Computer Modeling of the Cardiovascular System and Blood Pressure Regulation

by

Siri Kallhovd

THESIS

for the degree of

Master of Science

(Master i Anvendt matematikk og mekanikk)

Faculty of Mathematics and Natural Sciences
University of Oslo

October 2012

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Acknowledgements

I would first and foremost like to thank my supervisor, Joakim Sundnes at Simula Research Laboratory for his support and guidance during the work on my master thesis.

I would also like to thank Karin Toska at for providing one of the programs used as a starting point for my thesis. Discussing with her gave insight into the motivation behind the previous work on this subject.

I want to thank my family for their support throughout the year, and especially Gurli and Kjetil for letting me work at their kitchen table. Kjetil and Jenny also deserves thanks for looking through the thesis for grammatical errors.

Contents

Chapter 1

Introduction

The cardiovascular system consists of the heart and the vessels. The pressure in the arteries is regulated by the baroreflex, which means pressure reflex. The main purpose of this thesis is to couple a baroreflex model by Toska et al [4, 14] that previously was connected to a simple Windkessel model, to a closed loop circulation model with a four-chamber elastic heart that models pulsatile blood flow [1]. The latter model is called Circ_with_Baroreceptors, or CwB. The CwB model was written in JSim's MML (Mathematical Modeling Language), and the Toska model was programmed in Matlab. Both of these were converted into Python in the beginning of the project.

We will introduce common and specific models for resistance, flow, pressure, volume and compliance of blood vessels. We will particularly focus on how the baroreflex regulate the arterial blood pressure. The baroreflex is a part of the central nervous system.

The Toska model was made with the intension to optimize parameters of a baroreflex model to individuals. We will test the Toska model against measurements, for randomized start parameters. Originally it had only been tested against two parameter sets for each subject. We want to find out how stable the results from the optimizing process are.

When we connect the Toska baroreflexes to the pulsatile CwB circulation, we need to make some adjustments to the model to retain the original intent. This is particularly important for regulating the force of contraction. In the CwB heart model the compliance of the heart muscle varies over the heart beat. The Toska circulation involves average values beat-to-beat.

We compare the CwB model with both original and Toska baroreflexes, to find if the Toska model works when it comes to regulating the CwB circulation. We are interested in whether the Toska barorefex parameter values that fit the CwB circulation are within the values that were optimized when running the original Toska model for the individual measurements.

Chapter 2

Circulation models

Any model of the circulatory system will make choices about the level of detail that is desirable or practical, depending on what we want to study. A brief physiological overview of the circulation system shows that blood flows from the heart into the arteries, then enters the capillaries where the gas exchange occurs, and is finally transported back to the heart through the veins. The vessels that take blood depleted of oxygen from the right ventricle of the heart to the lungs, make up the pulmonary circulation. The vessels that take oxygen rich blood from the left ventricle out to all the cells in the body, are called the systemic circulation.

To each kind of vessel one assigns different properties to get the right behaviour. The resistance to flow in a vessel is dependant on viscosity of the blood and the radius of the vessel. Vessels with smaller radius have a larger resistance. When a pulse of blood enters an artery the compliant walls makes the vessel expand, and when the flow into the arteries diminish between heart beats, the loss of pressure will make the walls fall back, pushing flow through to the rest of the system. This is called elastic recoil, [10] The amplitude of the pressure pulse decreases throughout the circulation due to friction forces. In the veins the flow is almost steady.

Mathematical formulas can either be based on theoretical considerations of flow through vessels of given properties, or curve fitting based on results from experiments. Regarding the two main models of this thesis, we note that the CwB model has separated a number of different vessels, while the Toska model lacks a pulmonary circulation.

The overall purpose of this section is to present the building blocks that together make up the system of ordinary differential equations (ODEs) in the case of the CwB model, and the beat-to-beat discrete algebraic equations of the Toska model.

The CwB model source code presents some reference equations, among them equations for linear resistance and compliance vessels. To some of these equations one can find mathematical analogues to formulas for electric

circuit components, which makes it easier to make comprehensible drawings of different circulation models.

2.1 Linear resistance and compliance vessels

We want to work with simple vessels, so for instance we lump together all the systemic capillaries into one vessel and make a choice whether it is resistance to flow or compliant vessel wall that is most important to describe for the vessel.

A resistance vessel has inflexible walls so that the blood volume it can contain is fixed. This means that the flow into the vessel must be equal to the flow out of the vessel. The flow, Q through a resistance vessel depends linearly on the pressure drop, ΔP

$$Q = \frac{\Delta P}{R} \tag{2.1}$$

and the resistance, R is the opposition to this flow.

