ ,\ \underline{\Lambda}\rangle_Y \\
&= \langle \nabla_u J(\underline{x}, u)\ ,\ \delta u\rangle_U + \langle \nabla_u e(\underline{x}, u)^* \underline{\Lambda}\ ,\ \delta u\rangle_U \\
&= \langle \nabla_u \hat{J}(u)\ ,\ \delta u\rangle_U = 0 \qquad\qquad \forall \delta u \in U.
\end{aligned}$$

So, with use of the previous section the gradient of the reduced cost is described with:

$$\langle \nabla_u \hat{J}(u)\ ,\ \delta u\rangle_U = \langle 2\kappa_c^2 u - \underline{\beta}^T \underline{\lambda}_1\ ,\ \delta u\rangle_U \qquad \forall \delta u \in U. \tag{3.22}$$

With use of (3.21) one can also confirm that (3.11) holds:

$$\nabla_{\underline{x}} J(\underline{x}(u), u) + \nabla_{\underline{x}} e(\underline{x}(u), u)^* \underline{\Lambda} = \nabla_{\underline{x}} L(\underline{x}(u), u, \underline{\Lambda}) = 0.$$

Steepest descent method

The total cost should be minimized by changing the input function u of the system. To achieve this lowest cost, the steepest descent method (SDM) is applied. With this algorithm the nearest local minimum of a function, which is the cost $\hat{J}(u)$ in this situation, can be found. The update equation using this SDM for the input function u is described with:

$$u^{k+1} = u^k - s\nabla_u \hat{J}(u^k) \quad \text{for } k \geq 0. \tag{3.23}$$

In this formula, the gradient of the reduced cost function with respect to the input function u has to be known. In our situation this function is calculated in (3.22). The variable s is chosen in such a way that the cost function decreases sufficiently. The value for s is found using a backtracking method. When the system is simulated with this new value for u, the cost should be decreased. With a backtracking method, a good value for s in (3.23) has to be determined, so the cost function decreases. The backtracking method in this situation uses the Armijo rule: With use of the new calculated input function u^{k+1} the new cost $\hat{J}(u^{k+1})$ is calculated, which should be smaller or equal to the old cost function $\hat{J}(u^k)$ minus $sc||\nabla_u \hat{J}(u^k)||_U^2$, where c has a small value (around 10^{-3} or 10^{-4}):

$$\hat{J}(u^k - s\nabla_u \hat{J}(u^k)) \leq \hat{J}(u^k) - \underbrace{sc||\nabla_u \hat{J}(u^k)||_U^2}_{\geq 0} \quad \text{with} \quad s \geq 0. \tag{3.24}$$

The subtraction of this term in (3.24) gives a sufficient decrease which ensures convergence. This equation can be rewritten as a function of s only: $\varphi(s) = \hat{J}(u^k - s\nabla_u \hat{J}(u^k))$.

This results in:

$$\begin{aligned}\varphi(s) &= \hat{J}(u^k - s\nabla_u \hat{J}(u^k)) \\ \varphi(0) &= \hat{J}(u^k) \\ \varphi'(0) &= -\langle \nabla_u \hat{J}(u^k - s\nabla_u \hat{J}(u^k)) \ , \ \nabla_u \hat{J}(u^k)\rangle_U|_{s=0} \\ &= -||\nabla_u \hat{J}(u^k)||_U^2.\end{aligned}$$

With this introduced function $\varphi(s)$, equation (3.24) can be rewritten as:

$$\varphi(s) \leq \varphi(0) + sc\varphi'(0) = l(s) \qquad s \geq 0. \tag{3.25}$$

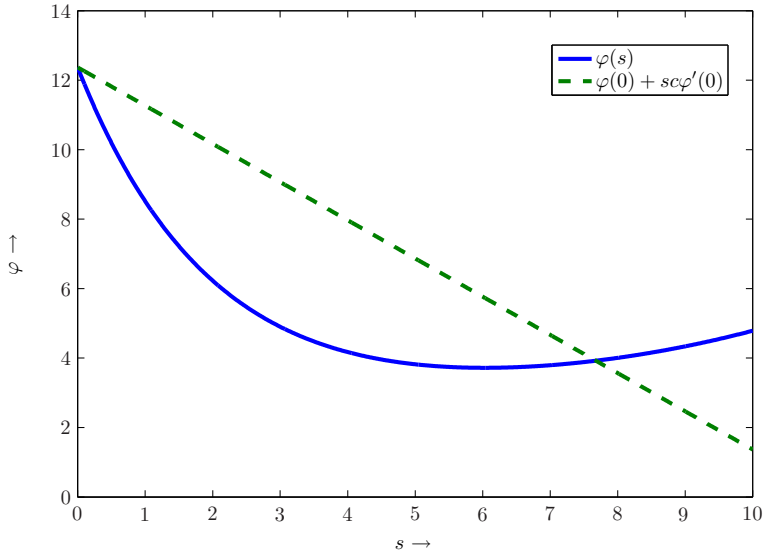


Figure 3.1: Visualisation backtracking (using the Armijo rule)

Inequality (3.25) is visualized in Figure 3.1. There exists a $\delta > 0$ such that $\varphi(s) < l(s)$ for $s \in (0, \delta)$. In Figure 3.1 we find $\delta \in (7, 8)$. However, if the value for s is taken too large, it is possible that the condition described in the equations (3.24) and (3.25) does not hold anymore (for example $s = 10$ in Figure 3.1). If this is the case, the value of s will be divided by two till those equations hold.

Bounded steepest descent method (gradient projection method)

With the previous steepest descent method, the value for the system input u can become very large if the parameters in the cost function are chosen on a wrong way. That is a reason to use the gradient projection method [6] instead, which ensures that the output of the controller is bounded by some upper- and lower-bound values. So, the values for $u(t)$ have to be between two bounds (a and b): $a \leq u(t) \leq b$ for $t \in (0, T)$. The realisation of this bounded version consists of a projection and a change in the rule used in the backtracking method (which was Armijo's rule). The projection of $u(t)$ to the bounded version $\hat{u}(t)$ at each time moment t in the controller output can be written as:

$$\hat{u}(t) = P(u(t)) = \begin{cases} a & : \quad u(t) \leq a, \\ b & : \quad u(t) \geq b, \\ u(t) & : \quad \text{otherwise.} \end{cases} \quad , \quad t \in (0, T).$$

As already mentioned, the equation for the backtracking algorithm has to be changed. The term in the Armijo rule which is directly related with the value of s is the norm of the (reduced) cost function $||\nabla_u \hat{J}(\hat{u}^k)||_U^2$, which goes to a value of 0 when the solution is optimal. With the bounded versions of the controller's output, this value is not always able to decrease to zero anymore. That's why this norm is replaced with the norm of $(\hat{u}^k - \hat{u}^{k-1})$, which results in the formula:

$$\hat{J}(\hat{u}^{k+1}(s)) \leq \hat{J}(\hat{u}^k) - sc||\hat{u}^k - \hat{u}^{k-1}||_U^2 \quad \text{with} \quad \hat{u}^{k+1}(s) = P(\hat{u}^k - s\nabla_u \hat{J}(\hat{u}^k)) \text{ and } s \geq 0. \tag{3.26}$$

Stopping condition of minimization

When the value for $||\nabla_u \hat{J}(u^k)||_U^2$, in iteration k is very small, the input function u is a candidate for the optimal solution. This is the main stopping condition that is used. However, also a relative stopping criteria and a maximum number of iterations are used in the simulations. If the gradient is not decreasing anymore because of the boundaries applied to the output of the controller, the minimization also has to stop. Therefore a stopping criteria using the norm of the difference of the controller output is also used as stopping criteria. This results in the following set of four stopping criteria:

$$\begin{array}{ll} ||\nabla_u \hat{J}(\hat{u}^k)||_U^2 < \epsilon_{abs} & \text{where } \epsilon_{abs} > 0 \text{ has a small value.} \\ \dfrac{||\nabla_u \hat{J}(\hat{u}^k)||_U^2}{||\nabla_u \hat{J}(u^0)||_U^2} < \epsilon_{rel} & \text{where } 0 < \epsilon_{rel} \leq \epsilon_{abs} \text{ has a small value.} \\ k = k_{max} & \text{where } k_{max} \text{ is a fixed number of iterations.} \\ ||\hat{u}^k - \hat{u}^{k-1}||_U^2 < \epsilon_u & \text{where } \epsilon_u \geq 0 \text{ has a small value.} \end{array} \tag{3.27}$$

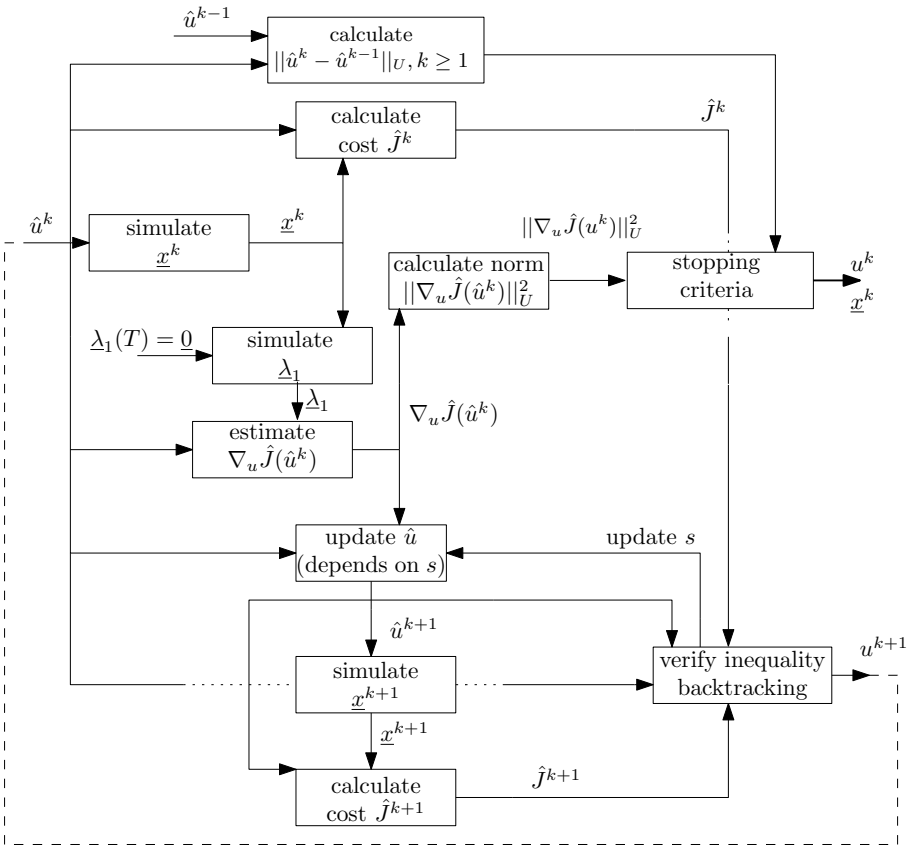


Figure 3.2: Flowchart of open loop receding horizon controller

3.7 Summary of the complete algorithm

Now all steps have been discussed in previous subsections, a short overview of the complete procedure can be created. This is also depicted in the flow chart in Figure 3.2, which is repeated for each time interval.

1. First of all, some starting value for the input u^0 (assumed to be between the bounds defined in the gradient projection method) and some initial conditions $\underline{x}_0$ have to be defined. Those can be taken from the literature or by doing some guesses based on simulations and measurements that are done earlier. The iterations are denoted by k, so $k = 0$ now.

2. For the input $\hat{u}^k$ the cardiovascular model for the state equations $\underline{x}^k$ can be solved for the complete time interval (using the initial conditions that are defined).

3. With the simulated values of the system's behaviour, the reduced cost function $\hat{J}^k$ can be estimated. This value is used later on by estimating the variable s in the backtracking algorithm. It's also good to compare those values for $\hat{J}$ during the number of iterations (k), so it can be shown that the cost really decreases.

4. With the previous calculated state variables, the Lagrange multiplier function $\underline{\lambda}_1^k$ can be estimated. Using those values for the Lagrange multipliers, the gradient (with respect to u) of the reduced cost ($\nabla_u \hat{J}(\hat{u}^k)$) and its norm ($||\nabla_u \hat{J}(\hat{u}^k)||_U^2$) can also be computed.

5. If one of the stopping criteria holds (3.27), then a candidate for the optimal solution has been found! The values for u and $\underline{x}$ are stored in this situation, and the second time interval is simulated with the same algorithm. Now, the initial values for the states $\underline{x}^0$ are the last values from the previous simulation.

6. However, if the stopping criteria does not hold, the values for u are updated using:
$\hat{u}^{k+1}(s) = P(\hat{u}^k - s\nabla_u \hat{J}(\hat{u}^k))$

The value of s has to confirm (3.26). If this condition doesn't hold, the value of s is divided by a factor two by the backtracking algorithm. When the condition holds, the new value for $\hat{u}^{k+1}$ is used as initial condition, so $k = k + 1$, and the algorithm starts at item 2. again!

3.8 Note on limits in integrals

In this chapter the integrals, for example in the cost functions and the ones needed in calculating the norms, are taken from 0 till T. This is a general notation that has to hold for this complete time interval. However, in the receding horizon controller those integrals have other limits, because the controller will step through the complete time interval from 0 till T in a multiple number of steps. Instead of integrating from 0 till T, this will be done from t till $t + \Delta T$ in each step.

3.9 Notes on implementation in Matlab

At the end of this chapter, some notes on the implementation of the Matlab scripts are given. Because of some changes in methods during the research-traject, some (crucial) functions of the scripts have been changed. To show and explain why those changes are made, this extra section is added to the report. First the methods of solving the differential equations are mentioned, and because of the change in these methods, also the way of integration has to be discussed.

Solving the sets of differential equations

In the very first start of the project, only the differential equations of the (uncontrolled) model had to be solved. This has been done using the build-in **ode45()** solver of Matlab, which gave nice results.
However, when trying to solve the equations for the adjoint equations in the receding horizon controller, some troubles occurred with this solver. The system became more stiff in some simulations, which however still could be solved using a stiff-solver of Matlab. Because this method was also not very suitable to be used backwards on a proper and fast way, a more simpler method for solving the differential equations was used.
The explicit Euler method is implemented in the scripts, which made the simulations much faster. Now the time step was fixed, which gives a less accurate result then using the non-linear timescale of the ODE solver. However, by taking enough simulations steps per time unit, this doesn't result in problems. The formulas for this way of solving differential equations (forward direction for the states $\underline{x}$ and backward direction for the adjoint equations $\underline{\lambda}$) are given by:

$$\underline{x}^{k+1} = \underline{x}^k + h f_x(\underline{x}^k, u^k) \qquad \text{where } f_x(\cdot) \text{ is the right-hand side of the differential equation for } \underline{\dot{x}}$$
$$\underline{\lambda}^k = \underline{\lambda}^{k+1} - h f_\lambda(\underline{\lambda}^{k+1}, \underline{x}^{k+1}) \qquad \text{where } f_\lambda(\cdot) \text{ is the right-hand side of the differential equation for } \underline{\dot{\lambda}}$$

In those two equations, h is the (fixed) time-step that is chosen for the simulations. In our simulations this should have a maximal value of $\frac{1}{200}$ minute, to assure that the results are of a good quality.

However, this simple Euler method is not accurate enough. With some small extensions, this method can be transformed to a much more accurate algorithm. Instead of using the Euler method, the 4th order Runga Kutta method is implemented for solving the sets of differential equations. The forward direction-method of this Runga Kutta technique is described with the following set of equations, where h is the time interval between two sample points $\underline{x}^k$ and $\underline{x}^{k+1}$:

$$\underline{k}_1 = h f_x\left(\underline{x}^k, u^k\right) \quad , \quad \underline{k}_2 = h f_x\left(\underline{x}^k + \frac{1}{2}\underline{k}_1, \frac{u^k + u^{k+1}}{2}\right)$$
$$\underline{k}_3 = h f_x\left(\underline{x}^k + \frac{1}{2}\underline{k}_2, \frac{u^k + u^{k+1}}{2}\right) \quad , \quad \underline{k}_4 = h f_x\left(\underline{x}^k + \underline{k}_3, u^{k+1}\right)$$

which are combined in the formula: $\underline{x}^{k+1} = \underline{x}^k + \frac{1}{6}\left(\underline{k}_1 + 2(\underline{k}_2 + \underline{k}_3) + \underline{k}_4\right)$

This Runga Kutta method can also be applied to the problem in the backward direction for the adjoint equations. This results in the following set of formulas:

$$\underline{l}_1 = -hf_\lambda\left(\underline{\lambda}^{k+1}, \underline{x}^{k+1}\right) \quad , \quad \underline{l}_2 = -hf_\lambda\left(\underline{\lambda}^{k+1} + \frac{1}{2}\underline{l}_1, \frac{\underline{x}^{k+1} + \underline{x}^k}{2}\right)$$

$$\underline{l}_3 = -hf_\lambda\left(\underline{\lambda}^{k+1} + \frac{1}{2}\underline{l}_2, \frac{\underline{x}^{k+1} + \underline{x}^k}{2}\right) \quad , \quad \underline{l}_4 = -hf_\lambda\left(\underline{\lambda}^{k+1} + \underline{l}_3, \underline{x}^k\right)$$

which are combined in the formula: $\underline{\lambda}^k = \underline{\lambda}^{k+1} + \frac{1}{6}\left(\underline{l}_1 + 2(\underline{l}_2 + \underline{l}_3) + \underline{l}_4\right)$

Calculating norm of discrete vector

In the backtracking method the norm of the gradient has to be calculated. Because the simulation results are numerically estimated at certain time moments, this norm is not equal to the continuous one. That's why this subsection deals with the differences between the continuous norm and the discrete norm that is used in the Matlab scripts later on.

The normal time continuous norm can be described with the following integral (as already mentioned in the beginning of this chapter):

$$||u||_U = \sqrt{\langle u\ ,\ u\rangle_U} = \sqrt{\int_0^T u(t)^2 dt}.$$

In the discrete norm, the function u is described as a vector which consists all values evaluated at certain time moments. This can be described with $\underline{u} = [u_1, u_2, \ldots, u_{nt}]$, where the first time moment evaluated is denoted with the number 1 and the last one is named nt. In this discrete situation, the integral can be solved using the trapezoidal method, which can be described with:

$$||\underline{u}||_{U,d} = \sqrt{\langle \underline{u}\ ,\ \underline{u}\rangle_{U,d}} = \sqrt{\left(\frac{u_1^2}{2} + \sum_{i=2}^{nt-1} u_i^2 + \frac{u_{nt}^2}{2}\right) dt}.$$

One can see that if nt goes to infinity, and so dt goes to zero, the discrete norm converges to the continuous one provided: u is continuous.
In the first version of the controller in Matlab, the implicit Euler method was implemented for solving the sets of differential equations. This is be replaced by a method with a higher order (the Runga-Kutta method that is described in the previous subsection), which requires a higher order method for integration to prevent the loss of information. That is the reason for replacing the trapezoidal method with the higher order Simpson's rule. Simpson's rule can be described, for a discrete function $\underline{u} = [u_1, u_2, \ldots, u_{nt}]$, with the formula:

$$||\underline{u}||_{U,d} = \sqrt{\langle \underline{u}\ ,\ \underline{u}\rangle_{U,d}} = \sqrt{\left(\frac{u_1^2}{3} + \frac{4}{3}\left(u_1^2 + u_3^2 + \cdots + u_{nt-1}^2\right) + \frac{2}{3}\left(u_2^2 + u_4^2 + \cdots + u_{nt}^2\right)\right) dt}.$$

CHAPTER 4

Receding horizon in closed loop situation

Now the receding horizon controller has been introduced in the previous chapter, it can be implemented in a closed loop situation. In this way, the system can be controlled using this regulator, which results in a stabilized system. How this controller is implemented in a closed loop situation is explained in the first section of this chapter. In the next section some simulation results are given. When doing simulations, parameters and time horizons can be chosen, which results in better or worse performance of the closed loop system. Conclusions on those settings are drawn in the last section of this chapter.

4.1 Comparison with loop in real system

In a human being, the heart rate is controlled by the baroreceptor loop. The real process in this loop is not completely clear however. The measurements of the arterial systemic pressure are done by the baroreceptor cells, and are transferred to the medulla by nerves. In the medulla some control action is applied, but this regulation process is not known exactly. After this process in the medulla, the heart rate is controlled by a signal that comes from the medulla and is transported by nerves to the sinusoidal node in the heart.

In our system, this complete loop has to be replaced by the receding horizon controller that is introduced in the previous chapter. This replacement can be depicted with the block schematic in Figure 4.1, where the dashed loop is the original loop in a human being, and the solid loop is the control loop with the receding horizon controller.