This is analogous of Ohm's Law, which states that the current is directly proportional to the electric potential drop over the wire/resistor. A resistor is only said to obey Ohm's Law if the resistance is constant, see for instance [15, p. 950-951].

In a compliance vessel the walls are flexible. The volume in the vessel depends on the pressure, and the compliance is a measure of how much the walls can stretch. A common assumption, see for instance Hoppensteadt and Peskin [6, p. 15-16], is that the resistance to blood flow is negligible, and it follows that the pressure drop over a compliance vessel is zero. Since the vessel walls are elastic, a given pressure, P determines the blood volume, V the vessel can contain. We have

$$V = CP \tag{2.2}$$

where C is the compliance. Again we have an electrical analogue, which is the capacitor. A capacitor is formed by two conductors separated by an insulator, for instance two parallel metal plates with air in between, as explained in [15, p. 909-910]. A major difference between the two systems is that in the electrical case there will be no accumulation of charge if there is no potential difference over the capacitor, but a vessel will contain some blood even if the pressure is zero. To account for this addition we write

$$V = V_d + CP \tag{2.3}$$

where V_d is the remaining "dead" volume if there is no pressure.

Flow through a compliance vessel can also be derived from a simplification of the Stoke's equation, as described by Keener and Sneyd [7, p.

474-478]. The vessel is considered to be a cylinder, and the flow is given by

$$Q \propto \frac{dP}{dx} A^2. \tag{2.4}$$

This is called Poiseuillle's law. The cross sectional area, A is assumed to depend linearly on the pressure. The vessel of length, L then has a pressure drop, and a resistance.

$$RQ = \frac{1}{3\gamma}(1+\gamma P)^3|_{P_1}^{P_0}, \tag{2.5}$$

$$\frac{V}{V_d} = \frac{3}{4}\left[\frac{(1+\gamma P)^4|_{P_1}^{P_0}}{(1+\gamma P)^3|_{P_1}^{P_0}}\right]. \tag{2.6}$$

Where $\gamma = c/A_d$, and c is compliance per length and A_d is the cross sectional area of the vessel at zero pressure.

By including only linear terms, we still get a linear model for the volume of the compliance vessel,

$$RQ = P_0 - P_1, \tag{2.7}$$

$$\frac{V}{V_d} = 1 + \frac{\gamma}{2}(P_0 + P_1). \tag{2.8}$$

If we also eliminate the pressure drop over the vessel, we end up with equation (2.3). If we instead assume that A was constant when we integrate equation (2.4) then we get a linear resistance vessel with constant volume.

2.1.1 The heart as a compliance vessel

When modelling the circulation, a heart model must be included. It is common to model the heart as a compliance vessel where the compliance varies with the heart cycle. We consider a model taken from the book by Hoppensteadt and Peskin [6, p. 11-12]. This model is only concerned with the ventricles, and neglects the influence of the smaller atria.

During contraction and ejection (systole) the heart walls are stiff and compliance is small, almost negligible. During filling (diastole) the walls are relaxed and there is a large compliance. This variation can be described by making the compliance a prescribed, periodic function of time. We have

$$V(t) = V_d + C(t)P(t) \tag{2.9}$$

At the end of the filling, when the inflow valve closes, the heart volume is at its maximum. This is called end diastolic volume, V_{ED}. Before the valve closes the pressure in the ventricle is essentially the same as the adjacent vein, P_v and we call the compliance at this time C_d.

$$V_{ED} = V_d + C_d P_v \tag{2.10}$$

During contraction the pressure increases inside the heart until it is slightly above the pressure in the arteries, P_a, which then pushes open the outflow valve. The valve stays open until the ventricular pressure falls slightly below the arterial pressure, P_a. When the outflow valve closes we find the minimum volume of the cycle, the end systolic volume, V_{ES}. We have

$$V_{ES} = V_d + C_s P_a \tag{2.11}$$

where the systolic compliance is C_s.

From this follows the stroke volume, V_{stroke} which is the ejected volume per heart beat.

$$V_{stroke} = V_{ED} - V_{ES} = C_d P_v - C_s P_a \tag{2.12}$$

When we know the heart rate, F (frequency) and V_{stroke} which is volume per beat, we can find the cardiac output, Q which models total flow out of the heart. To further simplify the formula for cardiac output, we take the systolic compliance to be so low, that we can neglect the influence of the end systolic volume. We have

$$Q = FV_{stroke} = FC_d P_v \quad \text{if} \quad C_s \approx 0 \tag{2.13}$$

This phenomenon, that cardiac output increases with increased filling, achieved by higher venous pressure, is commonly referred to as *Starling's law* [7, p. 484]. By modeling the heart in this way, we get a steady state situation, where there are no fluctuations. The result is that all variables can be considered as average values over a heart beat.