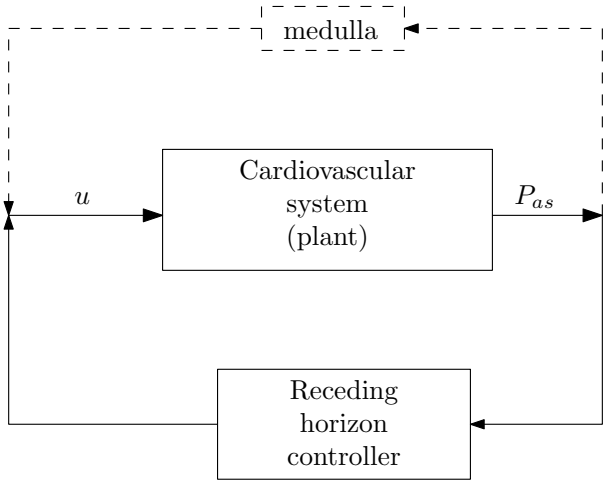


Figure 4.1: Block schematic closed loop system with controller

4.2 Simulation results of closed loop system

With this closed loop system, a lot of different simulations can be done. The parameters of the cardiovascular model are depending on the state of the system (exercise or rest). Those are "fixed" and summarized in Appendix A of this report. The parameters in the (reduced) cost function of the receding horizon controller are not set on a static value, but can be changed. Those changes influence the performance of the control loop. Some choices for those parameters are discussed in the first subsection. Another setting of the receding horizon controller that has large influences on the closed loop behaviour are the durations of the horizon in the controller and the duration of the output of this controller. This is discussed in the second subsection. After that, also the results of the optimisation inside the receding horizon controller are mentioned in a subsection.

4.2.1 Parameters in cost function

In each of the simulations done in this section, the system starts in the so called "rest"-state. So, as equilibrium point $\underline{x}^{eq}$ the values for $\underline{x}^{rest}$ will be chosen. However, if the system starts in this state, there's nothing that needs to be controlled anymore. That's why a "small" deviation (20 mmHg) of the arterial systemic pressure is given at time $t = 0$. At time moment $t = 8$ the patient acts exercising with a workload of $W = 75$ Watt, which results in a change of the equilibrium point to $\underline{x}^{eq} = \underline{x}^{exer}$. The controller should drive the results of the last simulations, where the states are in rest, towards the exercise state values. At $t = 18$ the patient gets back to rest, so the pressure has to decrease towards the rest values. The numerical values of the equilibrium states $\underline{x}^{rest}$ and $\underline{x}^{exer}$ used in the simulations are also given in Appendix A.

To estimate the behaviour of the changes in the parameters in a good way, the lower- and upper-bound in the gradient projection method in the controller are taken very large (positive and negative), so the controller shouldn't have any saturation. In the (reduced) cost function (4.1), there are three parameters that can be adjusted for the best performance of the closed loop system. Also the time duration ΔT of the control interval can be chosen freely. In the results in this subsection the duration of the horizon in the controller and the control interval are chosen the same.

$$J(\underline{x}, u) = \int_{t}^{t+\Delta T} q_{as}^2 \left(P_{as}(t) - P_{as}^{eq}\right)^2 + \kappa_c^2 u(t)^2 dt + \frac{\alpha_c}{2} \left(P_{as}(t+\Delta T) - P_{as}^{eq}\right)^2 \tag{4.1}$$

As mentioned before, the parameters q_{as}, α_c and κ_c are related with each other. If for example κ_c is taken large (with respect to q_{as} and α_c), the input is more penalized, which results in a slow decrease of the error in the pressure P_{as}. An example where this is the case is depicted in Figure 4.2, where the time interval for each iteration in the controller is set to 0.2 minutes and the parameters are chosen as $q_{as} = 0.4$, $\kappa_c = 1.2$ and $\alpha_c = 0.8$. The input u won't become very large, but the convergence of the arterial systemic pressure to its equilibrium value is not satisfying.

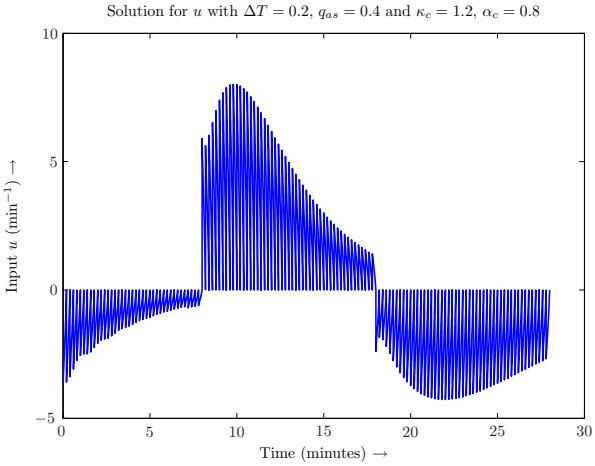
However, if the value for parameter κ_c is decreased a bit (to the new value of $\kappa_c = 0.8$), the results will become much better. Also the range of the input u is not that large. Those results are given in Figure 4.3.

Of course, the parameter κ_c can also be decreased more, which results in large values in the controller output function u. In Figure 4.4 the results are depicted where the parameter $\kappa_c = 0.4$.

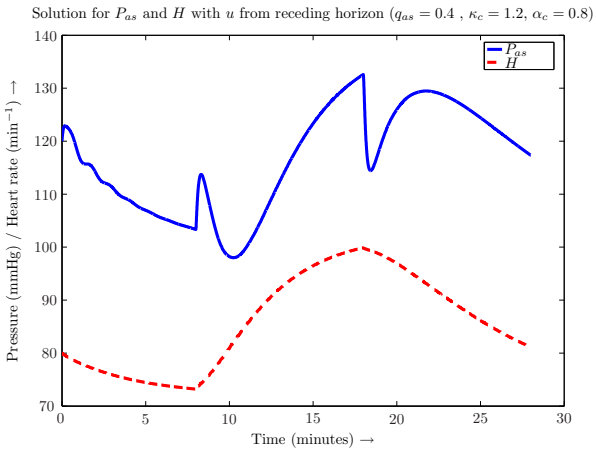
Also the time interval ΔT can be chosen in another way. Other simulations are done using larger time intervals ($\Delta T = 0.5$ and $\Delta T = 1$), which results in the Figures 4.5 till 4.10. If the time interval is increased, the performance of the controller looks better. However, the values of the control output u will increase and the simulation will also take more computation time.

The value for the parameter κ_c can also be taken too small, which results in a high output u (for all values of ΔT) and make the system start acting nervous. This is depicted in Figure 4.11.

In the previous simulations the output of the controller u is not bounded by the receding horizon controller. However, by using a small value for κ_c and using the bounded steepest decent method in the controller, the system will not exceed the defined boundaries. This is shown in Figure 4.12, where the boundaries for the controller output are set to ± 30 bpm.

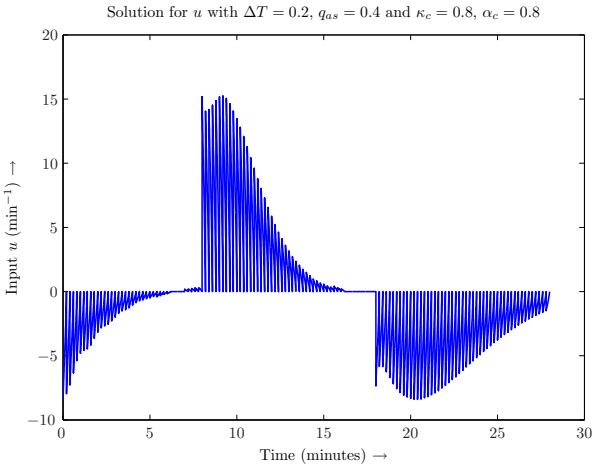


(a) Input u from the controller

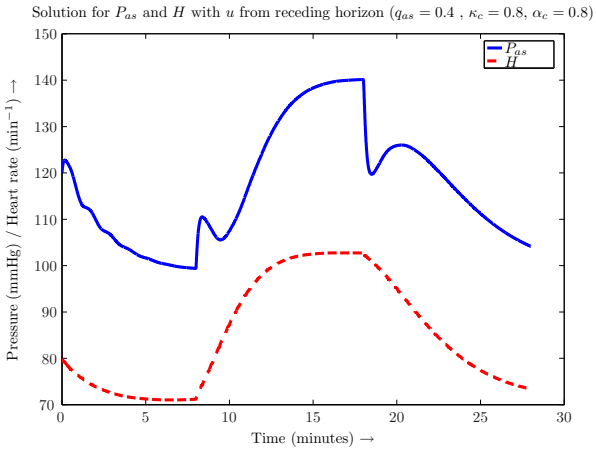


(b) Controlled states P_{as} and H

Figure 4.2: Results with $\Delta T = 0.2$, $q_{as} = 0.4$, $\kappa_c = 1.2$ and $\alpha_c = 0.8$.

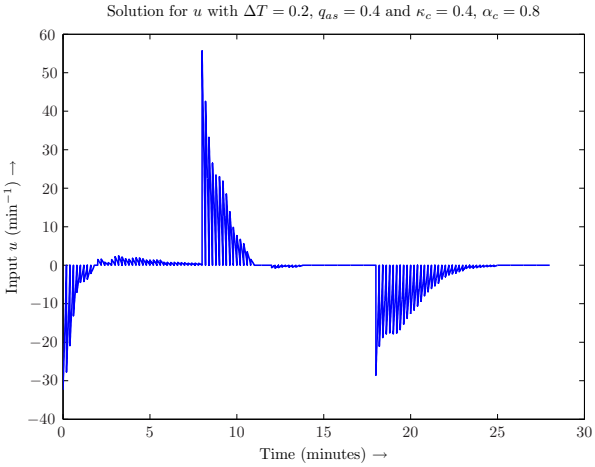


(a) Input u from the controller

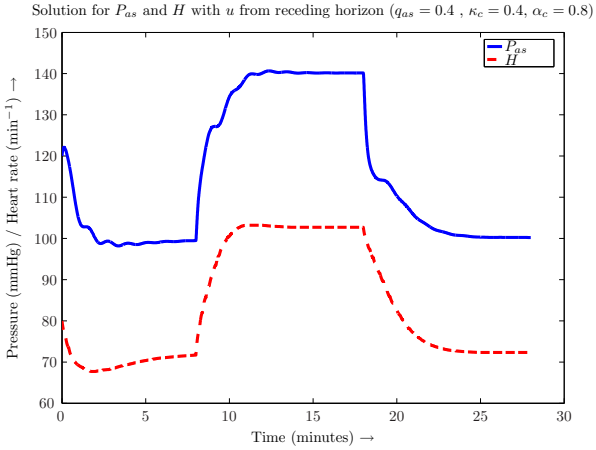


(b) Controlled states P_{as} and H

Figure 4.3: Results with $\Delta T = 0.2$, $q_{as} = 0.4$, $\kappa_c = 0.8$ and $\alpha_c = 0.8$.

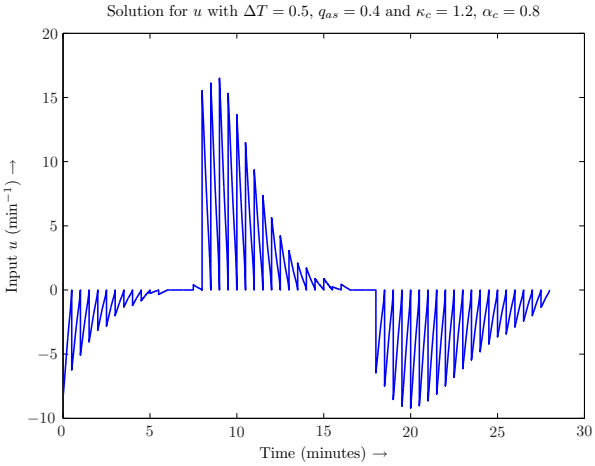


(a) Input u from the controller

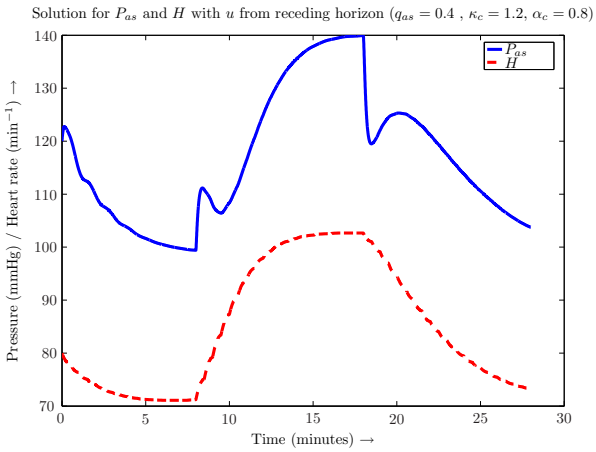


(b) Controlled states P_{as} and H

Figure 4.4: Results with $\Delta T = 0.2$, $q_{as} = 0.4$, $\kappa_c = 0.4$ and $\alpha_c = 0.8$.

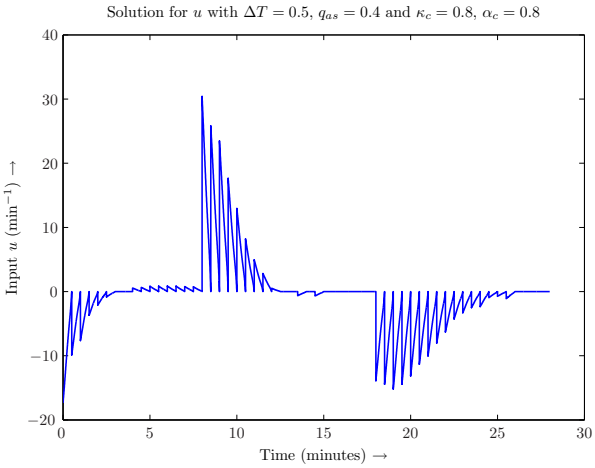


(a) Input u from the controller

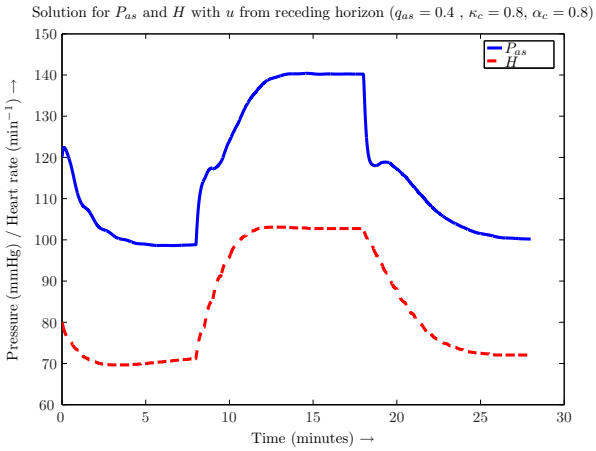


(b) Controlled states P_{as} and H

Figure 4.5: Results with $\Delta T = 0.5$, $q_{as} = 0.4$, $\kappa_c = 1.2$ and $\alpha_c = 0.8$.

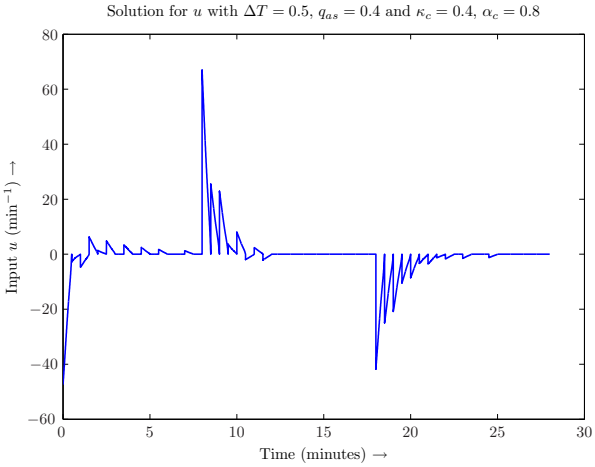


(a) Input u from the controller

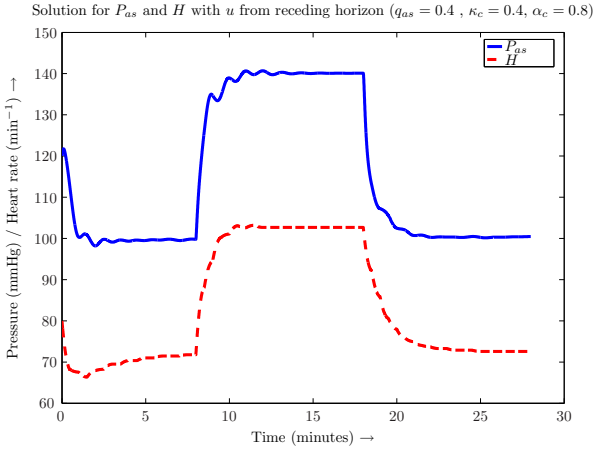


(b) Controlled states P_{as} and H

Figure 4.6: Results with $\Delta T = 0.5$, $q_{as} = 0.4$, $\kappa_c = 0.8$ and $\alpha_c = 0.8$.

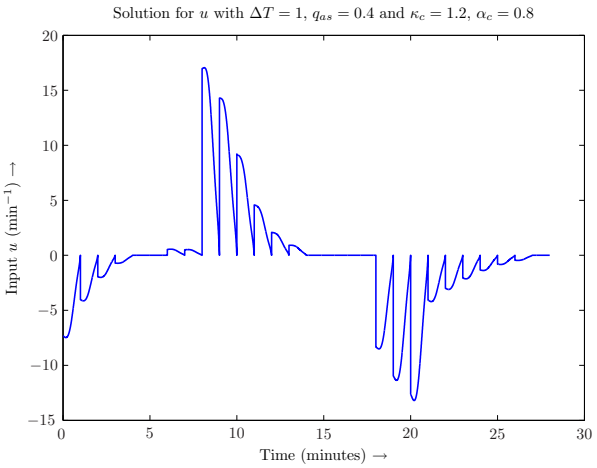


(a) Input u from the controller

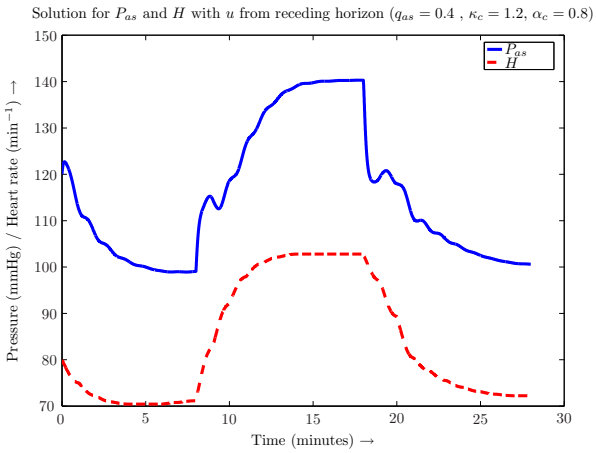


(b) Controlled states P_{as} and H

Figure 4.7: Results with $\Delta T = 0.5$, $q_{as} = 0.4$, $\kappa_c = 0.4$ and $\alpha_c = 0.8$.

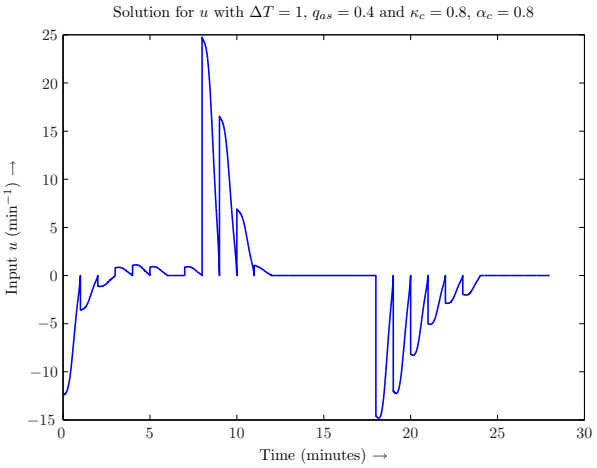


(a) Input u from the controller

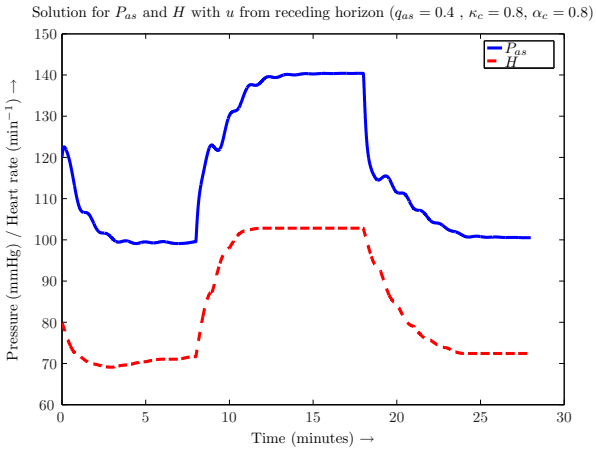


(b) Controlled states P_{as} and H

Figure 4.8: Results with $\Delta T = 1$, $q_{as} = 0.4$, $\kappa_c = 1.2$ and $\alpha_c = 0.8$.