2.1.2 Connecting vessels into a small circulation

The model components introduced above can be built into a steady state circulation with two compliance vessels and one resistance vessel in both the pulmonary and the systemic system, as presented in [6, p. 15-16]. Here C_{ld} and C_{rd} are the diastolic compliances of the left and right ventricle respectively. The equations that make up the model can be summarised as follows.

	Systemic	Pulmonary
Heart	$Q_L = FC_{ld}P_{pv}$	$Q_R = FC_{rd}P_{sv}$
Arteries	$V_{sa} = C_{sa}P_{sa}$	$V_{pa} = C_{pa}P_{pa}$
Capillaries	$Q_s = \frac{P_{sa}-P_{sv}}{R_s}$	$Q_p = \frac{P_{pa}-P_{pv}}{R_p}$
Veins	$V_{sv} = C_{sv}P_{sv}$	$V_{pv} = C_{pv}P_{pv}$

(2.14)

Since this is a steady state, non pulsatile flow, the flow into the vessel must be the same as the outgoing flow. If not, the system would not be in equilibrium, and volume would accumulate in parts of the system.

$$Q = Q_R = Q_L = Q_s = Q_p \tag{2.15}$$

On the time scale considered here, the total blood volume, V_t in the body will be constant.

$$V_t = V_{sa} + V_{sv} + V_{pa} + V_{pv} \tag{2.16}$$

The model equations in (2.14) can now be solved for the nine unknowns which are the volumes, the pressures, and the cardiac output, Q. All other properties in the model are considered to be known parameters. These can be estimated from measured resting values of volumes, pressures and flow. Even if this model is a highly simplified representation of the circulation, it can give useful insights into how changes in the parameters influence the system.

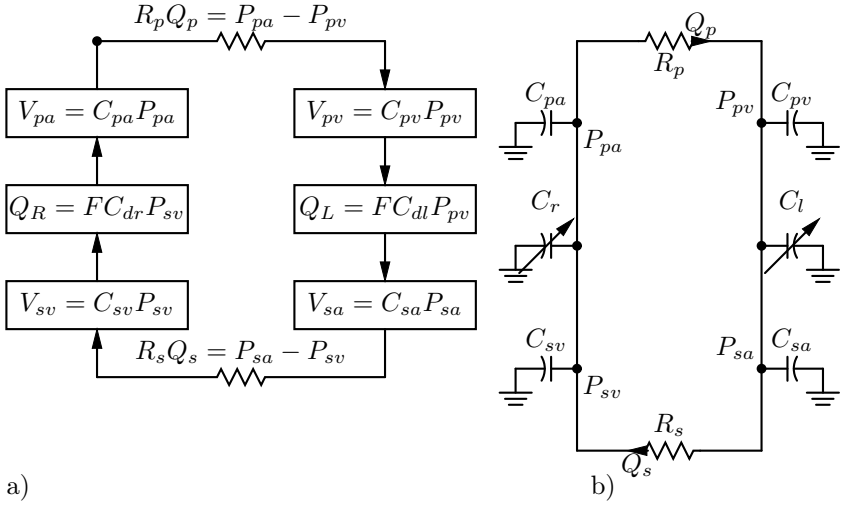


Figure 2.1: a) The Hoppensteadt and Peskin circulation as seen in [6, p. 15] , b) is the same circulation portrayed using electric circuit components.

The next system we consider is the simplest circulation found in the book by Keener and Sneyd [7, p. 491-492]. Given the definition of a compliance vessel with resistance, this circulation contains only two compliance vessels in the systemic system and one compliance vessel in the pulmonary system.

The equations in the model can be summarized as

	Systemic	Pulmonary
Heart	$Q = FC_{ld}P_{pv}$	$Q = FC_{rd}P_{sv}$
Arteries	$\begin{cases} Q = \frac{P_{sa}-P_s}{R_{sa}} \\ V_{sa} = V_d^s + \frac{C_{sa}}{2}(P_{sa}+P_s) \end{cases}$	
Arteries/Veins		$\begin{cases} Q = \frac{P_{pa}-P_{pv}}{R_p} \\ V_p = V_d^p + \frac{C_p}{2}(P_{pa}+P_{pv}) \end{cases}$
Veins	$\begin{cases} Q = \frac{P_s-P_{sv}}{R_{sv}} \\ V_{sv} = \frac{C_{sv}}{2}(P_s+P_{sv}) \end{cases}$	

(2.17)

The total blood volume, V_t in the body again has to be constant.

$$V_t = V_{sa} + V_{sv} + V_p \tag{2.18}$$

Given that we still assume a constant cardiac output, we have a steady state, non pulsatile system. The model is illustrated in Figure 2.2. Since the compliance equations here do not directly translate into electrical analogues, we can not make a corresponding circuit diagram in this case.

The obvious extension of these models, is to connect a heart that gives pulsatile outflow. In this case the compliance vessels, by their nature of being compliant, will have a changing volume, and the algebraic relations are changed into an a system of ODEs, ordinary differential equations.

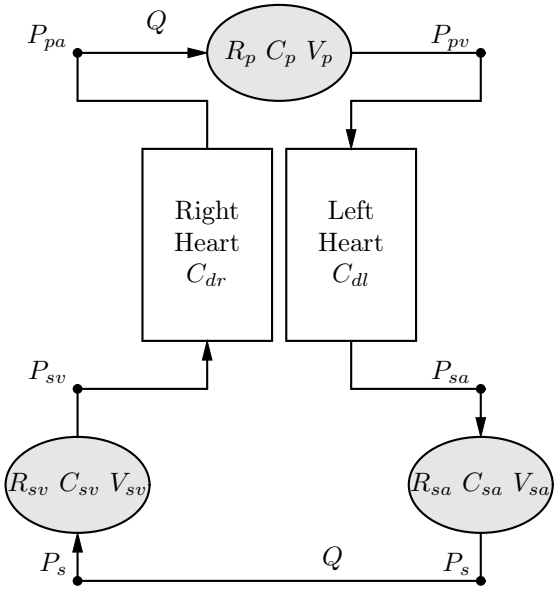


Figure 2.2: The Keener and Sneyd circulation as seen in [7, p. 492]

2.2 The CwB circulation model

We chose the model number 0095 (called *Circ_with_Baroreceptors*) in the JSim project [1], henceforth referred to as the CwB circulation model. For visualisation, Figure A.1 in the appendix is a rendition of the drawing in the MML source code of this model, with some corrections. The model consists of a four-chamber elastic heart, pulmonary and systemic circulation with additional branches for coronary and cerebral circulation, the pericardium and baroreflex. The CwB model consists of 36 first order, and one second order ordinary differential equations. The second order equation is for the baroreceptor signal and is decomposed into two first order equations. For most vessels and the heart chambers, the state variable is the volume, there are 24 equations for the change in these volumes. To model inertia in the aorta and the pulmonary artery, we have four equations where the state variable is the flow through the vessel. The model for the aorta also include two pressure state variables. Two state variables gives approximations for the mean arterial pressure and the cardiac output. In addition to the baroreceptor signal there are four delay-differential equations for the signal pathways of the baroreflex. The baroreflex is explained in Section 3.2. Eventually we want to examine the differences between this model and the baroreflex model by Toska et al [14].

2.2.1 Linearity is a simplification

We will examine how the CwB circulation model is put together. Throughout this section, it is helpful to check Figure A.1 for branches and relations between vessels. Most of the vessels in this model consist of one compliance vessel connected by one resistance vessel, to ensure that each vessel is both compliant and has some resistance to flow. This is different approach than in the model from Keener and Sneyd [7] presented in (2.17), where one vessel component has both resistance and compliance properties.

In many vessels both the resistance and compliance relations are chosen to be linear. The compliance relation is in the form of equation (2.2), but in addition there is included a non-linear term that only becomes important at small volumes, instead of simply setting a "dead" volume, see equation (2.26). These mostly linear vessels include all vessels in the systemic arterioles and capillaries, the coronary circulation, and the pulmonary arterioles, capillaries and veins.

Some bigger vessels in the CwB model are modeled as completely non-linear compliance relations. This includes the heart, the aorta, the systemic arteries, the systemic veins, the vena cava and the pulmonary arteries. The pressure in these vessels still depend on the volume

$$P_i = P_i[V_i] \tag{2.19}$$

The vena cava and the systemic arteries also have non-linear resistances which depend on the volume.

$$R_i = R_i[V_i] \tag{2.20}$$

We will comment on the justifications of each vessel's modeling as presented in the original article, [8].

For the vessels that are close to the thorax, that is the cavity where we find the heart and lungs, we have an external pressure, P_{ex} that we need to consider. Inside pleural membrane that encloses the lungs we have subatmospheric pressure. For the heart we also include the effect of the pericardial pressure, that depends on the entire volume of the heart. We will come back to this in Section 2.2.5. The chamber pressure, P_c inside of the vessels, influenced by an external pressure is

$$P_c = P_{TM} + P_{ex} \tag{2.21}$$

where P_{TM} is the transmural pressure across the vessel wall set up by the volume inside the vessel. This is explained in [9]. The pressures that we have considered until now are transmural pressures.