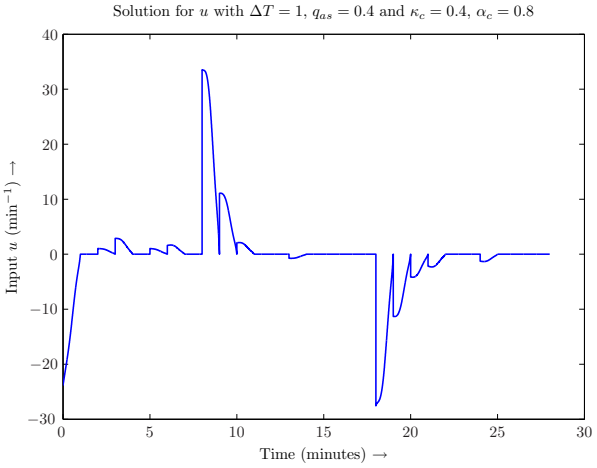


(a) Input u from the controller

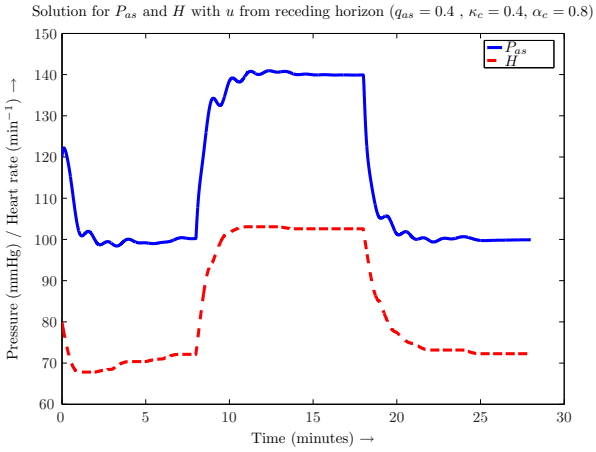


(b) Controlled states P_{as} and H

Figure 4.9: Results with $\Delta T = 1$, $q_{as} = 0.4$, $\kappa_c = 0.8$ and $\alpha_c = 0.8$.



(a) Input u from the controller



(b) Controlled states P_{as} and H

Figure 4.10: Results with $\Delta T = 1$, $q_{as} = 0.4$, $\kappa_c = 0.4$ and $\alpha_c = 0.8$.

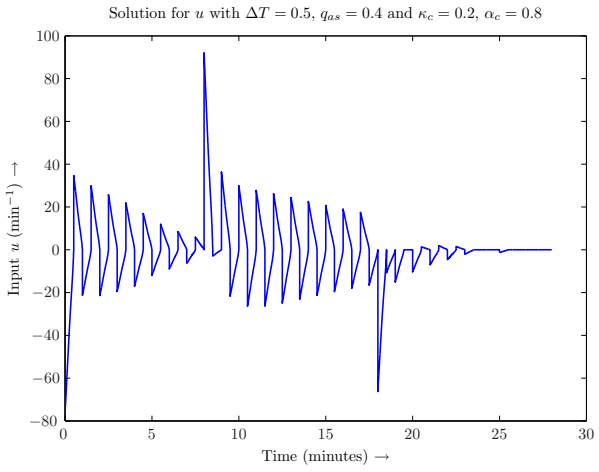
ΔT	κ_c	$\int_0^T q_{as}^2(P_{as}(t) - P_{as}^{eq}(t))^2 dt$	$\int_0^T \kappa_c^2 u(t)^2 dt$
0.2	0.4	2299	245
	0.8	1193	282
	1.2	268	196
0.5	0.4	1172	608
	0.8	478	500
	1.2	162	291
1	0.4	611	867
	0.8	343	532
	1.2	178	218

Table 4.1: Total error in simulations and parts of the cost function.

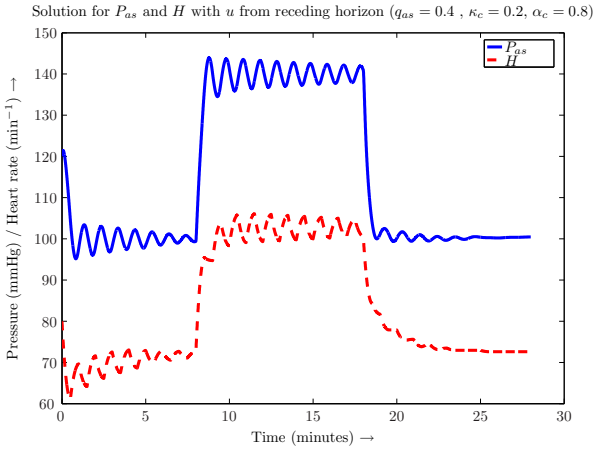
At the end of this subsection, a table with values for:

$$\int_0^T q_{as}^2(P_{as}(t) - P_{as}^{eq}(t))^2 dt \quad \text{and} \quad \int_0^T \kappa_c^2 u(t)^2 dt$$

are given with fixed values for q_{as} (0.4) and α_c (0.8) (Table 4.1). The values of both integrals in those table should be of the same order, because both the error of the arterial systolic pressure and the output of the controller should contribute with the same effort on the complete cost.
This is the case in the last simulation ($\kappa_c = 1.2$) with $\Delta T = 0.2$. However, the maximal value of the controller output is too large, so this is not a good solution.
If the time interval is set to $\Delta T = 0.5$, simulating using the parameter $\kappa_c = 0.8$ gives good results and also the output of the controller is not very large.
When using a larger time interval of $\Delta T = 1$, $\kappa_c = 1.2$ is a parameter that can be used for a good controller. This time interval is quite large, where a lot of changes in the complete system can occur during a single control iteration in the closed loop situation.

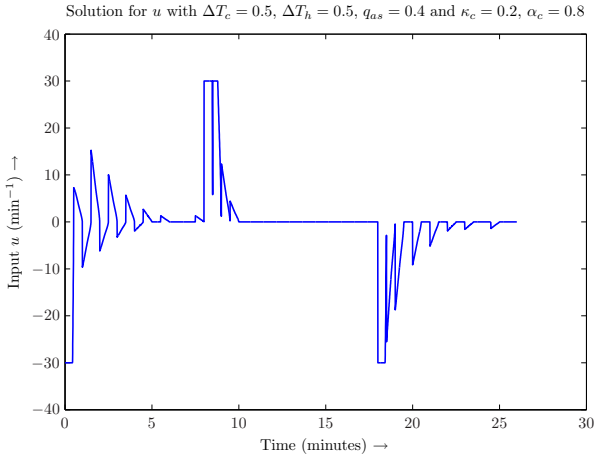


(a) Input u from the controller

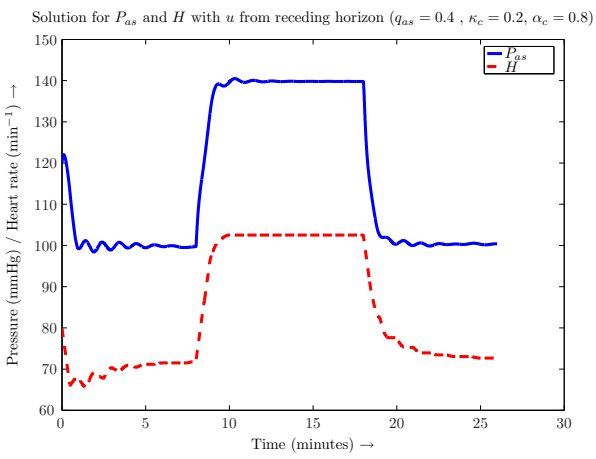


(b) Controlled states P_{as} and H

Figure 4.11: Results with $\Delta T = 0.5$, $q_{as} = 0.4$, $\kappa_c = 0.2$ and $\alpha_c = 0.8$, so κ_c is too small!



(a) Input u from the controller



(b) Controlled states P_{as} and H

Figure 4.12: Results with $\Delta T = 0.5$, $q_{as} = 0.4$, $\kappa_c = 0.2$ and $\alpha_c = 0.8$, where u is bounded!

4.2.2 Control and horizon time intervals

In the previous simulation results, there was only one time interval ΔT specified, which is used in the receding horizon controller and also is used for the interval to control the system. This doesn't have to be the case. In the block diagram in Figure 4.13 an overview of the in- and output of the controller are given. The lengths of the vectors used in the simulations with Matlab are mentioned in this schematic.

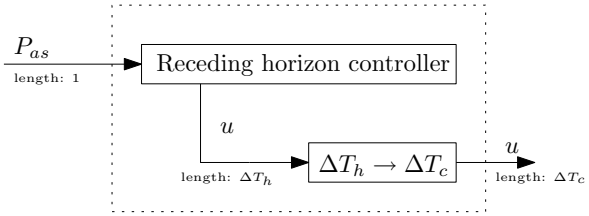


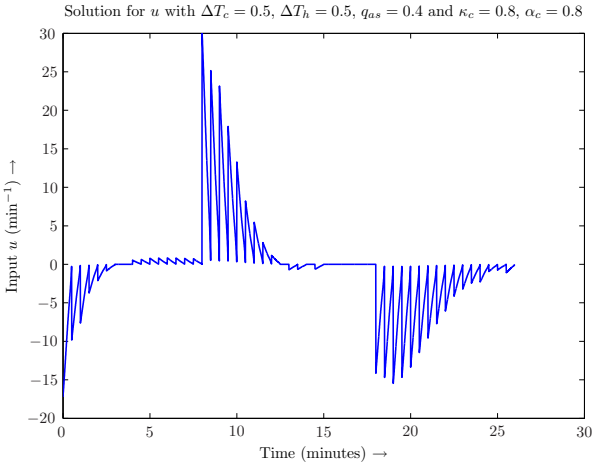
Figure 4.13: Lengths of arrays (time) in Matlab simulations.

At a certain time moment, the value for the arterial systemic pressure is measured and used as input for the receding horizon controller. This controller starts using this input as initial value for its model, and starts to predict the behaviour of the system for a certain horizon (ΔT_h). For this horizon, the best control output (also of the same length) is estimated which should drive the system to the desired equilibrium state. This controller output is fed back to the real system, but the length of this array doesn't have to be as long as the time horizon in the controller. This time interval is denoted with ΔT_c which is less or equal to the time-horizon in the controller (ΔT_h).

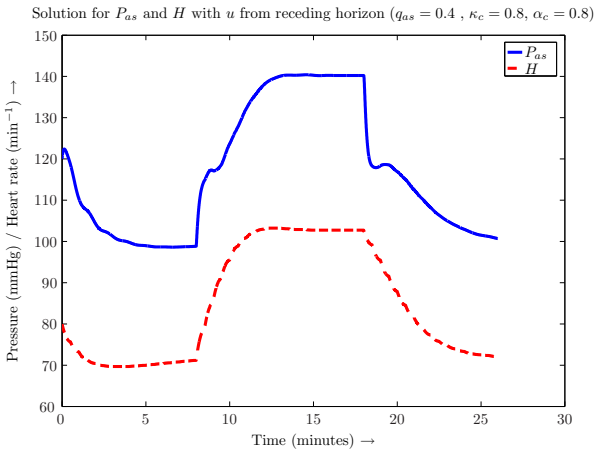
In the following simulations, the same scenario is used as in the previous subsection (small deviation, exercise state, rest state), however the output of the controller u is now limited by the boundaries $-30 \leq u \leq 30$. The values for the parameters in the cost function are now fixed: $q_{as} = 1, \kappa_c = 1$ and $\alpha_c = 1$. With those simulation settings, the following three cases are compared: $\Delta T_h = \Delta T_c = 0.5$, $\Delta T_h = 1.5, \Delta T_c = 0.75$ and $\Delta T_h = 2, \Delta T_c = 1$. The results for the output u and the corresponding output of the states (P_{as} and H) are drawn in Figures 4.14 till 4.16. One can notice that the controller output doesn't converge to zero, after each time step ΔT_c if this time interval is not equal to the horizon time interval ΔT_h, because in those situations this convergence is not in this time interval anymore.

One can see that the behaviour of the closed loop system improves if $\Delta T_h = 2\Delta T_c$, however if the horizon time interval increases more the result becomes worse. That occurs because of the truncation of useful controller output data from the receding horizon controller to the input of the system. It also cost more computation time if the horizon increases, because the computer needs to estimate more time samples of the differential equations.

The numerical results of the simulations with the time intervals $\Delta T_h = 0.5$ and $\Delta T_h = 1$ are given in Table 4.2. One can see that the latter has a smaller error, with almost the same amount of controller output as in the first case.

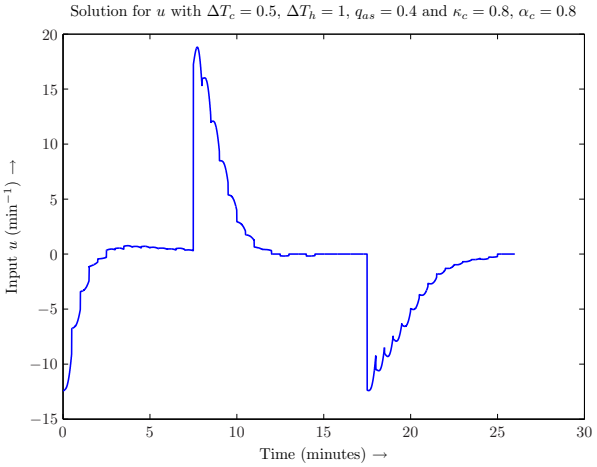


(a) Input u from the controller

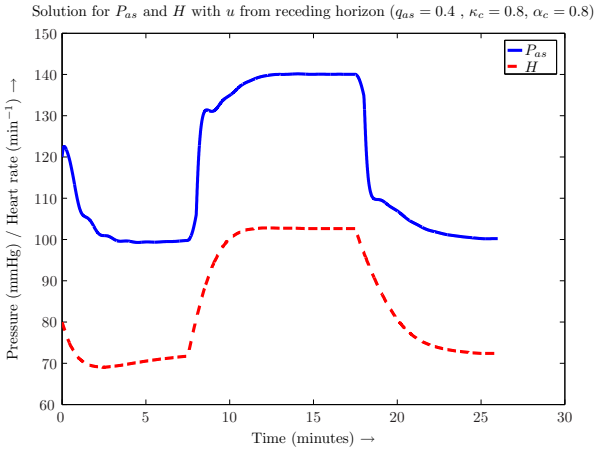


(b) Controlled states P_{as} and H

Figure 4.14: Results with $\Delta T_c = \Delta T_h = 0.5$.

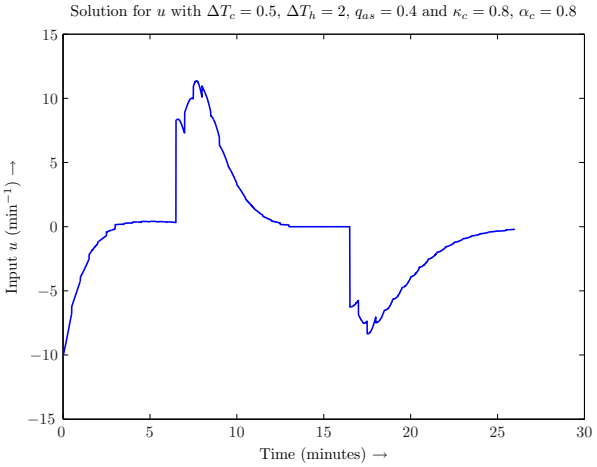


(a) Input u from the controller

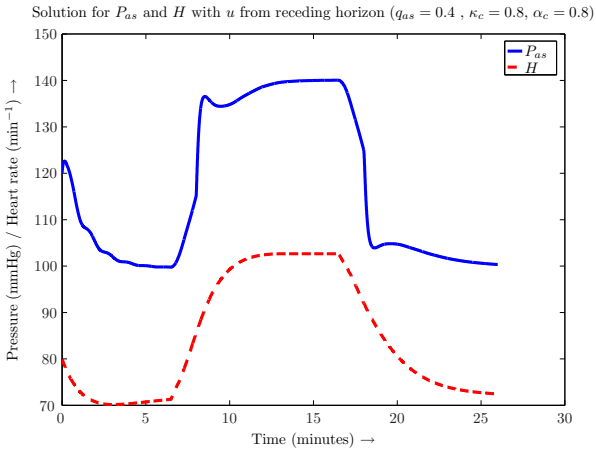


(b) Controlled states P_{as} and H

Figure 4.15: Results with $\Delta T_c = 0.5$ and $\Delta T_h = 1$.



(a) Input u from the controller



(b) Controlled states P_{as} and H

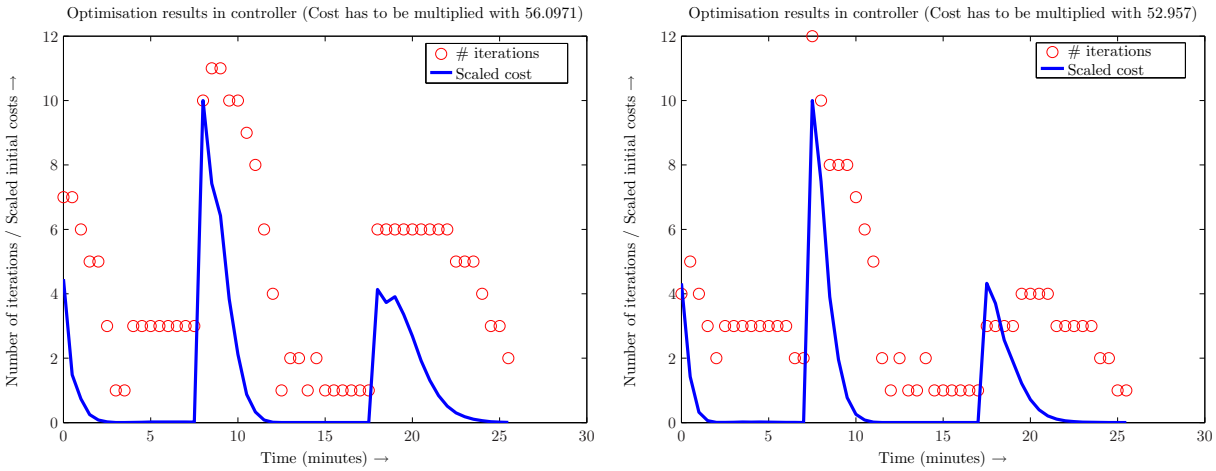
Figure 4.16: Results with $\Delta T_c = 0.5$ and $\Delta T_h = 2$.

ΔT_c	ΔT_h	$\int_0^T q_{as}^2(P_{as}(t) - P_{as}^{eq}(t))^2 dt$	$\int_0^T \kappa_c^2 u(t)^2 dt$
0.5	0.5	481	497
0.5	1	161	460

Table 4.2: Numerical values for cost using different time intervals.

4.2.3 Optimisation results in controller

Another interesting aspect of the simulation results is the number of iterations that is needed in the receding horizon controller for estimating the output u. In this subsection those statistics are depicted for two different time horizons in the controller ($\Delta T_h = 0.5$ and $\Delta T_h = 1$). The results for those two simulations are depicted in Figure 4.17. Also a (scaled) version of the initial cost at the beginning of each new iteration in the controller is shown. In this way the decrease of the cost function can be compared with the number of iterations that are used. As you expect, the number of iterations will decrease if the cost becomes smaller.
In the second subplot of Figure 4.17, the error decreases faster. Also the number of iterations in the controller decreased, however one have to notice that each iteration takes (around) the double computation time because of the larger horizon. However, the latter simulation gives better results (with a bit more computation time).



(a) $\Delta T_h = \Delta T_c = 0.5$ (b) $\Delta T_h = 2\Delta T_c = 1$

Figure 4.17: Statistics of the receding horizon controller.

Parameter / Time-interval	Value	Main influence
q_{as}	0.4	Suppression of total error in P_{as}
κ_c	0.8	Limit the controller output u
α_c	0.8	Stability issues
ΔT_c	0.5 minutes	Fast simulations
ΔT_h	1 minute	Better performance

Table 4.3: Simulation settings for a good behaviour.

4.3 Conclusions on simulation results

The behaviour and performance of the closed loop system are depending on both the parameters which are present in the cost function, the time interval of the control output and the prediction horizon in the controller itself.

The value for the parameter q_{as} determines the total error of the controller between the desired value of P_{as}^{eq} and the values of P_{as} out of the system. The value for the cost-function-parameter α_c determines the stability of the solution, so this value should not be taken too small with respect to the other parameters. The parameter κ_c penalises the output of the controller u, which should not be too large (in case if there are no boundaries). So, the value of this parameter should not be taken too small. Also in the case if the output is bounded in the SDM, this value should not be too small so that the system doesn't start behaving nervous. Quite good values for the parameters that are used in this chapter are: $q_{as} = 0.4$, $\kappa_c = 0.8$ and $\alpha_c = 0.8$. In chapter 6 those parameters are estimated using a parameter estimation with clinical measurement data. In that case, parameters will be found that really match reality.

The value of the control interval should not be too small, because then the performance of the controller is not that good. If it's made larger, the performance but also the computation time will increase! In the simulations done in this chapter, a good value of ΔT_c is around 0.5 minutes. Also taking different intervals for the horizon in the controller results in better results. The value for the horizon should not be much larger than the value of the control output, because this will take a lot of computation time and also won't result in a very nice behaviour. A value that seems to give good performance is $\Delta T_h = 1$.

All those parameters and time intervals are summarized in Table 4.3.

CHAPTER 5

Noise sensitivity of closed loop system

The real system, including the control loop with the medulla, in a human body is very complicated. The signals are transported using nerves, which also transmit information about other processes in the human body. A phenomena that is called cross-talk occurs in a lot in normal communication lines that are close to each other. This principle can also occur in the real human being of course.

This is just one example of noise which is added to signals that are travelling through the human body. Of course there can be a long list of other things that happen in the body that influence some communication in nerves. Therefore, it might be a good idea to investigate how the receding horizon controller, which is implemented in the closed loop system, deals with noise.

In this chapter additions of noises on two different places in the closed loop system are investigated. Some noise will be added on the output of the receding horizon controller, which normally comes from the medulla, and on the output of the complete cardiovascular system that is interested for the baroreceptor loop, the arterial systemic pressure.

5.1 Noise on controller output and system output

As already mentioned in the short introduction of this chapter, two different noises will be added to the complete system. Those disturbances are white noise sources and are denoted with the symbols η_u and η_P, which are depicted in the block diagram of Figure 5.1.

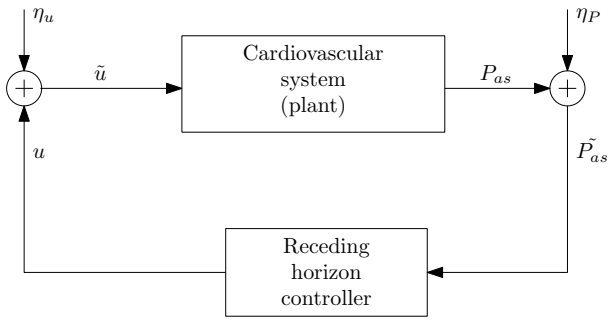


Figure 5.1: Block schematic of closed loop system with noise.

As shown in the figure above, the new input of the system is $\tilde{u} = u + \eta_u$ and the input for the receding horizon controller is replaced by $\tilde{P_{as}} = P_{as} + \eta_P$. The two noise sources are generated by a random noise generator, which is available in Matlab. Normally this noise has an amplitude between 0 and 1, but in our situation it is converted to $\pm\sigma_u$ and $\pm\sigma_P$, which are the amplitudes (positive and negative) for the controller output u and the system output P_{as} noises.

In the first simulations, only one noise source should be enabled at the same time, so the influence of that specific noise on the closed loop system can be investigated.

Ampl. noise on u (bpm)	Ampl. noise on P_{as} (mmHg)	$\int_0^T q_{as}^2(P_{as}(t) - P_{as}^{eq}(t))^2 dt$	$\int_0^T \kappa_c^2 u(t)^2 dt$
0	0	161	460
1	0	161	467
3	0	161	508
0	3	187	466
0	6	205	435
3	6	224	510

Table 5.1: Influence of noise on the cost function.

Noise on the controller output u

In the following simulations the parameters and time-intervals are chosen as in Table 4.3 of the previous chapter. In Figure 5.2 till Figure 5.4 three simulation results are depicted: one without the addition of noise (Figure 5.2), one with a noise addition of ± 1 min^{-1} (Figure 5.3) and in the last one a noise of ± 3 min^{-1} has been added to the amplitude of the controller output u (Figure 5.4). Those values are chosen in this way, because this is 10% of the boundary values set in the bounded steepest descent method.

From the simulations it can be seen that there is an influence on the behaviour of the closed loop system, however the receding horizon controller is able to regulate with those noise additions. So, the receding horizon controller can deal with noise on its output, which is fed to the cardiovascular system.

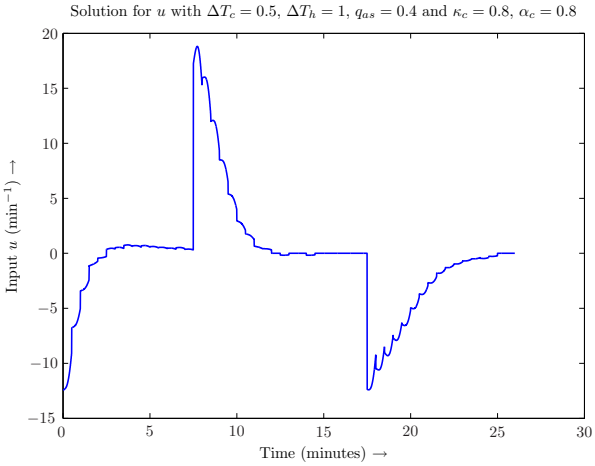
Noise on the system output P_{as}

In the following simulations a noise is added to the output of the cardiovascular system. One should expect that this has some larger influence on the system, because only one sample of this output is used as reference in the receding horizon controller. Before starting those simulations, some possible extensions of the controller were thought out. Taking into account the previous predicted values for the system or averaging some more output samples from the system were options for this extension, but they are not needed at all! The system is able to deal with, relative small, noises on the output. In the Figures 5.5 and 5.6 two random noises with amplitudes of 3 mmHg and 6 mmHg are added to the output and fed back to the controller. It takes some more time to stabilize the system to its desired equilibrium states, but the results are good!

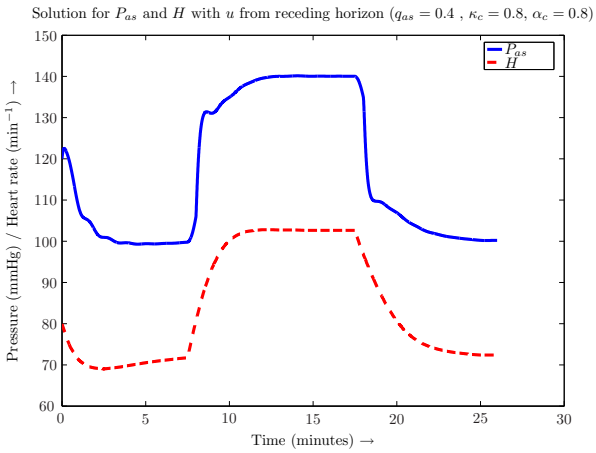
Noise on both outputs u and P_{as}

If both noises are enabled, the closed loop system still looks stabilized, but it is not that good anymore. In the situation depicted in Figure 5.7 both noise sources are enabled at the previous maximal values (± 3 bpm and ± 6 mmHg). This worse behaviour is also visible in the numerical results.

The result of all those simulations are also given in an overview in Table 5.1. One can see that the integral over the error in u increases when noise is added to the controller output. This doesn't influence the value for the error in P_{as} during the complete interval from 0 till T. When noise is added to the output of the cardiovascular system, this influences the other integral in the table (error in P_{as}). There's also some influence on the value for the integrated value of the controller output u, because a change in the input for the controller (P_{as}) implies a control action, which results in a change of u. If both noises are enabled this results, as already mentioned, in worse (numerical) results.

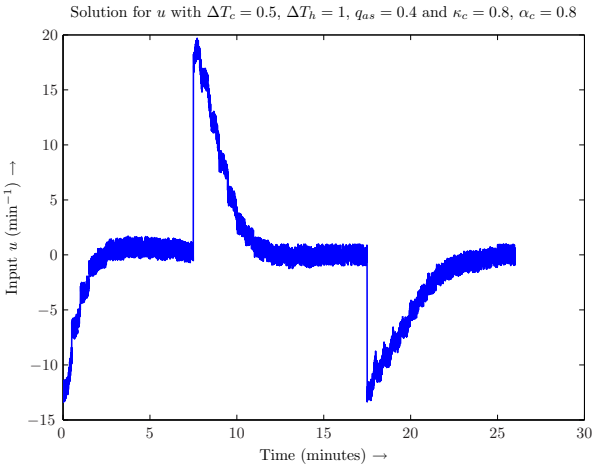


(a) Input u from the controller

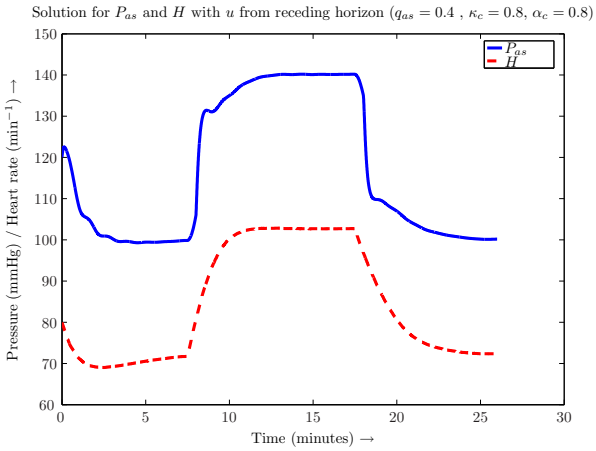


(b) Controlled states P_{as} and H

Figure 5.2: No addition of noise to the controller output u.

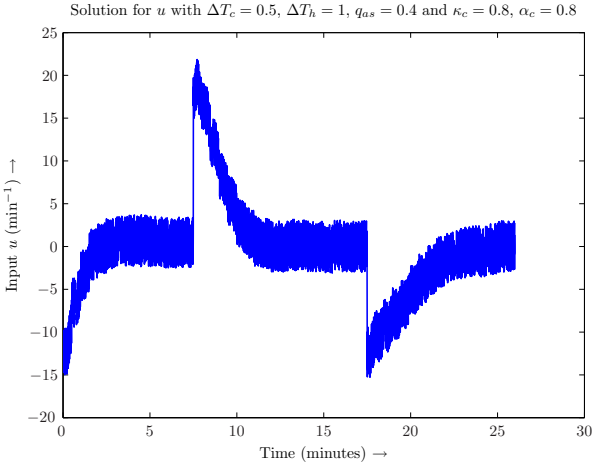


(a) Input u from the controller

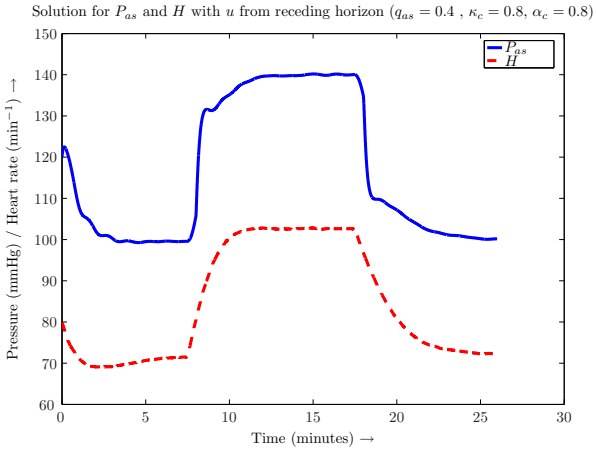


(b) Controlled states P_{as} and H

Figure 5.3: Addition of noise to the controller output u with amplitude of ± 1 min^{-1}.

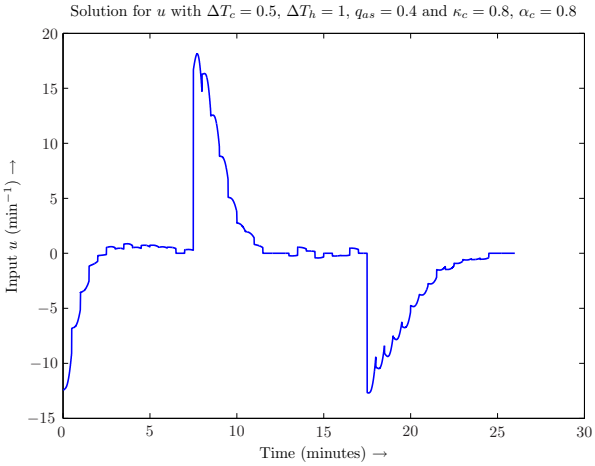


(a) Input u from the controller

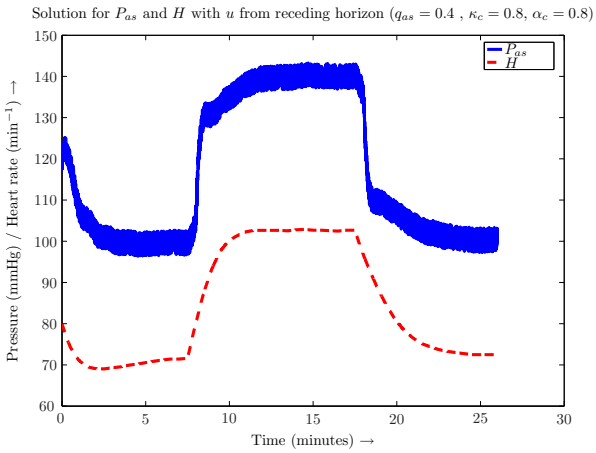


(b) Controlled states P_{as} and H

Figure 5.4: Addition of noise to the controller output u with amplitude of ± 3 min^{-1}.

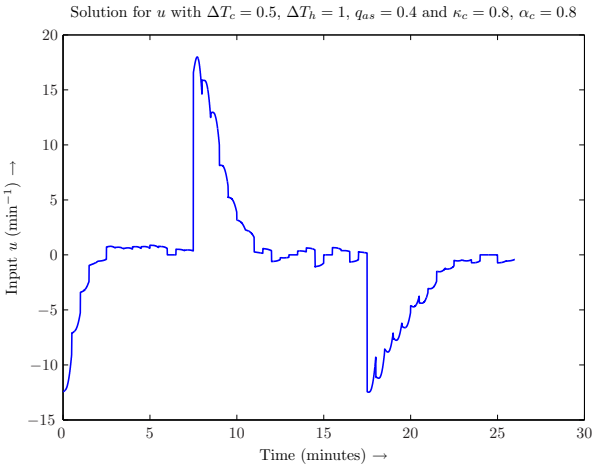


(a) Input u from the controller

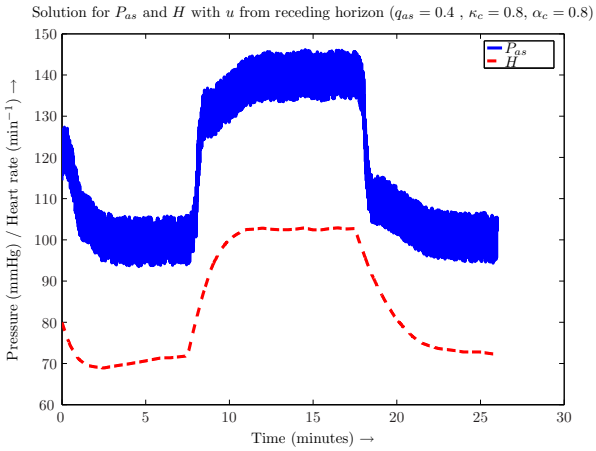


(b) Controlled states P_{as} and H

Figure 5.5: Addition of noise to the system output P_{as} with amplitude of ± 3 mmHg.

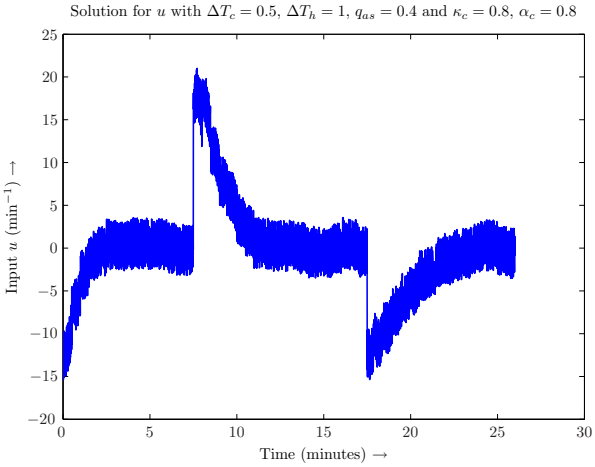


(a) Input u from the controller

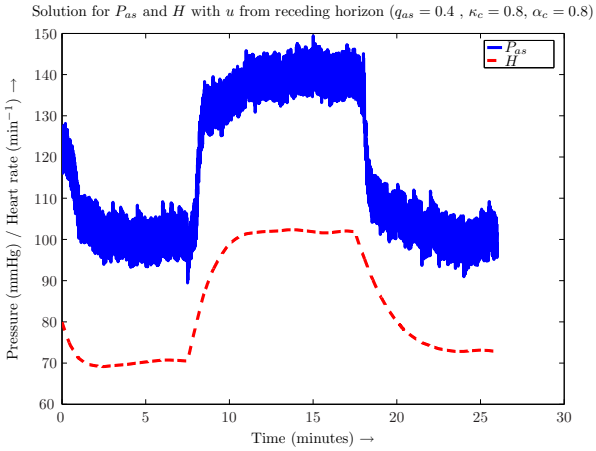


(b) Controlled states P_{as} and H

Figure 5.6: Addition of noise to the system output P_{as} with amplitude of ± 6 mmHg.



(a) Input u from the controller



(b) Controlled states P_{as} and H

Figure 5.7: Addition of noise to both output ($u \pm 3$ min^{-1} and $P_{as} \pm 6$ mmHg.)

CHAPTER 6

Parameter estimation

In this chapter of the report the estimation for the controller parameters will be discussed. The earlier discussed closed loop situation has to be compared with the reality. That is the reason why the parameters are identified using clinical measurement data. To perform a parameter estimation, an optimisation problem has to be defined. This is done in the first section of this chapter. The estimation resulted in some (small) problems, so it has been divided in several steps that are discussed in the second section. The parameters that are used in the model are identified by an estimation using the LQR controller for the baroreceptor loop [1]. Now this controller has been replaced by another type, those parameters should be re-estimated. Because this is very time-consuming, only two important parameters of the model are estimated at the end of this chapter.

6.1 Defining the problem for the parameter estimation

With this parameter estimation, the closed loop system should have the same performance and behaviour as the results from measurements done in real life. To estimate the values of the parameters an optimisation solver in Matlab tries to solve a minimization problem. This problem needs to minimize the difference between the simulation results of the closed loop system and the measurements by adjusting the parameters in the receding horizon controller. Those parameters are the time interval of each control iteration ΔT_c, the time horizon inside the controller ΔT_h and the parameters in the cost function that needs to be minimised in this receding horizon controller (q_{as}, κ_c and α_c). Because the latter time interval mentioned should not be smaller than the value of ΔT_c, this interval can be written as: $\Delta T_h = K_\Delta \Delta T_c$. Those two different time intervals are depicted in Figure 6.1. When taking one as lower-bound of this parameter K_Δ, the problem with a too small time interval inside the controller can not occur. The other parameters should also have some lower- and upper bound values, which have to be defined for the optimisation, e.g.:

$$0.2 \leq \Delta T_c \leq 1, \quad 1 \leq K_\Delta \leq 2.5, \quad 0.01 \leq q_{as} \leq 2, \quad 0.01 \leq \kappa_c \leq 2, \quad 0.01 \leq \alpha_c \leq 2. \tag{6.1}$$

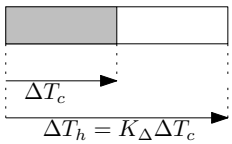


Figure 6.1: Different time intervals ΔT_c and ΔT_h

Knowing the bounds of the parameters, the minimization problem that needs to be solved can be defined:

$$\min_{\Delta T_c, K_\Delta, q_{as}, \kappa_c, \alpha_c} J_{param} = \int_0^T |H^{sim}(t) - H^{meas}(t)| + |P_{as}^{sim}(t) - P_{as}^{meas}(t)| dt \tag{6.2}$$

subject to: (6.1),

where $H^{sim}(t)$ and $P_{as}^{sim}(t)$ are the values for the heart rate and arterial systemic pressure estimated using the closed loop situation described in Chapter 4. $H^{meas}(t)$ and P_{as}^{meas} are measured values of those two states. There are two datasets with measurement results available for this estimation. Those datasets contain information of a transition from the resting state, without workload, to a state with an exercise of 75 Watt. Because the parameters that need to be estimated have the largest influence on the transition from the resting state to the exercise state, the estimation is only performed in this interval instead of the complete measurement data, that also contains a lot of information about the steady states.