2.2.2 Resistance vessels

The most straight forward example of a pure resistance vessel in this model is the cerebral circulation. It is modelled as a single linear resistance vessel, with inflexible walls. The flow through this cerebral vessel is Q_{cbr}, and the resistance is R_{cbr}. The blood vessels that go to the brain branch off from the heart.

$$Q_{cbr} = \frac{P_{aodc} - P_{vcc}}{R_{cbr}} \tag{2.22}$$

The purpose of this simple vessel is to get the right fraction of the blood volume to pass through the brain. The pressures in this equation are those of the distal part of the aorta, P_{aodc}, and at the vena cava, P_{vcc}. These vessels are both connected to the heart.

The heart valves are also modelled as linear resistance vessels, but with a switch that cuts off the flow when the pressure conditions dictate that the valves are closed. For instance the mitral valve is open when the pressure in the left atrium, P_{lac} is higher than the pressure in the left ventricle, P_{lvc}, and closed otherwise

$$Q_{la} = \begin{cases} \frac{P_{lac} - P_{lvc}}{R_{la}} & \text{if } P_{lac} > P_{lvc}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

The flow out of the left atrium, Q_{la} is determined by the resistance, R_{la} through the valve. In a healthy heart, valve resistances are low.

In a real vessel the smooth muscle in the walls can contract or relax according to how much blood the tissue needs. When a vessel contracts the resistance will rise and the flow abates. In the CwB model there is only included variable resistance in the vena cava and the systemic arteries. From the program we find that most of the non-linear vessel components are taken from the article [8]. This article put together models of vessels from other papers. We will only discuss the intention of each component, not the rational behind obtaining it.

The resistance of the vena cava, R_{vc} decreases when the volume, V_{vc} comes closer to the maximum volume, $V_{vc,max}$.

$$R_{vc} = K_R \left(\frac{V_{vc,max}}{V_{vc}} \right)^2 + R_0 \tag{2.24}$$

K_R is a scaling parameter and R_0 is an offset parameter. The flow through the variable resistances are still given by the pressure difference over the resistance, similar to equation (2.22).

The resistance of the systemic arteries, R_{sa} involves two terms. The first term is affected by the baroreflex, and the second is on the same form as the vena cava resistance.

$$R_{sa} = K_r e^{4 \times F_{vaso}} + K_r \left(\frac{V_{sa,max}}{V_{sa}} \right)^2 \tag{2.25}$$

K_r is a resistance scaling factor, and F_{vaso} is a normalized baroreflex signal of sympathetic efferent firing frequency. We will introduce the baroreflex in the next chapter.

2.2.3 Compliance vessels

Most of the vessels in this model consist of one compliance vessel connected to one resistance vessel, to ensure that each vessel is both compliant and has some resistance to flow. The pressure is dependent on the volume and compliance. The flow is dependent on the pressure difference from one vessel to the next in addition to the resistance.

The systemic capillaries are modelled as one such vessel, see Figure A.1. We have the pressure, P_{sc} given by the compliance relation

$$P_{sc} = \frac{V_{sc}}{C_{sc}} - \frac{P_{x2}}{e^{V_{sc}/V_{x8}} - 1} \tag{2.26}$$

The second term is chosen to give a realistic P-V relation, this term is only significant at a very low capillary volume, V_{sc}. The choice of the constants $P_{x2} = 2$ mmHg and $V_{x8} = 8$ ml relate to the size of the vessel. In other vessels different choices have been made. When $P_{sc} = 0$ this equation can be solved for the corresponding "dead" volume of the vessel.

Since the heart is beating, the system is not in a steady state, and we have pulsatile flow. This means every vessel has a changing volume. The volume change is the difference between inflow and outflow, as for the volume of the systemic veins, V_{sv},

$$\frac{dV_{sv}}{dt} = Q_{sc} - Q_{sv}. \tag{2.27}$$

The CwB model includes several junctions of the circulation. Conservation of mass states that all flow entering a junction must also leave the junction. In electric circuits this is called Kirchhoff's junction rule and relates to currents. As an example of branching of vessels, we consider the vena cava, which receives blood from two vessels, the systemic veins and the cerebral vessel. The volume is given by

$$\frac{dV_{vc}}{dt} = Q_{sv} + Q_{cbr} - Q_{vc}. \tag{2.28}$$

This branch can be seen in Figure A.1.

The first of the more complicated compliance relations is the case of the proximal aorta, right next to the heart. Here there are two compliance relations connected to one junction, see Figure A.1. There is however a resistance to flow, R_{taop} between the proximal aorta compliance, C_{aop} and the rest of the circulation, which eliminates the problem of having first order derivatives of two variables in one equation. The pressure difference between the junction, P_{aopc} and that set up by the proximal aorta compliance, C_{aop}, results in a flow through the resistance R_{taop} . Since the flow doesn't branch off, this flow must be a change in the volume of the proximal aorta, V_{aop}.

$$\frac{dV_{aop}}{dt} = \left[P_{aopc} - \left(\frac{V_{aop}}{C_{aop}} - \frac{P_{x2}}{e^{V_{aop}/V_{x8}} - 1}\right)\right] / R_{taop} \tag{2.29}$$

The proximal part of the aorta is connected to the proximal epicardial arteries in the coronary circualtion, which are also compliant (C_{corao}). The pressure relation at the junction is rather complex

$$\frac{dP_{aopc}}{dt} = \left[Q_{lv} - \frac{dV_{aop}}{dt} - Q_{aop} - Q_{corao}\right]\left[\frac{1}{C_{corao}} + \frac{P_{x2}}{V_{x1}}\frac{e^{V_{corao}/V_{x1}}}{(e^{V_{corao}/V_{x1}} - 1)^2}\right] \tag{2.30}$$

Decoding the previous equation, taking into account Kirchhoff's junction rule and Figure A.1, we find that

$$\frac{dP_{aopc}}{dt} = \frac{dV_{corao}}{dt}\left[\frac{1}{C_{corao}} + \frac{P_{x2}}{V_{x1}}\frac{e^{V_{corao}/V_{x1}}}{(e^{V_{corao}/V_{x1}} - 1)^2}\right] \tag{2.31}$$

This equation we observe is the time derivative of a compliance equation on the same form as equation (2.26).

The next compliance relation that needs to be addressed, is the distal pulmonary arteries. Here too, there is a resistance to flow between the compliance vessel and the circulation, see Figure A.1

$$P_{padc} = \frac{V_{pad}}{C_{tpad}} - \frac{P_{x2}}{e^{V_{pad}/V_{x8}} - 1} + (Q_{pap} - Q_{pad})R_{tpad} + P_{plc} \tag{2.32}$$

The volume, V_{pad} determines the pressure from the compliant walls, and then there is a pressure drop as a result of the flow through the resistance, R_{tpad}. This flow is inevitably the difference between the flow going into the vessel, Q_{pap} and the one going out, Q_{pad}. The equation also includes the pleural chamber pressure, P_{plc} as this is close to the lungs.

In the distal aorta one can recognise a similar construction, but owing to a branch off to the cerebral circulation, through a resistance R_{crb}, it has additional features.

$$P_{aodc} = \frac{\left[\left[\frac{V_{aod}}{C_{aod}} - \frac{P_{x2}}{e^{V_{aod}/V_{x8}} - 1} + (Q_{aop} - Q_{aod})R_{taod}\right] \cdot R_{crb} + P_{vcc}R_{taod}\right]}{(R_{crb} + R_{taod})} \tag{2.33}$$

We note that in the case of $R_{crb} \to \infty$, which would be the same as if there is no branch, the equation simplifies to the form of (2.32). The cerebral circulation rejoins the systemic circulation at the vena cava, where the pressure is P_{vcc}.

For the proximal pulmonary arteries, the chamber pressure, P_{papc} is determined from different formulas depending on whether the pulmonary valve is open or closed. When the valve is closed, we have an equation for the pressure on the form of (2.32), only there is no flow into the compliance vessel

$$P_{papc2} = \frac{V_{pap}}{C_{tpap}} - \frac{P_{x2}}{e^{V_{pap}/V_{x8}} - 1} - Q_{pap}R_{tpap} + P_{plc}. \tag{2.34}$$

When the pulmonary valve is open, the flow out of the ventricle, Q_{rv} is modeled on the form of a branch, like equation (2.33). We have

$$P_{papc1} = \frac{\left[\left[\frac{V_{pap}}{C_{tpap}} - \frac{P_{x2}}{e^{V_{pap}/V_{x8}} - 1} - Q_{pap}R_{tpap} + P_{plc}\right] \cdot R_{puv} + P_{rvc}R_{tpap}\right]}{(R_{puv} + R_{tpap})} \tag{2.35}$$

The chamber pressure in the right ventricle determines if the valve is open or closed.

$$P_{papc} = \begin{cases} P_{papc1} & \text{if } P_{rvc} > P_{papc1} \\ P_{papc2} & \text{otherwise} \end{cases} \tag{2.36}$$

The rest of the vessels have non-linear pressure-volume relations, found in the article [8]. We still only discuss the intention of each component. The components were developed in other articles.