There is one large difference between the simulation results in the previous chapters and the one depicted in this chapter. The values of the equilibrium states are different then the ones used before. In the following simulations and parameter estimations the equilibrium states have the values:

$$\underline{x}^{rest} = [105, 4, 15, 6, 60, 0, 4, 0, 25, 80]^T \quad \& \quad \underline{x}^{exer} = [122, 4, 15, 6, 90, 0, 4, 0, 17, 105]^T.$$

6.2 Steps taken during estimation

This section is divided into several parts, because the estimation of all parameters can't be done in just one run. Because of problems with simulations, like several local minima in the gain K_Δ and numerical approximation of the gradient from the time horizon, the estimation of all parameters is divided into the following steps: First of all a starting value for the parameter with the largest influence on the controller output κ_c is determined. After this estimation, the values for the time intervals ΔT_c and ΔT_h, which is rewritten as a multiplication of the variable K_Δ with ΔT_c, have to be estimated. With those two (fixed) time intervals the other unknown parameters in the receding horizon controller can be estimated. Before doing this time-consuming estimation with the measurement dataset, this estimation is being tested using a simulated dataset with known parameters. Two different Matlab-toolboxes, the build-in **fmincon()**-algorithm and the SolvOpt-toolbox [4], are used for those estimations. The results of those parameter estimations of the control parameters only are not that satisfying. So the previous estimation of some parameters in the cardiovascular model [1] has been done over again for some parameters.

6.2.1 Estimating a proper starting value for controller output-related parameter

The most crucial parameter in the cost function in the receding horizon controller is the one related with the controller output u. A proper starting value for this parameter κ_c can be estimated using the **fmincon()** function in Matlab, with defining the values of all other parameters fixed as they have been in previous simulations:

$$[\Delta T_c, K_\Delta, q_{as}, \alpha_c] = [0.5, 2, 0.4, 0.8]$$

İn this simulation, the number of iterations of the optimisation algorithm has been limited to 5, because the value of this parameter does not have to be that exact. Later on it will be adjusted in estimations anyway. The optimiser outputs the following optimisation steps:

```
                                 Max     Line search  Directional  First-order
 Iter F-count        f(x)   constraint   steplength   derivative   optimality Procedure
    0      2      68.0553         -0.6
    1      4      56.4146   5.551e-017            1         5.22         49.2
    2      6      55.3929      -0.1047            1        -3.31         31.6
    3     15      55.0858       -0.114      0.00781        -35.3         37.5  Hessian modified twice
    4     22      55.0181      -0.1181       0.0313        -4.15           32
    5     35      54.9997      -0.1187     0.000488        -37.8         37.7  Hessian modified twice
Maximum number of iterations exceeded;
 increase OPTIONS.MaxIter.
```

When using the parameter's starting value of 0.8 the optimiser results a value of 0.3181, which results in a lower value for the cost function. This value is much smaller then the value used in previous

simulations, because the values of the equilibrium states are changed. The value of the arterial systemic pressure in exercise state is lower in the measurements than the value in simulation before. Now the value of the controller output doesn't need to be that large anymore, because the error of the arterial systemic pressure is smaller.
This estimated value is used in the next parameter estimations and simulations as starting value for the parameter κ_c.

6.2.2 Estimate a value for the control horizon

The first problem that arose during estimation all parameters with the Matlab function **fmincon()** was the estimation of the control interval ΔT_c. The **fmincon()** algorithm tries to make small steps for estimating the gradient, so the optimal value for the parameter can be estimated. When this is applied to the parameter ΔT_c, this results in problems because of the step size of the simulation and the measurement results. This can be clarified with the following example:
If the initial value of the parameter is taken to be 0.5 minutes, the **fmincon()** algorithm tries to estimate the value of the cost function using this initial value and also evaluates the cost function with some small deviation (0.500002 minutes for example). However, the step size of the simulation in the receding horizon controller and the step size in the measurements is fixed to 0.005 minutes (200 samples per minute). So, changing the control interval with a value which is smaller then the step size results in problems using the current scripts that are implemented in Matlab.
Some possible solutions to solve this problem are:

- Decreasing the step size / Increasing the sample rate:
  It is possible to decrease the step size of the simulations in the receding horizon controller, which results in more calculations for the complete simulation of the time interval. To be able to compare the results of the simulations using the receding horizon controller, the values of the measurements have to be interpolated. This interpolation of measurements is undesired, but can be applied of course.

- Rounding the parameter ΔT_c to multiples of the step size:
  Another possibility, which doesn't work however, is rounding the value of the parameter to multiples of the step size. The implementation of this method is very simple. Because the solver makes very small step sizes for estimating the gradient of the parameter, this rounding operation influences those gradient results. Because the value of the deviated parameter is rounded, it will have the same value as the initial parameter (in our example 0.500002 minutes is rounded to 0.500 minutes) and so the value of the gradient is going to 0. So, the solver will return that an optimal value has been estimated.

However, a control interval ΔT_c should not be more accurate then the step size of the measurements, because in a real life application those measurements will be a limiting factor. To estimate a proper value for this time interval, which can be used in the parameter estimation of the other parameters, the cost J_{param} is estimated using different values of ΔT_c with fixed values for the other parameters. In the following simulations those parameters were fixed to:

$$[K_\Delta, q_{as}, \kappa_c, \alpha_c] = [2, 0.4, 0.3181, 0.8]$$

Those results are depicted in Figure 6.2. A good value with a low cost in this plot is $\Delta T_c = 0.4$, which will be used in later simulations and parameter estimations.

6.2.3 Estimate a value for the gain of the horizon inside the controller

Some experimental simulations using different values for this parameter have been shown that $K_\Delta = 2$ has given nice results. When trying to estimate this parameter with the **fmincon()** algorithm and using the starting value of the parameter to be equal to 2, the solver returns that this value is optimal. However, when using an initial value of 1.6, this also appears to be an optimal solution. The resulting cost using this value is larger then the cost using $K_\Delta = 2$ however, so more investigation has to be done. It seems that there are more local optimal values possible for this parameter, so a simulation with a range of values for this parameter (and by setting the other parameters fixed) has to be performed. The results of this simulation are shown in Figure 6.3, where can be shown that there are indeed some local optimal values. In the following simulations $K_\Delta = 2$ is used as fixed value for the next estimations, because it results in a low cost using the parameter-set in Figure 6.3 and in good results in previous chapters.

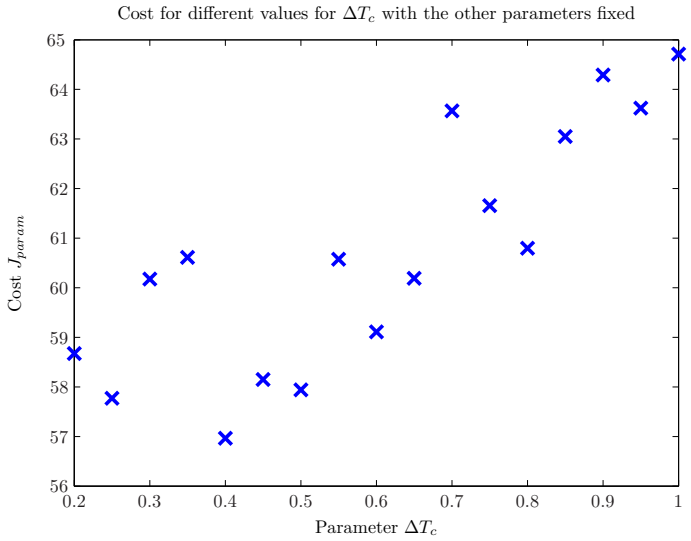


Figure 6.2: Cost with several values for ΔT_c with other parameters fixed.

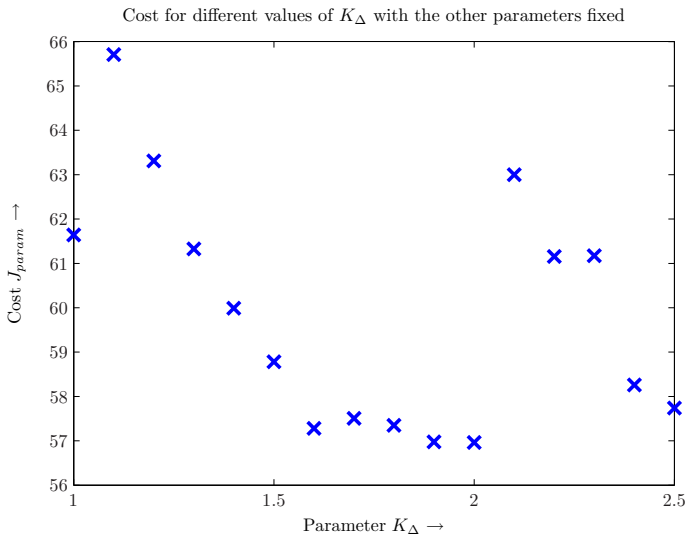


Figure 6.3: Cost with several values for K_Δ with other parameters fixed.

6.2.4 Estimating the values for the other parameters

Knowing the values for the fixed time related parameters ΔT_c and K_Δ, it is possible to estimate the values of the other unknown parameters in the receding horizon controller problem: q_{as}, κ_c & α_c.

Estimation with a known data-set

Because this is a time-consuming parameter estimation, first a run with a generated data-set, where the parameter values are known, has been performed, so the functionality of the scripts can be verified. The parameters used by generating the data-set are: $q_{as} = 0.4$, $\kappa_c = 0.3181$ and $\alpha_c = 0.8$. The starting values used in the **fmincon()** algorithm are $q_{as} = 0.3$, $\kappa_c = 0.4$ and $\alpha_c = 0.6$. This results in the following output of the optimiser:

Iter	F-count	f(x)	Max constraint	Line search steplength	Directional derivative	First-order optimality	Procedure
0	4	3.38657	-0.29				
1	10	2.86429	-0.2925	0.25	80.8	42.3	
2	16	2.33158	-0.2194	0.25	4.06	49.6	
3	23	1.47262	-0.4068	0.125	41	25.9	
4	28	1.05466	-0.2479	0.5	7.78	30.9	
5	39	0.940364	-0.2595	0.00781	-7.59	22.1	
6	45	0.815099	-0.2925	0.25	-1.24	26.5	
7	59	0.807211	-0.2929	0.000977	-8.06	26.3	Hessian modified
8	71	0.80391	-0.2931	0.00391	-0.844	26.2	
9	85	0.802785	-0.2931	0.000977	-1.15	26.2	
...	...	.....	.....	.....	.....	.....	

In those results some iterations were truncated, because the value for the cost function didn't change significant anymore. The values found for the parameters by the **fmincon()** algorithm are: $q_{as} = 0.3031$, $\kappa_c = 0.3774$ and $\alpha_c = 1.01830$, which are not the same as the values used when generating the data-set! That's why another optimisation toolbox is tried for the estimation of multiple parameters. In [1] the SolvOpt toolbox was used for the estimation of parameters, so this might be a good choice for the estimation of the parameters in the cost function of the receding horizon controller. SolvOpt [4] is a toolbox for Matlab that makes use of Shor's r-Algorithm. This optimiser provides the following output:

Iter.#	Function	Step Value	Gradient Norm
1	3.40932e+000	3.19197e-001	8.91252e+000
2	7.10620e-001	3.37751e-001	1.46239e+001
3	5.03522e-001	4.34072e-002	2.93076e+000
4	5.03415e-001	5.96319e-005	2.93175e+000
5	5.03415e-001	3.66265e-007	3.79954e+008
6	5.01190e-001	2.15015e-002	3.81202e+000
7	5.62710e-001	5.84367e-003	1.43510e+001
8	6.30110e-001	4.37695e-003	9.70061e+000
9	5.22887e-001	2.80409e-002	1.02110e+001
10	5.17176e-001	9.13889e-004	9.89456e+000
11	5.17175e-001	2.34080e-007	9.89562e+000
12	5.17175e-001	2.56215e-009	8.11623e+008
13	5.17224e-001	6.51224e-006	9.89393e+000
14	7.04589e-001	1.69023e-005	1.78181e+001
15	6.88117e-001	4.36827e-002	1.62742e+001
16	5.63789e-001	4.06016e-002	1.87492e+001
17	5.84438e-001	2.04968e-002	5.13517e+000
18	6.43867e-001	1.67643e-001	1.79320e+001
19	8.10938e-001	3.24068e-001	8.20306e+000
20	3.04923e-001	4.44218e-001	4.98653e+000
21	2.95743e-001	2.27953e-002	3.54291e+000
22	2.98034e-001	7.13164e-003	7.15002e+000
23	2.79047e-001	2.01220e-002	5.25262e+000
24	7.91519e-002	1.92151e-001	2.51665e+001
25	6.19650e-002	1.44299e-002	1.47890e+001
26	2.92805e-002	2.03590e-002	9.95689e+000
27	1.57005e-002	8.79709e-003	2.32011e+001
28	1.64859e-003	3.10456e-003	6.21694e+000
29	5.39313e-004	4.21267e-004	1.82306e+001
30	5.20116e-004	5.09729e-004	5.27205e+000
31	3.32720e-004	1.58621e-004	1.86693e+001
32	8.67127e-005	1.70083e-004	4.07501e+000
33	7.05856e-005	1.93226e-005	1.20508e+001
34	4.36223e-005	4.34067e-005	2.50469e+001
35	2.09912e-005	2.59274e-006	3.44443e+000
36	1.14874e-005	5.81368e-006	1.87636e+001
37	8.01783e-006	7.18902e-006	7.98942e+000

After those 37 iterations, which is equal to 497 function evaluations, the good values for the parameters have been found!

Estimation with measurements

So, it seems that the SolvOpt optimisation tool results in a better estimation of the parameters. So, this tool is used to estimate the parameters in the measurement-data, using the following starting values for the parameters:

$$[q_{as}, \kappa_c, \alpha_c] = [0.4, 0.3181, 0.8].$$

This resulted in the following simulation output, which has taken over two days on my notebook (Intel Centrino 1.4 GHz, 512MB memory):

```
Iter.# ..... Function ... Step Value ... Gradient Norm
    1    5.58777e+001    3.10646e-001      2.00005e+002
    2    5.55441e+001    1.11134e-001      3.09183e+000
    3    5.56520e+001    1.38928e-001      2.92658e+000
    4    5.52814e+001    3.05088e-001      2.29451e+000
    5    5.76122e+001    7.18310e-002      2.00014e+002
    6    5.52522e+001    3.79329e-002      1.72855e+000
    7    5.53165e+001    8.76398e-002      2.00000e+002
    8    5.51741e+001    8.87469e-003      3.77144e-001
    9    5.52365e+001    2.76282e-002      1.07123e+000
  ...    ............    ............      ............
  ...    ............    ............      ............
  534    5.50301e+001    9.16660e-008      2.84223e+000
SolvOpt: Termination warning:
Result may not provide the optimum. The function apparently has many extremum points.
The above warning may be reasoned by inaccurate gradient approximation
```

The estimated parameter values by SolvOpt are:

$$[q_{as}, \kappa_c, \alpha_c] = [0.0100, 0.0100, 1.0233].$$

The decrease of the cost is not that large however. It looks that it keeps jumping between two different sets of parameters with a very small value for the step value. So, the parameter estimation is also performed using the **fmincon()** function. In this estimation, the same starting values and cost function are used as in the one with SolvOpt. This resulted into the following output:

```
                          Max     Line search  Directional  First-order
 Iter F-count        f(x)   constraint   steplength   derivative   optimality Procedure
    0       4      56.9638      -0.3081
    1       8      55.1699  -8.674e-018            1        0.841         11.1
    2      20      55.1565  -8.674e-018      0.00391        -3.66         8.04
    3      27      54.1481    -0.001728        0.125        -66.6           41  Hessian modified twice
    4      38      53.8484    -0.001715      0.00781        -18.6         57.8  Hessian modified twice
    5      46      53.8441     -0.00202       0.0625      -0.0658         12.9
    6      69       53.843    -0.002013     3.05e-005        -16.5         14.8
    7      79      53.8419    -0.003429       0.0156      -0.0646         9.94
    8      93      53.8419    -0.003718     0.000977      -0.0623         9.92
    9     104      53.8418    -0.003805      0.00781      -0.0082         9.91  Hessian modified twice
  ...     ...     .........    ........    .........      .......       ......
```

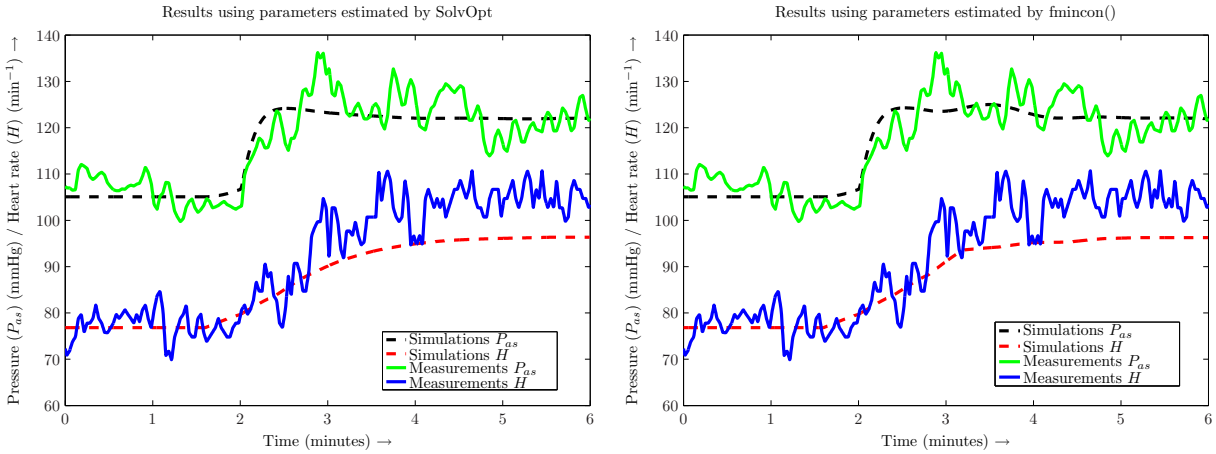
Now **fmincon()** results in a set of parameters with a smaller value for the cost function ($J_{param} = 53.8418$) than the one found by SolvOpt. The parameters found by **fmincon()** have the following values:

$$[q_{as}, \kappa_c, \alpha_c] = [0.0241, 0.0185, 0.7099].$$

Those found parameters result in a lower value of the cost function, which should provide a better fit of the measurement data. In Figure 6.4 both sets of parameters are used in a simulation and compared with the measurement data set used in the estimation.

Because the numerical results of the SolvOpt optimiser look like a local minima of the cost function, the results of the **fmincon()** simulation have been used as starting values for the parameters in the estimation, what resulted into the following output:

```
Iter.# ..... Function ... Step Value ... Gradient Norm
    1    5.51148e+001    3.85758e-001      2.44623e+000
    2    5.52828e+001    2.08296e-002      2.06888e+000
    3    5.98092e+001    3.34642e-002      2.82854e+002
    4    5.51309e+001    3.55096e-002      2.46571e+000
    5    5.52790e+001    3.50125e-003      2.00101e+000
    6    5.51259e+001    3.81105e-003      2.57444e+000
    7    5.52749e+001    3.97206e-003      1.92718e+000
    8    5.53099e+001    3.39272e-003      2.00009e+002
    9    5.52739e+001    3.51496e-003      1.91655e+000
   10    5.55668e+001    1.33882e-002      2.00017e+002
   ..    ............    ............      ............
```



(a) Parameters found by SolvOpt

(b) Parameters found by fmincon()

Figure 6.4: Simulations results with estimated parameters vs measurement data.

One can notice that the optimiser increases the value of the cost in the first iterations! After a lot of function evaluations, the cost doesn't decrease sufficient using this set of beginning values found for the parameters by **fmincon()**.

One can see in Figure 6.4 that the values for the arterial systemic pressure P_{as} fit quite good using both parameter-sets, but there is a relative large error in the value of the heart rate H however. In Figure 6.5 the absolute error of the measurements and the simulated data (using the parameters found by **fmincon()**) this can be confirmed. This latter error can be compensated by changing some parameters in the cardiovascular model. Two of the model-parameters that can have a large influence in the relation between the pressure P_{as} and the heart rate H are c_l and c_r. The values for those two parameters are determined using earlier parameter estimation in [1], where the LQR controller was used for the baroreceptor loop. To improve the quality of the system, those two parameters need to be estimated with this new type of controller. The minimization problem defined in section 6.1 needs to be extended a bit.