For the systemic veins, the vessel walls will stiffen for increasing volume, V_{sv}, which increases the transmural pressure P_{sv}:

$$P_{sv} = -K_{sv} \log_{10} \left(\frac{V_{max,sv}}{V_{sv}} - 0.99 \right) \tag{2.37}$$

K_{sv} is a pressure scaling factor, and $V_{max,sv}$ is the maximum volume of systemic veins.

For certain pressure conditions the vena cava can collapse. The pleural membrane envelops each lung, and inside the lungs there is subatmospheric pressure, P_{plc}. If the pleural pressure is greater than the luminal (cavity) pressure of the vena cava, caval volume will decrease and resistance increase.

In the article the vena cava transmural pressure, P_{vc} is on the following form, V_0 is the unstressed volume.

$$P_{vc} = \begin{cases} D_1 + K_1(V_{vc} - V_0) & \text{if } V_{vc} \geq V_0 \\ \\ D_2 + K_2 \cdot e^{(V_{vc}/V_{vc,min})} & \text{if } V_{vc} < V_0 \end{cases} \tag{2.38}$$

$V_{vc,min}$ is the minimum volume. The values of D_2,D_2,K_1 and K_2 are based on curve fitting. We note that for $V_{vc} \geq V_0$ the vena cava is described by a linear compliance vessel. For the vena cava in the program, the D_1 parameter is chosen to be the pressure at $V_{vc} = V_0$, which will make the transition smooth.

$$P_{vc} = \begin{cases} D_2 + K_2 \cdot e^{(V_0/V_{vc,min})} + K_1(V_{vc} - V_0) & \text{if } V_{vc} \geq V_0 \\ \\ D_2 + K_2 \cdot e^{(V_{vc}/V_{vc,min})} & \text{if } V_{vc} < V_0 \end{cases} \tag{2.39}$$

The baroreflex can influence the systemic arteries to contract or relax. When the vessel contracts the walls also stiffen. The following P-V relations are developed from active and passive length-tension relationship of the muscle. The pressure in the systemic arteries activated by sympathetic stimuli, P^a_{sa}, is determined by the volume, V_{sa}

$$P^a_{sa} = K_c \log_{10} \left(\frac{V_{sa} - V_{sa,0}}{D_0} + 1 \right), \tag{2.40}$$

where $V_{sa,0}$ is the minimal volume, K_c is a scaling parameter, and D_0 is a volume parameter. The passive pressure in the systemic arteries, P^p_{sa}:

$$P^p_{sa} = K_{p1} \cdot e^{\tau_p (V_{sa} - V_{sa,0})} + K_{p2}(V_{sa} - V_{sa,0})^2 \tag{2.41}$$

where K_{p1}, K_{p2} and τ_p are scaling parameters. The baroreflex determine which of the two is the most dominant. The baroreflex will be explained in Section 3.2.

2.2.4 Inertance vessels

Inertance, L is an opposition to change of flow, Q. The pressure difference ΔP over an inerthance is

$$\Delta P = L\frac{dQ}{dt} \qquad \Rightarrow \frac{dQ}{dt} = \Delta P/L. \tag{2.42}$$

Both in the pulmonary arteries, and the aorta there is included two inertance vessels, in both cases named proximal and distal.

We will consider the inertance component, L_{pad} of the distal pulmonary artery, see Figure A.1. Since there is no expression for the pressure immediately before the inertance component, the pressure, P_{padc} in the point before the resistance, R_{pad} in series with the inertance is used as a starting point, and the pressure drop caused by this resistance is subtracted, to find the remaining pressure drop over the inertance.

$$\frac{dQ_{pad}}{dt} = \left[(P_{padc} - Q_{pad}R_{pad}) - P_{pac}\right]/L_{pad} \tag{2.43}$$

The pressure in the point after the inertance component is the pressure of the pulmonary arterioles, P_{pac}. A similar equation to that of the distal pulmonary artery is employed in all four cases.

In the smaller vessels the resistance causes the flow to decrease, this makes inertance less important [9]. One reason not to include inertance properties in the larger veins is that the flow is almost steady, a small variation in flow makes a small pressure difference over the inertance component. It can then be neglected.

The electrical analogue of inertance is inductance. In an electrical circuit, an inductance is a coil, [15, p. 1151].

2.2.5 The Heart

The CwB model includes a four-chamber elastic heart. The pressure in each chamber is determined by the phase of the heart cycle and the blood volume inside the heart.