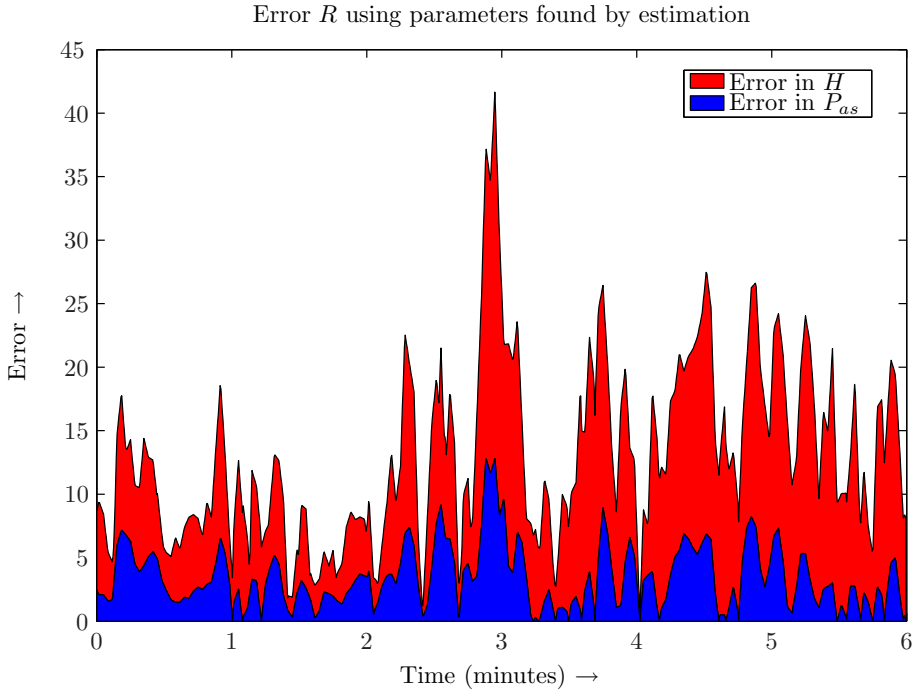


Figure 6.5: Absolute errors $|H^{sim}(t) - H^{meas}(t)|$ and $|P_{as}^{sim}(t) - P_{as}^{meas}(t)|$.

6.2.5 Estimating parameters in controller and model

Now also the parameters c_r and c_l have to be defined in the optimisation problem. Before redefining the minimization problem, those two parameters have to be bounded:

$$0.01 \leq c_r \leq 0.1 \quad \text{and} \quad 0.01 \leq c_l \leq 0.1. \tag{6.3}$$

The value for the control output interval ΔT_c is fixed to a value of 0.4 minutes and K_Δ is fixed to a value of 2, so those parameters should not be in the minimization problem anymore. The new minimization problem for the parameter estimation can be defined as:

$$\min_{q_{as},\kappa_c,\alpha_c,c_r,c_l} J_{param} = \int_0^T |H^{sim}(t) - H^{meas}(t)| + |P_{as}^{sim}(t) - P_{as}^{meas}(t)|dt \tag{6.4}$$
$$\text{subject to : } (6.1) \;\&\; (6.3)$$

The starting values of the parameters c_l and c_r are estimated on the same way as done for the parameter κ_c: by doing a short parameter estimation with the beginning parameters used in the previous chapter. This results in the new values for c_l and c_r that are used as starting conditions of the new parameter estimation: $c_l = 0.0157$ and $c_r = 0.0487$. The **fmincon()** optimiser resulted the following output during this short estimation:

```
                                 Max      Line search  Directional  First-order
 Iter F-count        f(x)    constraint   steplength   derivative   optimality Procedure
    0       3      53.8418     -0.01205
    1       8      51.2935    -0.009041         0.25         95.4    6.25e+003
    2      13      49.7956     -0.00678         0.25         1.41    3.22e+003
    3      21      49.1816    -0.009381       0.0313          170    5.33e+003
    4      26      45.5809     -0.01165         0.25        -1.93          515
    5      33      44.8525     -0.01227       0.0625         3.23          188
Maximum number of iterations exceeded;
 increase OPTIONS.MaxIter.
```

As said before, now a large parameter estimation can be performed using the following initial conditions (ΔT_c is fixed to 0.4 and K_Δ is set to 2):

$$[\kappa_c, \alpha_c, c_l, c_r] = [0.2882, 0.5934, 0.3177, 0.0157, 0.0487]$$

After a long simulation, the following output results from the optimisation tool fmincon():

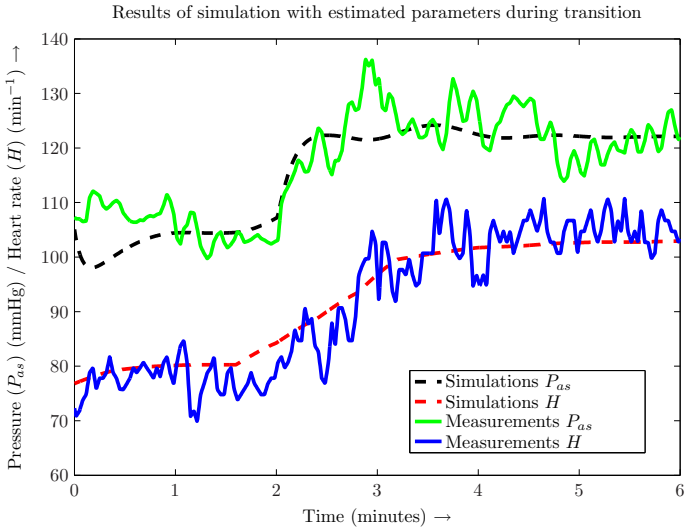


Figure 6.6: Simulation results of system with estimated parameters and measurements.

Iter	F-count	f(x)	Max constraint	Line search steplength	Directional derivative	First-order optimality	Procedure
0	6	44.8525	0				
1	17	44.7946	0	0.0313	11.6	1.44e+003	
2	24	43.8901	0	0.5	4.89	3.38e+003	
3	32	43.8658	0	0.25	9.24	2.76e+003	
4	39	42.4828	0	0.5	-0.714	75.4	
5	46	42.2889	0	0.5	0.719	1.62e+003	
6	57	42.2787	0	0.0313	-0.307	1.41e+003	
7	67	42.2627	0	0.0625	-0.218	1.25e+003	
8	77	42.2494	0	0.0625	-0.207	1.22e+003	
9	84	42.2053	0	0.5	0.207	582	
10	94	42.2017	0	0.0625	-0.0574	584	
11	104	42.1865	0	0.0625	-0.178	28.3	
12	114	42.1838	0	0.0625	-0.0427	338	
13	124	42.1779	0	0.0625	-0.149	177	
14	140	42.1777	0	0.000977	-0.142	126	Hessian modified
...	...	.....	.....	.....	.....	.....	

The values found for the parameters give a much better result then the ones used in the beginning. The cost (J_{param}) is decreased from a value of around 57 (with the old model parameters) to a value of 42, which is a large improvement. The values of those parameters found by the optimiser are:

$$[q_{as}, \kappa_c, \alpha_c, c_l, c_r] = [0.0881, 0.03741, 0.12, 0.01062, 0.08543]$$

The simulation results of the transition and the measurements are plotted in Figure 6.6. The error in the value of the heart rate H after the transition is decreased. This can be verified in Figure 6.7 which depicts the absolute errors in the heart rate and the arterial systemic pressure.

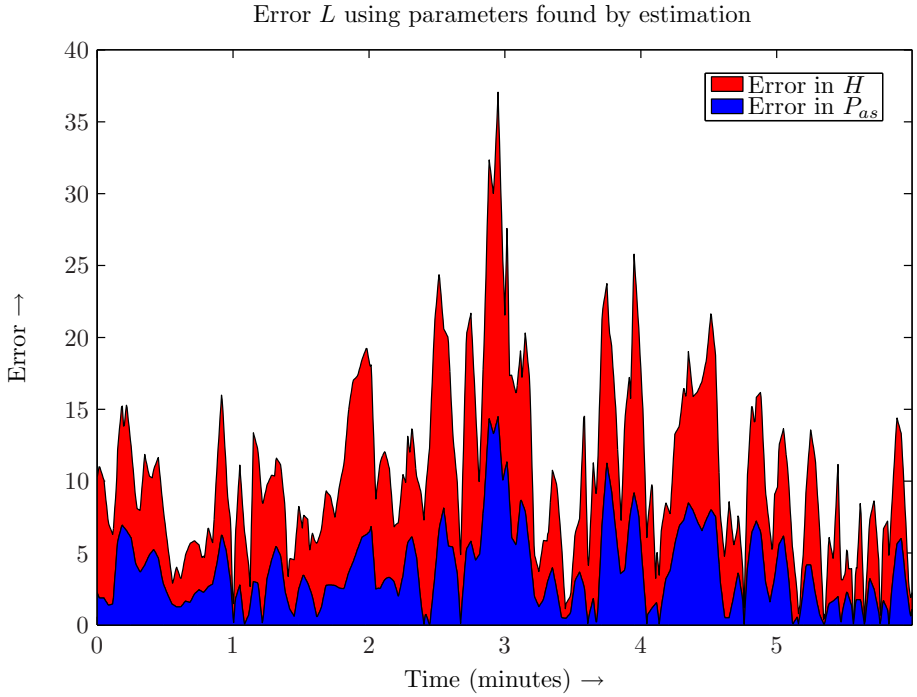


Figure 6.7: Absolute errors $|H^{sim}(t) - H^{meas}(t)|$ and $|P_{as}^{sim}(t) - P_{as}^{meas}(t)|$.

CHAPTER 7

Summary, conclusions and recommendations

In the beginning of this report, the baroreceptor loop was modelled using an LQR controller, which stabilized the closed loop behaviour of the system. Because the model for the cardiovascular system contains a set of non-linear differential equations, it had to be linearized for this control technique. This linearization is undesirable, so a non-linear controller is designed for the baroreceptor loop in this research report.

A receding horizon controller has been introduced and implemented. In Chapter 4 the results of this controller are depicted and discussed. The behaviour of the system depends on the parameters that are used in the cost function of the controller. Also the different time horizons play a role in the performance of the controller. How those different properties of the controller depend on this behaviour and performance is discussed, which resulted in a set of parameters and horizons that gives good results.

Using those parameters, the behaviour of the system with noise applied on (both) the in- and output of the system is investigated in Chapter 5. It seems that the receding horizon controller can deal with those noise additions, because the system keeps behaving stable.

Because the model will later on be compared with the real process in a human being, the parameters chosen should result in simulations that match the real life. Because clinical measurement data is available, a parameter estimation using two different optimizers has been performed. After some problems, a set of parameters has been found which resulted in simulation results that match reality. Not only the parameters in the controller are estimated but, because of the replacement of the LQR controller, also parameters in the model had to be re-estimated. All those estimations are discussed in Chapter 7 of this report.

In the last weeks of my traineeship period I have spent a lot of time on (re-)doing parameter estimations, which are quite time consuming, and writing a paper [5] that is submitted to a journal about cardiovascular systems now. Prof. Kappel did some comparisons of the results with the previous used LQR controller and the receding horizon control technique. A detailed comparison with the results obtained with the LQR controller is not possible at this stage of the investigations. The reason is that a careful parameter identification procedure, which involves all parameters of the receding horizon controller and the cardiovascular model (as done before for the system when using the LQR controller [1], has to be done.

This is one of the things that could be done in future work. Also more complicated models for the cardiovascular (possible in combination with the respiratory) systems can use this type of controller. Another future application on this topic is to utilize this control strategy in cases where state constraints are important, as for instance in models for the reaction of the cardiovascular system to hemorrhage.

APPENDIX A

Parameters used in the model

Parameter	Meaning	Units	Value
c_{as}	Compliance of the arterial systemic compartment	liter/mmHg	0.01016
c_{vs}	Compliance of the venous systemic compartment	liter/mmHg	0.64995
c_{ap}	Compliance of the arterial pulmonary compartment	liter/mmHg	0.03609
c_{vp}	Compliance of the venous pulmonary compartment	liter/mmHg	0.14078
c_l	Compliance of the relaxed left ventricle	liter/mmHg	0.023054
c_r	Compliance of the relaxed right ventricle	liter/mmHg	0.044326
V_{tot}	Total blood volume	liter	6.1
R_p	Resistance in the peripheral region of the pulmonary circuit	mmHg min/liter	1.6 / 0.32
R_l	Inflow resistance of the left ventricle	mmHg min/liter	0.243
R_r	Inflow resistance of the right ventricle	mmHg min/liter	0.0567
κ	Coefficient in Bazetts formula (2.7)	$\text{min}^{\frac{1}{2}}$	0.0516^6
C_{a,O_2}	O_2-concentration in arterial systemic blood	1	0.2
K	Constant in the formula for the biochemical energy flow	liter	15.9591
A_{pesk}	Constant in the formula (2.12)	mmHg min/liter	178 / 254
M_0	Metabolic rate with $W = 0$	liter/min	0.35
ρ	Coefficient of W in the relation with the metabolic rate	liter/(min Watt)	0.011
α_l	Coefficient of S_l in the differential equation for S_l	min^{-2}	31.592
α_r	Coefficient of S_r in the differential equation for S_r	min^{-2}	28.342
β_l	Coefficient of H in the differential equation for S_l	mmHg/min	25.065
β_r	Coefficient of H in the differential equation for S_r	mmHg/min	1.416
γ_l	Coefficient of $\dot{S}_l$ in the differential equation for S_l	min^{-1}	1.332
γ_r	Coefficient of $\dot{S}_r$ in the differential equation for S_r	min^{-1}	2.045

Table A.1: Parameters used in the model for the cardiovascular system.

In Table A.1 the parameters that are used in the simulations are given. If two values are given in the table, the first one is the parameter used when the system is in the resting phase ($W = 0$) and the second one if the system is active ($W = 75$).

The values of the equilibrium states $\underline{x}^{rest}$ and $\underline{x}^{exer}$ using those sets of parameters are given by:

$$\underline{x}^{rest} = [105.0767, 4.3015, 12.2508, 4.9232, 60.9315, 0.0205, 3.8376, 0.0001, 22.0047, 76.8127]^T$$

and

$$\underline{x}^{exer} = [121.7808, 3.7156, 10.9826, 8.3589, 79.6893, -0.0122, 5.0180, -0.0002, 14.3997, 100.4369]^T.$$

APPENDIX B

Matlab files

In this appendix, all Matlab files created during the traineeship are included. The names and a short description of those files are:

- **simulate.m**: The main file which calls all other functions
- **param_mod.m**: All parameters for the model and simulation are stored in this M-file
- **sim_model.m**: Simulate the model of the cardiovascular system
- **nonlin_model.m**: Contains the model equations of the system
- **recedinghorizon.m**: The function that calls all function in the receding horizon controller
- **param_rhc.m**: Parameters used for the receding horizon controller
- **calccost.m**: Calculate the cost function in the receding horizon controller
- **simpson.m**: The function that integrates using Simpson's rule
- **sim_lambda.m**: Simulate the values for the Lagrange multipliers
- **calculate_Fx.m**: Calculate the values for the Fx matrix (derivation done by hand)
- **test_AD.m**: Do a simulation using the Automatic Differentiation toolbox TomLab
- **update_u.m**: Update the values for u, using the Armijo Backtracking algorithm
- **paramest.m**: Parameter estimation of control- and model parameters
- **costfun.m**: Cost function called during the parameter estimation

simulate.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: simulate.m
% ------------------------------------------------------------------------
% Usage: simulate;
%        Simulates the receding horizon controller closed loop.
%
% Input:  None
% Output: None
% ------------------------------------------------------------------------

%% SOME MATLAB THINGS
format long e;
set(0,'defaulttextinterpreter','none');
clear all;

%% INITIAL VALUES FOR THE SYSTEM
% Define a value for x0 as initial condition at the first run:
% x0 = [105, 4.3, 12, 5, 60, 0, 4, 0, 22, 78.85]';
% x0  =   [122 4 15 6 90 0 4 0 17 105]';
x0 = [121.7808, 3.7156, 10.9826, 8.3589, 79.6893, -0.0122, 5.0180, ...
                                                  -0.0002, 14.3997, 100.4369]';
x0_contr = x0;

%% GET THE PARAMETERS FOR THE RECEDING HORIZON CONTROLLER
% Call the param_rh function described in the corresponding m-file:
% param_rh = param_rhc([0.2882, 0.5934, 0.3177, 999, 999]);
% param_rh.modparam = [999, 999];
param_rh = param_rhc([8.810724771415854e-002    3.740844246730053e-002  ...
                                        1.200000000000000e-001, 999, 999]);
param_rh.modparam = [1.062151316587921e-002 8.542885243501266e-002];


%% TIME SETTINGS FOR THE TOTAL SIMULATION
% Define the total simulation time t_sim in minutes:
t_sim = 26;

% Calculate the number of iterations in the simulate for-loop
numit = ceil(t_sim / param_rh.time_contr);

%% SET THE CHANGES IN THE EQ.POINTS FOR THE RH CONTROLLER & MODEL SIM.
% Define the changes in the format: [time, state (1=rest, 2=exer)]
param_rh.eq = [0, 1 ;
               10, 2 ;
               18, 1];

%% CREATE SOME EMPTY ARRAYS TO SAVE DATA
u_total = [];
x_total = [];
t_total = [];
noise_total = [];
simres_opt = [];
simres_simtime = [];
x_eq_total = [];
optim_it = [];
optim_J = [];

%% START THE SIMULATION FOR-LOOP
% Call the for-loop:
for i = 1 : numit
    disp('Iteration'); disp(i);
    % Define the beginning and end-time of this iteration
    param_rh.t0 = (i-1) * param_rh.time_contr;

    % Calculate the output of the Receding Horizon controller
    [t, u, simres] = recedinghorizon(x0_contr, param_rh);

    % Add some noise to the output of the controller
    mask = (rand(1,length(t)) < 0.5);
    mask = mask - (mask == 0);
```

```
    noise_u = param_rh.noise_u * rand(1,length(t)) .* mask;
    noise_total = [noise_total , noise_u(1:end-1)];
    u = u + noise_u;

    % Save the simulation results and the input u in arrays
    t_total = [t_total , t(1:end-1)];
    u_total = [u_total , u(1:end-1)];
    simres_opt = [simres_opt ; simres.optimisation_stats];
    simres_simtime = [simres_simtime ; simres.simtime];
    x_eq_total = [x_eq_total, simres.x_eq(:,1:end-1)];

    % Save the number of iterations in the optimisation for a plot
    optim_it = [optim_it, sum(simres_opt(:,1) == param_rh.t0) ];
    optim_J  = [optim_J , simres_opt(end+1-optim_it(end),3)];

    % Simulate the system instead of feeding the output of the controller
    % to the real system (only if noise is added to the input!)
    if (param_rh.noise_u)
        x = sim_model(t, u, x0, param_rh.eq, param_rh);
    else
        x = simres.x;
    end

    % Set the last value of the simulated x as input for the next
    % simulation of the model (without noise!)
    x0 = x(:,end);

    % Add the noise to the output of the system. So create a mask and add
    % the noise
    mask = (rand(1,length(t)) < 0.5);
    mask = mask - (mask == 0);
    x = x + [param_rh.noise_x * rand(1,length(t)) .* mask ; zeros(9,length(t))];

    % Set the initial value for the receding horizon controller
    x0_contr = x(:,end);

    % Save those results to a large array
    x_total = [x_total , x(:,1:end-1)];
end
%% PLOT THE RESULTS
% Plot the function u in the first figure
figure; plot(t_total, u_total, 'LineWidth', 1);
xlabel('Time (minutes) \rightarrow');
ylabel('Input u (min^{-1}) \rightarrow');
title(['Solution for u with \Delta T_c = ', num2str(param_rh.time_contr),  ...
    ', \Delta T_h = ', num2str(param_rh.time_sim), ', q_{as} = ', ...
       num2str(param_rh.q_as), ' and \kappa_c = ', num2str(param_rh.kappa_c), ...
        ', \alpha_c = ', num2str(param_rh.alpha_c) , '']);
% Plot the states P_as and H in the second figure
figure;
plot(t_total, x_total(1,:), t_total, x_total(10,:), '--r', 'LineWidth', 2);
xlabel('Time (minutes) \rightarrow');
ylabel('Pressure (mmHg) / Heart rate (min^{-1}) \rightarrow');
title(['Solution for P_{as} and H with u from receding horizon (q_{as} = ', ...
       num2str(param_rh.q_as), ' , \kappa_c = ', num2str(param_rh.kappa_c), ...
        ', \alpha_c = ', num2str(param_rh.alpha_c) , ')']);
h = legend('P_{as}', 'H');
set(h, 'FontSize', 24);
% Plot the results of the optimisation: decrease of interations and init.
% cost (relative)
figure; plot(0:param_rh.time_contr:t_sim-param_rh.time_contr, optim_it, 'or', ...
                                                        'MarkerSize',7);
hold on; plot(0:param_rh.time_contr:t_sim-param_rh.time_contr, ...
                                        optim_J./max(optim_J).*10, 'LineWidth', 2);
title(['Optimisation results in controller (Cost has to be multiplied with ', ...
                                                    num2str(max(optim_J)/10), ')']);
xlabel('Time (minutes) \rightarrow');
ylabel('Number of iterations / Scaled initial costs \rightarrow');
i = legend('\# iterations', 'Scaled cost');
set(i, 'FontSize', 24);

%% CLEAR SOME UNDESIRED VARS IN THE WORKSPACE
clear x0 x0_contr mask x u t simres numit i t_sim h i ans
```

param_mod.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: param_mod.m
% ------------------------------------------------------------------------
% Usage: parameters = param_mod ( state, guess );
%        Set the parameters for the CV system
%
% Input:  state: which state are we (rest / exercise)
%         guess: the values for [c_l and c_r] (param.est)
% Output: parameters: a struct with the parameters used
% ------------------------------------------------------------------------
function parameters = param_mod ( state, guess )

% Save the state given in the parameters array
parameters.state    =   state;

% Set all parameters:
parameters.c_as     =   0.01016;
parameters.c_vs     =   0.64995;
parameters.c_ap     =   0.03609;
parameters.c_vp     =   0.14078;
if (guess(1) == 999)
    parameters.c_l      =   0.023054;
else
    parameters.c_l = guess(1);
end
if (guess(2) == 999)
    parameters.c_r      =   0.044326;
else
    parameters.c_r = guess(2);
end
parameters.R_l      =   0.243;
parameters.R_r      =   0.0567;
parameters.kappa    =   0.0516^6;
parameters.C_aO2    =   0.2;
parameters.K        =   15.9591;
parameters.M_0      =   0.35;
parameters.rho      =   0.011;
parameters.alpha_l  =   31.592;
parameters.alpha_r  =   28.342;
parameters.beta_l   =   25.065;
parameters.beta_r   =   1.416;
parameters.gamma_l  =   1.332;
parameters.gamma_r  =   2.045;

% Parameters depending on the workload (exercise or rest)
if (state == 1)
    parameters.R_p      =   1.6;
    parameters.A_pesk   =   178;
    parameters.W        =   0;
else
    parameters.R_p      =   0.32;
    parameters.A_pesk   =   254;
    parameters.W        =   75;
end
```

sim_model.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: sim_model.m
% ------------------------------------------------------------------------
% Usage: x = sim_model(t, u, x0, changes);
%        Simulate the model for the CV-system open-loop.
%
% Input:  t: the array with time elements (equally spread!)
%         u: the output of the controller
%         x0: the initial value of the states in the system
%         changes: possible changes in equilibr. states that can occur
%         param_rh: parameters for the values of c_r and c_l (param est)
% Output: x: the output of the system for all states
% ------------------------------------------------------------------------
function x = sim_model(t, u, x0, changes, param_rh)

%% START THE SIMULATION OF THE MODEL
% Calculate the timestep h and the number of samples
h = (t(end)-t(1))/length(t);
numsamp = length(t);

% Create an empty matrix for x and set the initial value
x = zeros(10,numsamp);
x(:,1) = x0;

% Set some temporary variables for the changes statement
p = 1;
p_max = size(changes,1);
parameters = param_mod(1, param_rh.modparam);

% Walk through the for loop to solve the differential equations using the
% Runga Kutta (or Euler) method:

for j = 1 : numsamp - 1
    % Check if the parameters for the simulation are changed
    for p = 1 : p_max
        if (t(j) >  changes(p,1))
            parameters = param_mod( changes(p,2), param_rh.modparam );
        end
    end

    % The old lines for the Euler method:
    %x(j+1,:) = x(j,:) + h * nonlin_model(t(j), x(j,:), parameters, u, t)';

    % The four parts of the Runga Kutta method:
    F1 = nonlin_model(t(j), x(:,j), parameters, u, t);
    F2 = nonlin_model(t(j) + h/2, x(:,j) + h/2 * F1, parameters, u, t);
    F3 = nonlin_model(t(j) + h/2, x(:,j) + h/2 * F2, parameters, u, t);
    F4 = nonlin_model(t(j) + h, x(:,j) + h * F3, parameters, u, t);

    % Combine those four parts to estimate the new value for x
    x(:,j+1) = x(:,j) + h/6 * (F1 + 2*(F2+F3) + F4);
end
```

nonlin_model.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: nonlin_model.m
% ------------------------------------------------------------------------
% Usage: dx = nonlin_model(t, x, parameters, u, tspan);
%        The differential equations for the nonlin model
%
% Input:  t: the array with time elements (equally spread!)
%         x: the previous simulated values of the states
%         u: the array with control output u
%         tspan: the time array with the control output u
% Output: dx: the value to be added to x
% ------------------------------------------------------------------------
function dx = nonlin_model(t, x, parameters, u, tspan)

% Interpolate the value for u at time t (of the ODE solver)
u = interp1(tspan,u,t);

% Get W just from the parameters
W = parameters.W;

% Make some conversions of the states x to the real names:
P_as = x(1); P_vs = x(2); P_ap = x(3); P_vp = x(4); S_l = x(5);
sigma_l = x(6); S_r = x(7); sigma_r = x(8); R_s = x(9); H = x(10);

% First calculate the nonlinear functions t_d, k(H) and a(H)
t_d = 1 / sqrt(H) * (1 / sqrt(H) - parameters.kappa);
k_l = exp(-t_d / (parameters.R_l * parameters.c_l));
a_l = 1 - k_l;
k_r = exp(-t_d / (parameters.R_r * parameters.c_r));
a_r = 1 - k_r;

% Calculate the differential equations
dP_as = 1 / parameters.c_as * ( H * (parameters.c_l * a_l * P_vp * S_l ) / ...
        (a_l * P_as + k_l * S_l) - 1 / R_s * (P_as - P_vs) );
dP_vs = 1 / parameters.c_vs * ( 1 / R_s * (P_as - P_vs) - ...
        H * (parameters.c_r * a_r * P_vs * S_r ) / (a_r * P_ap + k_r * S_r));
dP_ap = 1 / parameters.c_ap * ( H * (parameters.c_r * a_r * P_vs * S_r) / ...
        (a_r * P_ap + k_r * S_r) - 1 / parameters.R_p * (P_ap - P_vp));
dP_vp = 1 / parameters.c_vp * (1 / parameters.R_p * (P_ap - P_vp) - ...
        H * (parameters.c_l * a_l * P_vp * S_l) / (a_l * P_as + k_l * S_l));

dS_l = sigma_l;

dsigma_l = parameters.beta_l * H - parameters.alpha_l * S_l - parameters.gamma_l * sigma_l;

dS_r = sigma_r;

dsigma_r = parameters.beta_r * H - parameters.alpha_r * S_r - parameters.gamma_r * sigma_r;

dR_s = 1 / parameters.K * (parameters.A_pesk * ( (P_as - P_vs) / R_s * parameters.C_aO2 - ...
       (parameters.M_0 + parameters.rho * W)) - (P_as - P_vs));

dH = u;

% Create an empty array for the states
dx = zeros(10,1);

% Create the differential equations
dx(1) = dP_as;
dx(2) = dP_vs;
dx(3) = dP_ap;
dx(4) = dP_vp;
dx(5) = dS_l;
dx(6) = dsigma_l;
dx(7) = dS_r;
dx(8) = dsigma_r;
dx(9) = dR_s;
dx(10) = dH;
```

recedinghorizon.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: recedinghorizon.m
% ------------------------------------------------------------------------
% Usage: [t, u, simres] = recedinghorizon(x0, param_rh);
%        Estimate the output of the receding horizon controller
%
% Input:  x0: vector with values for the output of the last simulation
%         param_rh: struct with parameters for the controller
% Output: t: an array with the simulation time
%         u: the control output
%         simres: other results of the controller in a struct
% ------------------------------------------------------------------------
function [t, u, simres] = recedinghorizon(x0, param_rh);

%% CREATE THE TIMESPANS FOR THE SIMULATION & CONTROL HORIZONS
% The timespans are linear and defined by:
t_sim = param_rh.t0:(1/param_rh.samples):(param_rh.t0 + param_rh.time_sim);
t_contr = param_rh.t0:(1/param_rh.samples):(param_rh.t0+param_rh.time_contr);


%% CREATE A x_eq MATRIX BECAUSE CHANGES IN THE TIME INTERVAL CAN OCCUR
% First create two empty masks
mask_1 = zeros(size(t_sim));
mask_2 = zeros(size(t_sim));

% Walk through the matrix which contains the changes
for i = 1 : size(param_rh.eq,1)
    if (param_rh.eq(i,2) == 1)
        mask_1 = mask_1 + (t_sim > param_rh.eq(i,1));
        if (i ~= size(param_rh.eq,1))
            mask_1 = mask_1 - (t_sim > param_rh.eq(i+1,1));
        end
    end
    if (param_rh.eq(i,2) == 2)
        mask_2 = mask_2 + (t_sim > param_rh.eq(i,1));
        if (i ~= size(param_rh.eq,1))
            mask_2 = mask_2 - (t_sim > param_rh.eq(i+1,1));
        end
    end
end

% Define the two equilibria points
x_rest  =   param_rh.x_rest;
x_exer  =   param_rh.x_exer;

% Create the complete matrix
x_eq = x_rest * mask_1 + x_exer * mask_2;

%% SET SOME INITIAL VALUE FOR u AND SOME INITIAL VARIABLES
% The value of u initial will be zero, with length defined by t_sim:
u = zeros(1, length(t_sim));
u_old = zeros(1,length(t_sim));

% The first iteration will be 0, so k = 0:
k = 0;
simres.optimisation_stats = [];
tic;

%% START THE WHILE LOOP FOR THE RECEDING HORIZON CONTROLLER
while (1)

    % SIMULATE THE MODEL FOR A TIME INTERVAL t_sim:
    x = sim_model(t_sim, u, x0, param_rh.eq, param_rh);

    % CALCULATING THE COST FUNCTION
    J = calccost(x, u, t_sim, param_rh, x_eq);

    % CALCULATE THE LAGRANGE MULTIPLIERS FOR A TIME INTERVAL t_sim:
    lambda = sim_lambda(t_sim, param_rh, x, x_eq);
```

```
70
71      % CALCULATE THE GRADIENT USING THE KNOWN FORMULA
72      grad_J = 2 * param_rh.kappa_c^2 .* u - lambda(10,:);
73
74      % DETERMINE THE U-NORM OF THIS GRADIENT
75      % norm_grad_J = trapz(t_sim,grad_J.^2);             % Old: using trapz()
76      norm_grad_J = simpson(t_sim,grad_J.^2);
77
78      % SOME SPECIAL THINGS TO DO AT k = 0:
79      if ( k == 0 )
80          % Save the initial norm of the gradient when k = 0 for stopping
81          % criteria:
82          norm_grad_J_0 = norm_grad_J;
83
84          % There's no good value for s_k yet, because the backtracking has
85          % never been performed yet (so set 9.9999):
86          s_k = 9.9999;
87
88          % Some debugging information
89          if (param_rh.debug)
90              disp(['    |---------|-----------------|-------------------|------------|']);
91              disp(['    |    k    |        J        |    norm grad J    |     s      |']);
92              disp(['    |---------|-----------------|-------------------|------------|']);
93          end
94      end
95
96      % DEBUG INFORMATION AT EACH ITERATION k
97      if (param_rh.debug)
98      disp(['    |    ', num2str(k,'%02.2d'), '   |    ', num2str(J,'%09.2f'), '    |    ',...
99              num2str(norm_grad_J,'%011.4f'), '    |   ', num2str(s_k,'%06.4f') , '   |' ]);
100     end
101     simres.optimisation_stats = [simres.optimisation_stats ; param_rh.t0 , k , ...
102                                                              J, norm_grad_J, s_k];
103
104     % DEFINE THE BREAK CONDITIONS IF OPTIMISATION IS GOOD ENOUGH
105     break1 = (norm_grad_J < param_rh.eps_abs);
106     break2 = (norm_grad_J / norm_grad_J_0 < param_rh.eps_rel);
107     break3 = (k > param_rh.k_max);
108     break4 = (s_k < 0.0000001);
109     break5 = (abs(simpson(t_sim,u_old.^2)-simpson(t_sim,u.^2))<param_rh.eps_u)&&(k ~= 0);
110
111     % CHECK IF BREAKING CONDITIONS HOLD
112     if (break1 || break2 || break3 || break4 || break5)
113         % Some debug information
114         if (param_rh.debug)
115             disp(['    |---------|-----------------|-------------------|------------|']);
116             if (break1)
117                 disp(['    |      ABSOLUTE STOPPING CRITERIA! (eps_abs = ', ...
118                             num2str(param_rh.eps_abs,'%06.4f'), ')        |']);
119             end
120             if (break2)
121                 disp(['    |      RELATIVE STOPPING CRITERIA! (eps_rel = ', ...
122                             num2str(param_rh.eps_rel,'%06.4f'), ')        |']);
123             end
124             if (break3)
125                 disp(['    |      MAXIMUM NUMBER ITERATIONS REACHED! (k_max = ', ...
126                              num2str(param_rh.k_max,'%02.2d'), ')       |']);
127             end
128             if (break4)
129               disp(['    |      VALUE FOR S TOO SMALL: (s < 0.0000001)                |']);
130             end
131             disp(['    |---------|-----------------|-------------------|------------|']);
132             disp(' ');
133         end
134
135         % STOP THE WHILE(1) LOOP AND SAVE SIMULATION TIME
136         simtime = toc;
137         simres.simtime = [param_rh.t0 , simtime];
138         simres.x = x(:,1:length(t_contr));
139         simres.x_eq = x_eq(:,1:length(t_contr));
140         u = u(1:length(t_contr));
141         t = t_contr;
142         break;
```

```
143     end
144
145     % UPDATE THE VALUES FOR THE INPUT, AND SAVE THE OLD ONE!
146     u_old = u;
147     [u, s_k] = update_u(x0, x_eq, param_rh, J, u, t_sim, grad_J, norm_grad_J);
148
149     % INCREASE k WITH ONE
150     k = k+1;
151
152 end
```

calccost.m

```
1  % ------------------------------------------------------------------------
2  % RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: calccost.m
3  % ------------------------------------------------------------------------
4  % Usage: J = calccost(x, u, t, param_rh, x_eq);
5  %        Calculate the cost in the controller
6  %
7  % Input:  x: the previous simulated values of the states
8  %         u: the array with control output u
9  %         t: the array with time elements (equally spread!)
10 %         param_rh: the parameters of the r.h. controller
11 %         x_eq: the equilibrium state
12 % Output: J: the value of the cost
13 % ------------------------------------------------------------------------
14 function J = calccost(x, u, t, param_rh, x_eq)
15
16 %% START THE INTEGRATION
17 % The term that needs to be integrated
18 to_integrate = param_rh.q_as^2 .* ( x(1,:) - x_eq(1,:) ).^2 + param_rh.kappa_c^2 .* u.^2;
19 %
20 %
21 % Calculate the integral using trapz()
22 J = simpson(t, to_integrate) + param_rh.alpha_c/2 * (x(1,end) - x_eq(1,end))^2;
```

param_rhc.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: param_rhc.m
% ------------------------------------------------------------------------
% Usage: param_rh = param_rhc( parameters_used );
%        Set the parameters for the receding horizon controller
%
% Input:  vector with parameters:
%         [ q_as, kappa_c, alpha_c, Delta T_c, K_Delta ]
%         Value is equal to 999 if the default must be used (needed in the
%         parameter estimation)
% Output: struct with all parameters for the r.h.controller
% ------------------------------------------------------------------------
function param_rh = param_rhc( parameters_used );

%% DEFINE THE PARAMETERS USED IN THE RECEDING HORIZON CONTROLLER
% The parameters defined in the cost function of the controller
if (parameters_used(1) ~= 999)
    param_rh.q_as = parameters_used(1);
else
    param_rh.q_as = 0.4;
end

if (parameters_used(2) ~= 999)
    param_rh.kappa_c = parameters_used(2);
else
    param_rh.kappa_c = 0.3181;
end

if (parameters_used(3) ~= 999)
    param_rh.alpha_c = parameters_used(3);
else
    param_rh.alpha_c = 0.8;
end

% Set some time related parameters
if (parameters_used(4) ~= 999)     % The time of the control horizon
    param_rh.time_contr = round(parameters_used(4)/.005)*.005;
else
    param_rh.time_contr = 0.4;
end
if (parameters_used(5) ~= 999)     % The time of the predict horizon
    param_rh.time_sim = parameters_used(5) * param_rh.time_contr;
else
    param_rh.time_sim = 2 * param_rh.time_contr;
end

param_rh.samples = 200;            % The number of samples per minute

% Set the two equilibrium states (only P_as is realy interested)
param_rh.x_rest = [105 4 15 6 60 0 4 0 25 80]';
param_rh.x_exer = [122 4 15 6 90 0 4 0 17 105]';

% Set the boundaries for the bounded steepest decend method
param_rh.u_min = -30;
param_rh.u_max = 30;

% Set the noise parameters in procents
param_rh.noise_u = 3;
param_rh.noise_x = 6;

% Set the break condition parameters
param_rh.eps_abs = param_rh.time_contr^4;
param_rh.eps_rel = 0.001;
param_rh.eps_u = 0.1;
param_rh.k_max = 20;

% Set a debug variable and value for backtracking
param_rh.debug = 0;
param_rh.s = 2;
```

simpson.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: simpson.m
% ------------------------------------------------------------------------
% Usage: integral = simpson(t,f);
%        Estimate an integral using Simpson's rule
%
% Input:  t: the array with the time intervals
%         f: the array with the function values
% Output: integral: the value of the integral
% ------------------------------------------------------------------------
function integral = simpson(t,f)

%% Simpson's rule integrates using the formula:
% \int_a^b f(t)dt = \frac{b-a}{6} [ f(a) + 4f((a+b)/2) + f(b)]
% for a small interval (a, b).
%
% The value of f(b) has to be counted twice in larger intervals, because it
% occurs to be in the integral from a to b, and in the one from b to b+h,
% where h = (b-a)/2

% Estimate the length of the vector f and the timestep h
n = length(f) - 1;
h = t(2)-t(1);

% Set the sum zero in the beginning
sum = 0;

% The initial value of the vector y
y1 = f(1);

% Loop through the sum
for j=2:2:n
    y2 = f(j);
    y3 = f(j+1);
    sum = sum + (y1 + 4*y2 + y3);
    y1 = y3;
end

% Output the value of the integral
integral = sum * h / 3;
```

sim_lambda.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: sim_model.m
% ------------------------------------------------------------------------
% Usage: lambda = sim_lambda(t, param_rh, x, x_eq);
%        Simulate the model for the adjoint functions.
%
% Input:  t: the array with time elements (equally spread!)
%         param_rh: the paramters of the controller
%         x: the simulate values of the states
%         x_eq: the matrix with the equilibr. states
% Output: lambda: the output of lambda_1
% ------------------------------------------------------------------------
function lambda = sim_lambda(t, param_rh, x, x_eq)

%% START THE SIMULATION OF LAMBDA
% Estimate the timestep
h = (t(end) - t(1))/length(t);

% First create an matrix with empty lambdas (and also lambda(T))
lambda = zeros(size(x));
lambda(1,end) = -param_rh.alpha_c * (x(1,end) - x_eq(1,end));

```

```
% Express the value for the Q matrix
Q = diag ( [param_rh.q_as^2, 0, 0, 0, 0, 0, 0, 0, 0]');

% Calculate the two possible sets of paramaters for the model
parameters_rest = param_mod(1, param_rh.modparam);
parameters_exer = param_mod(2, param_rh.modparam);

% Run through the for loop from t + \Delta T -> t
for j = size(x,2)-1:-1:1
    % The old lines for Euler
    %lambda(j,:) = lambda(j+1,:)' + timestep .* calculate_Fx(x(j+1,:), parameters)' ...
    %                     * lambda(j+1,:)' - timestep .* 2*Q * (x(j+1,:)' - x_eq);

    % First set the good set of parameters
    if (x_eq(1,j+1) == 105)
        parameters = parameters_rest;
    end
    if (x_eq(1,j+1) == 122)
        parameters = parameters_exer;
    end

    % Calculate the four parts of the Runga Kutta method
    G1 = calculate_Fx(x(:,j+1), parameters)' * lambda(:,j+1) - 2*Q*(x(:,j+1) - x_eq(:,j+1));
    G2 = calculate_Fx( (x(:,j+1)+x(:,j))/2, parameters)' * ( lambda(:,j+1) + h/2 * G1 ) -...
         2 * Q * ( (x(:,j+1) + x(:,j))/2 - (x_eq(:,j+1) + x_eq(:,j))/2);
    G3 = calculate_Fx( (x(:,j+1)+x(:,j))/2, parameters)' * ( lambda(:,j+1) + h/2 * G2 ) -...
         2 * Q * ( (x(:,j+1) + x(:,j))/2 - (x_eq(:,j+1) + x_eq(:,j))/2);
    G4 = calculate_Fx(x(:,j), parameters)' * ( lambda(:,j+1) + h * G3 ) - ...
         2 * Q * ( x(:,j) - x_eq(:,j));

    % Combine those values to estimate the new value of lambda
    lambda(:,j) = lambda(:,j+1) + h/6 * (G1 + 2 * (G2 + G3) + G4);
end
```

calculate_Fx.m

```
% ---------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: calculate_Fx.m
% ---------------------------------------------------------------------------
% Usage: Fx = calculate_Fx(x, parameters);
%        This M-file calculates the F_x matrix used for
%        calculating the Lagrange multipliers.
% Input:  x: The previous calculated (or initial) values for the state x
%         parameters: All parameters of the model
% Output: Fx: The matrix F_x that is calculated
% ---------------------------------------------------------------------------
function Fx = calculate_Fx(x, parameters)

%% START OF THE CALCULATION OF THE F_X MATRIX
% Make some conversions of the states x to the real names:
P_as = x(1); P_vs = x(2); P_ap = x(3); P_vp = x(4); S_l = x(5);
sigma_l = x(6); S_r = x(7); sigma_r = x(8); R_s = x(9); H = x(10);

% First calculate the nonlinear functions t_d, k(H) and a(H)
t_d = 1 / sqrt(H) * (1 / sqrt(H) - parameters.kappa);
k_l = exp(-t_d / (parameters.R_l * parameters.c_l));
a_l = 1 - k_l;
k_r = exp(-t_d / (parameters.R_r * parameters.c_r));
a_r = 1 - k_r;

% Then calculate the F_x matrix:
Fx(:,1) = [ 1 / parameters.c_as * ( (-H * parameters.c_l * a_l^2 * P_vp * S_l)/ ...
                                        (a_l * P_as + k_l * S_l)^2 - 1 / R_s ) ;
            1 / (R_s * parameters.c_vs) ;
            0 ;
            1 / parameters.c_vp *  ( H * parameters.c_l * a_l^2 * P_vp * S_l)/ ...
                                        (a_l * P_as + k_l * S_l)^2 ;
            0 ;
            0 ;
```

```
            0 ;
            0 ;
            1 / parameters.K * (parameters.A_pesk * parameters.C_aO2 / R_s - 1) ;
            0 ; ];

Fx(:,2) = [ 1 / (R_s * parameters.c_as) ;
            1 / parameters.c_vs * (-1 / R_s - H * parameters.c_r * a_r * S_r / ...
                                    (a_r * P_ap + k_r * S_r) ) ;
            1 / parameters.c_ap * H * parameters.c_r * a_r * S_r / ...
                                    (a_r * P_ap + k_r * S_r) ;
            0 ;
            0 ;
            0 ;
            0 ;
            0 ;
            - 1 / parameters.K * ( parameters.A_pesk * parameters.C_aO2 / R_s - 1);
            0 ; ];

Fx(:,3) = [ 0 ;
            1 / parameters.c_vs * H * parameters.c_r * a_r^2 * P_vs * S_r / ...
                                    (a_r * P_ap + k_r * S_r)^2 ;
            1 / parameters.c_ap * ( - H * parameters.c_r * a_r^2 * P_vs * S_r / ...
                                (a_r * P_ap + k_r * S_r)^2 - 1 / parameters.R_p ) ;
            1 / (parameters.R_p * parameters.c_vp) ;
            0 ;
            0 ;
            0 ;
            0 ;
            0 ;
            0 ; ];

Fx(:,4) = [ 1 / parameters.c_as * H * parameters.c_l * a_l * S_l / ...
                                    (a_l * P_as + k_l * S_l) ;
            0 ;
            1 / (parameters.R_p * parameters.c_ap) ;
            1 / parameters.c_vp * ( - 1 / parameters.R_p - H * parameters.c_l ...
                                    * a_l * S_l / (a_l * P_as + k_l * S_l) ) ;
            0 ;
            0 ;
            0 ;
            0 ;
            0 ;
            0 ; ];

Fx(:,5) = [ 1 / parameters.c_as * H * parameters.c_l * a_l^2 * P_vp * P_as / ...
                                    (a_l * P_as + k_l * S_l)^2 ;
            0 ;
            0 ;
            - 1 / parameters.c_vp * H * parameters.c_l * a_l^2 * P_vp * P_as / ...
                                    (a_l * P_as + k_l * S_l)^2 ;
            0 ;
            - parameters.alpha_l ;
            0 ;
            0 ;
            0 ;
            0 ; ];

Fx(:,6) = [ 0; 0; 0; 0; 1; -parameters.gamma_l ; 0; 0; 0; 0; ];

Fx(:,7) = [ 0 ;
            - 1 / parameters.c_vs * H * parameters.c_r * a_r^2 * P_vs * P_ap / ...
                                    (a_r * P_ap + k_r * S_r)^2;
            1 / parameters.c_ap * H * parameters.c_r * a_r^2 * P_vs * P_ap / ...
                                    (a_r * P_ap + k_r * S_r)^2;
            0 ;
            0 ;
            0 ;
            0 ;
            - parameters.alpha_r ;
            0 ;
            0 ; ];

Fx(:,8) = [ 0; 0; 0; 0; 0; 0; 1; -parameters.gamma_r ; 0; 0; ];
```

```
107
108 Fx(:,9) = [ (P_as - P_vs) / (R_s^2 * parameters.c_as) ;
109              - (P_as - P_vs) / (R_s^2 * parameters.c_vs) ;
110              0 ;
111              0 ;
112              0 ;
113              0 ;
114              0 ;
115              0 ;
116              - parameters.A_pesk * parameters.C_aO2 * (P_as - P_vs) / ...
117                                           (R_s^2 * parameters.K) ;
118              0 ; ];
119
120 Fx(:,10) = [ 1 / parameters.c_as * ( (parameters.c_l * a_l * P_vp * S_l)/ ...
121                         (a_l * P_as + k_l * S_l) + (parameters.c_l * P_vp * S_l^2)/ ...
122                         (parameters.R_l * parameters.c_l * (a_l * P_as + k_l * S_l)^2) ...
123                         * (parameters.kappa / 2 * sqrt(H) - 1 / H) * exp (- t_d / ...
124                         ( parameters.R_l * parameters.c_l ) ) );
125               -1 / parameters.c_vs * ( (parameters.c_r * a_r * P_vs * S_r)/ ...
126                         (a_r * P_ap + k_r * S_r) + (parameters.c_r * P_vs * S_r^2)/ ...
127                         (parameters.R_r * parameters.c_r * (a_r * P_ap + k_r * S_r)^2) ...
128                         * (parameters.kappa / 2 * sqrt(H) - 1 / H) * exp (- t_d / ...
129                         ( parameters.R_r * parameters.c_r ) ) );
130               1 / parameters.c_ap * ( (parameters.c_r * a_r * P_vs * S_r)/ ...
131                         (a_r * P_ap + k_r * S_r) + (parameters.c_r * P_vs * S_r^2)/ ...
132                         (parameters.R_r * parameters.c_r * (a_r * P_ap + k_r * S_r)^2) ...
133                         * (parameters.kappa / 2 * sqrt(H) - 1 / H) * exp (- t_d / ...
134                         ( parameters.R_r * parameters.c_r ) ) );
135               -1 / parameters.c_vp * ( (parameters.c_l * a_l * P_vp * S_l)/ ...
136                         (a_l * P_as + k_l * S_l) + (parameters.c_l * P_vp * S_l^2)/ ...
137                         (parameters.R_l * parameters.c_l * (a_l * P_as + k_l * S_l)^2) ...
138                         * (parameters.kappa / 2 * sqrt(H) - 1 / H) * exp (- t_d / ...
139                         ( parameters.R_l * parameters.c_l ) ) );
140               0 ;
141               parameters.beta_l ;
142               0 ;
143               parameters.beta_r ;
144               0 ;
145               0 ; ];
```

test_AD.m

```
1  % -------------------------------------------------------------------------
2  % RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: test_AD.m
3  % -------------------------------------------------------------------------
4  % Usage: test_AD()
5  %        Example how to use the Automatic differentiation toolbox.
6  %
7  % Input:  None
8  % Output: None
9  % -------------------------------------------------------------------------
10
11 %% START THE MATLAB M-file
12
13 % Set the parameters, the input and state values at some time moment t
14 parameters = param('rest');
15 u = 23;
16 x = [117.9532 4.3113 12.8487 5.2619 63.4276 -5.1138 3.9966 0.0019 23.8777 80];
17
18 % Create a TomLab state vector x. We want to have the derivative of those x
19 % vector to all states x, so we have to apply a speye (sparse eye) of the
20 % length of this vector x (which is ten):
21 x = fmad(x, speye(10));
22
23 % Make some conversions of the states x to the real names:
24 P_as = x(1); P_vs = x(2); P_ap = x(3); P_vp = x(4); S_l = x(5);
25 sigma_l = x(6); S_r = x(7); sigma_r = x(8); R_s = x(9); H = x(10);
26
27 % First calculate the nonlinear functions t_d, k(H) and a(H)
```

```
t_d = 1 / sqrt(H) * (1 / sqrt(H) - parameters.kappa);
k_l = exp(-t_d / (parameters.R_l * parameters.c_l));
a_l = 1 - k_l;
k_r = exp(-t_d / (parameters.R_r * parameters.c_r));
a_r = 1 - k_r;

% Calculate the differential equations
dP_as = 1 / parameters.c_as * ( H * (parameters.c_l * a_l * P_vp * S_l ) / ...
        (a_l * P_as + k_l * S_l) - 1 / R_s * (P_as - P_vs) );

dP_vs = 1 / parameters.c_vs * ( 1 / R_s * (P_as - P_vs) - ...
        H * (parameters.c_r * a_r * P_vs * S_r ) / (a_r * P_ap + k_r * S_r));

dP_ap = 1 / parameters.c_ap * ( H * (parameters.c_r * a_r * P_vs * S_r) / ...
        (a_r * P_ap + k_r * S_r) - 1 / parameters.R_p * (P_ap - P_vp));

dP_vp = 1 / parameters.c_vp * (1 / parameters.R_p * (P_ap - P_vp) - ...
        H * (parameters.c_l * a_l * P_vp * S_l) / (a_l * P_as + k_l * S_l));

dS_l = sigma_l;

dsigma_l = parameters.beta_l * H - parameters.alpha_l * S_l - parameters.gamma_l * sigma_l;

dS_r = sigma_r;

dsigma_r = parameters.beta_r * H - parameters.alpha_r * S_r - parameters.gamma_r * sigma_r;

dR_s = 1 / parameters.K * (parameters.A_pesk * ( (P_as - P_vs) / R_s * parameters.C_aO2 -...
       (parameters.M_0 + parameters.rho * parameters.W)) - (P_as - P_vs));

dH = u;

% Create the differential equations
dx(1) = dP_as;
dx(2) = dP_vs;
dx(3) = dP_ap;
dx(4) = dP_vp;
dx(5) = dS_l;
dx(6) = dsigma_l;
dx(7) = dS_r;
dx(8) = dsigma_r;
dx(9) = dR_s;
dx(10) = dH;

% TomLab always evaluates the normal evalation and that one of the
% derivative. We're interesten in the last one, so just get this value out
% of the algorithm using the command getderivs().
ddx = getderivs(dx);

% We want to reshape the results, to get the Fx matrix we are interested
% in.
Fx = reshape(ddx, [10 10]);
```

update_u.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: update_u.m
% ------------------------------------------------------------------------
% Usage: [u, s_k] = update_u(x0, x_eq, param_rh, J, u_old, t_sim, grad_J,
%                                                              norm_grad_J);
%        Update the value of the controller output
%
% Input:  trivial
% Output: trivial
% ------------------------------------------------------------------------
function [u, s_k] = update_u(x0, x_eq, param_rh, J, u_old, t_sim, grad_J, norm_grad_J)

%% START THE UPDATING
% Also some small constant c is needed for the algorithm
```

```
c = 10^-4;

% Enter the update loop for the backtracking
while (1)
    % Determine some new value for u (with some value for s)
    u = u_old - param_rh.s * grad_J;

    % Use the projection of this u to the boundaries for u (u_min and u_max)
    u_p = (u < param_rh.u_max & u > param_rh.u_min) .* u + (u < param_rh.u_min)...
                    .* param_rh.u_min + (u > param_rh.u_max) .* param_rh.u_max;
    u = u_p;

    % Calculate the norm (squared) of u_old - u_p for the back. algorithm
    norm_u = simpson(t_sim, abs(u_p - u_old).^2);

    % Calculate the values for the states and the cost with this new u
    x = sim_model(t_sim, u, x0, param_rh.eq, param_rh);
    J_new = calccost(x, u, t_sim, param_rh, x_eq);

    % Check if the value for s is correct, and if not divide it by two
    if (J_new <= J - param_rh.s*c* norm_u) % norm_grad_J)% (old value instead of norm_u)
        % The backtracking algorithm condition holds! Break...
        break;
    else
        % Divide the value for s by two
        param_rh.s = param_rh.s / 2;
    end
end
s_k = param_rh.s;
```

paramest.m

```
% ------------------------------------------------------------------------
% RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: paramest.m
% ------------------------------------------------------------------------
% Usage: paramest()
%
% Input:  None
% Output: None
% ------------------------------------------------------------------------

% Clear everything
clear all; close all;

% Display the start time of simulation
time_start = datestr(now)

% Set the options for the optimiser:

options = optimset('Display', 'iter', ...       % Display info each iter.
          'Diagnostics', 'on', ...
          'MaxIter', 100, ...                   % Maximal number of iter.
          'Tolfun', 1e-3, ...                   % Tolerance in cost func.
          'DiffMinChange', 1e-8, ...
          'DiffMaxChange', 1e-2, ...
          'TolX', 1e-5);                        % Tolerance in parameter

% Call the fmincon optimizer:
[ param, ...        % Parameter(s) found by the optimizer
  Jval,  ...        % The cost value with these parameters
  exit,  ...        % The exit flag of the optimizer
  overv, ...        % Some overview of the optimisation
  lambda,...        % Lagrange multipliers
  grad ...          % Gradient at solution param
            ] = fmincon(@(param)costfun(param), ...
                [0.2882, 0.5934, 0.3177, 0.0157, 0.0487],...% Init. values for the params
                [], ...             % A: No constraints (Ax < b)
                [], ...             % b: ""
                [], ...             % B: No constraints (Bx = c)
```

```
38                  [], ...             % c: ""
39                  [0.01, 0.01, 0.01, 0.01, 0.01], ... % Lower bound parameters
40                  [0.2, 0.2, 1, 0.12, 0.12], ...      % Upper bound parameters
41                  [], ...             % No nonlin. ineq. constraints
42                  options);           % The options defined before
43
44 % Set the end time of the simulation
45 time_end = datestr(now)
```

costfun.m

```
1  % --------------------------------------------------------------------------
2  % RECEDING HORIZON CONTROLLER - MATLAB FUNCTION: costfun.m
3  % --------------------------------------------------------------------------
4  % Usage: J = costfun(x);
5  %        Cost function FOR THE PARAMETER ESTIMATION
6  %
7  % Input:  x: the values for the parameters that have to be estimated
8  % Output: J: the cost for this set
9  %
10 % NOTE: The array with which parameters have to be estimated has to be
11 %       defined IN the M-file! (Because SolvOpt can't deal with it)
12 % --------------------------------------------------------------------------
13 function J = costfun(x)
14
15 %% THE FUNCTION THAT NEEDS TO BE MINIMIZED
16 % Copy the parameters from x into a variable param and param_save (because
17 % x will be overwritten)
18 param = x
19 param_save = x;
20
21 % Set some bounds for the parameters (because SolvOpt can't...)
22 b1 = 0.01;          % Lowerbound
23 b2 = 1.2;           % Upperbound
24 for i = 1:3
25     if (param(i) > b2)
26         param(i) = b2;
27     end
28     if (param(i) < b1)
29         param(i) = b1;
30     end
31 end
32
33 % IMPORTANT: The array with the parameters that need to be investigated !!!
34 % [q_as, alpha_c, kappa_c, Delta T_c, K_Delta, c_l, c_r] (999 is default)
35 param_inv = [1,2,3,6,7];
36
37 % Load the dataset
38 load 'peer1_trans';                    % Measurements 'peer1'
39 % load 'data_simresults_04_03181_08.mat'; % Ideal data with params as in
40
41 %% Convert the parameters to one array, where a parameter set to 0 will have
42 % the default value specified in the model description.
43 % parameters_used = [ q_as, kappa_c, alpha_c, delta_T_c, fac_delta_T_h, ..]
44 parameters_used = ones(1,7).*999;
45 for i=1:length(param_inv)
46     parameters_used(param_inv(i)) = param(i);
47 end
48
49 %% Simulate the model with the receding horizon controller:
50 % Set the parameters for the controller using the changes done by the
51 % fmincon() function:
52 param_rh = param_rhc(parameters_used);
53
54 % Set two possible parameters in the model for estimation:
55 param_rh.modparam = parameters_used(6:7);
56
57 % Set some initial values for x0, t_sim and the transitions:
58 x0 = [105.0767, 4.3015, 12.2508, 4.9232, 60.9315, ...
```

```
       0.0205, 3.8376, 0.0001, 22.0047, 76.8127]';

t_sim = 5;
t = 0;
numit = ceil(t_sim / param_rh.time_contr);
param_rh.eq = [0, 1];
x_total = [];
for i = 1 : numit
    param_rh.t0 = t(end);
    [t, u, simres] = recedinghorizon(x0, param_rh);
    x = simres.x;
    x0 = x(:,end);
    x_total = [x_total , x(:,1:end-1)];
end
x0 = mean(x_total(:,end-3:end)')';

% x0 = [122 4 15 6 90 0 4 0 17 105]';
% x0 = [121.7808, 3.7156, 10.9826, 8.3589, 79.6893, ...
                          -0.0122, 5.0180, -0.0002, 14.3997, 100.4369]';

% Set the simulation time, initial time & number of iterations
t_sim = 6;
t = 0;
numit = ceil(t_sim / param_rh.time_contr);

% Set the changes in the eq. states which are visible in the data-sets
param_rh.eq = [0, 1 ;
               2, 2 ];
% param_rh.eq = [0, 2 ];


% Define two empty arrays
x_total = [];
t_total = [];

% Do the simulations
for i = 1 : numit
    % Set the starting time of this iteration
    param_rh.t0 = t(end);
    % Calculate the receding horizon output with the given x0 and param_rh
    [t, u, simres] = recedinghorizon(x0, param_rh);
    % Save the time array
    t_total = [t_total , t(1:end-1)];
    % Save the x array and set the last value as new initial condition
    x = simres.x;
    x0 = x(:,end);
    % Save the array into the big array
    x_total = [x_total , x(:,1:end-1)];
end

%% Integrate the difference between the simulation and measurement results:
% Set the maximal length for x, just in case if there goes something
% wrong...
maximal = length(P_as);

% Calculate the real cost function of this optimisation
x = param_save;
J1 = simpson(time, abs(P_as(1:maximal) - x_total(1,1:maximal)) ...
                                + abs(H(1:maximal) - x_total(10,1:maximal)));
J2 = 200 * (max(0,x(1) - b2) + max(0,-x(1) + b1) + max(0,x(2) - b2) + ...
                    max(0, -x(2) + b1) + max(0,x(3) - b2) + max(0,-x(3)+b1));
J = J1 + J2
```

BIBLIOGRAPHY

[1] J.J. Batzel, F. Kappel, D. Schneditz, and H.Tran.
Cardiovascular & Respiratory Systems: Modeling, Analysis & Control. SIAM, 2007.

[2] D.M. Prett C.E. Garcia and M. Morari. Model predictive control: Theory and practice - a survey. *Automatica*, 25:335–348, 1989.

[3] Shaun A. Forth and Marcus M. Edvall.
User Guide for Mad - A Matlab Automatic Differentiation Toolbox
TOMLAB/MAD Version 1.1 - The Forward Method. TOMLAB Optimization, 2003.

[4] Franz Kappel and Alexei V. Kuntsevich. An implementation of shor's r-algorithm. *Computational Optimization and Applications*, 15:193–205, 2000.

[5] M. Mutsaers M. Bachar J. Batzel F. Kappel and S. Volkwein. Receding horizon controller for the baroreceptor loop in a model for the cardiovascular system. *SFB-Report No 2007-001*, 2007.

[6] C.T. Kelley.
Iterative Methods for Optimization. SIAM, 1999.

[7] C.S. Peskin. *Lectures on mathematical aspects of physiology, in F.C. Hoppensteadt (ed.)*
Mathematical Aspects of Physiology, volume 19. American Mathematical Society, 1981.

[8] F. Allgower T. Badgwell J. Qin J. Rawlings and S. Wright. Nonlinear predictive control and moving horizon estimation - an introductory overview. *Advances in control, highlights of ECC'99*, pages 391–449, 1999.

[9] Arun Verma. An introduction to automatic differentiation. *Current Science*, 78(7):804–807, April 10th 2000.