We show the equations of the right heart chambers, and the left heart has a similar form. The chamber pressure of the right atrium, P_{rac}

$$P_{rac} = (V_{ra} - V_{ra,r})E_{ra} - \frac{P_{x2}}{e^{V_{ra}/V_{x8}} - 1} + P_{pcdc} \tag{2.44}$$

The right ventricular pressure, P_{rvc}

$$P_{rvc} = a_2(F_{con})(V_{rv} - V_{rv,r})E_{rv} - \frac{P_{x2}}{e^{V_{rv}/V_{x8}} - 1} + P_{pcdc} \tag{2.45}$$

E_{ra} and E_{rv} are the elastances of the right atrium and ventricle respectively. $V_{ra,r}$ and $V_{rv,r}$ are the unstressed volume of each chamber. We note that the

unstressed volume has a similar purpose as the "dead" volume in equation (2.3). The second term is a scaling for small volumes like in equation (2.26).

Elastance is the inverse of compliance, so a high compliance corresponds to a low elastance. The elastance is high during systole and low during diastole.

Atrial systole is triggered before ventricular systole. The electric activation is in each case a periodic function based on the heart rate. The activation function in the CwB model is from article [5], see the program in the appendix. In the article it was used only on elastance of the ventricles, E_{rv} and E_{lv}. In the CwB model it was also used for elastance of the atria, and to scale the unstressed volumes of the ventricles and atria. The unstressed volumes then becomes lower during systole than during diastole, which also raises the pressure during systole.

The sympathetic barosignal $a_2(F_{con})$ that influence elastance and therefore the force of the contraction, will be explained in Section 3.2.

The pericardium is the membrane around the heart. The pericardium chamber pressure, P_{pcdc} is dependant of the volume inside the pericardium, V_{pcd}. V_{pcd} is modeled to contain the pulsatile blood volume of the four heart chambers, the blood in the coronary circulation and the constant volumes of the myocardium, (heart muscle) and pericardial fluid. The P-V-relation for the pericardium in the CwB program is

$$P_{pcdc} = K_{pcd} e^{((V_{pcd} - V_{pcd,0})/\phi_{pcd})} - \frac{P_{x2}}{e^{V_{pcd}/V_{x75}} - 1} + P_{plc}, \tag{2.46}$$

the first term of the above formula has been used for the pressure in the pericardium in article, [2]. K_{pcd} is a pressure scaling factor, ϕ_{pcd} and $V_{pcd,0}$ are fitted parameters. The pericardial pressure is negative in relation to ground because of the influence of pleural chamber pressure of the lungs, P_{plc}. We checked this for a run in JSim, and plot of the P-V-relation. The pericardial pressure is the only part of the model that includes interaction between the ventricles and atria.

2.3 Toska circulation model

The Toska circulation model captures variations from beat to beat, but neglects the pulsatile pattern of the flow. The main reason for choosing a beat-to-beat model was that the measurements were sampled on a beat-to-beat basis [4].

The heart is modeled as a RR interval and a stroke volume, SV. Both of these are determined by the baroreflex signals, so the formulas for them will be introduced in the baroreflex chapter. For now, they can be considered constant. The RR interval is the inverse of the heart rate,

$$RR = 1/HR. \tag{2.47}$$

It is called a RR interval after the naming convention for the different parts of an ECG signal. The ECG signal is a measurement of the electrical activation of the heart. The highest spike in signal is the R wave, which results from the activation of the ventricles. The RR interval therefore begins and ends at the start of the ventricular systole.

The Toska model divides the circulation into the aorta and the peripheral vessels, see Figure 2.3. The peripheral vessels are divided into two parallel vessels representing exercising and non-exercising tissue. Total peripheral conductance, G_p is the combined effective conductance of the two parallel vessels.

We want to calculate total peripheral conductance from measurements. Since this is a beat-to-beat model, the flow into the vessels can be represented by the cardiac output, CO that is defined in equation (2.13). If we assume zero venous pressure, then equation (2.1) implies that an estimate of the total peripheral conductance is

$$G_p = SV/(MAP \cdot RR) = CO/MAP, \tag{2.48}$$

where MAP is the mean arterial pressure.

The recording of data started with the subject at rest. After a 10s countdown the subject performed moderate dynamic exercise while supine, that is laying down, see [13] for detailed information on the setup of the experiment. The muscles that perform work are the quadriceps muscles in the thigh. The femoral arteries is the main artery of each leg.

2.3.1 Windkessel model of the aorta

A linear elastic reservoir with a certain compliance is a one-element Windkessel model [4], which we have previously called a linear compliance vessel. The pressure in the aorta, P_a is then given by

$$P_a = V_a/C, \tag{2.49}$$

where C is the compliance of the aorta.

The stroke volume is added to the compliant aorta in one go at the end of the systole. Even though the model only samples the pressure at the end of the diastole, P_d and mean arterial pressure, MAP beat-to-beat, these values are found considering flow out of a compliant vessel.

Difference between inflow and outflow in a compliance vessel gives a changing volume, here the volume of the aorta, V_a. The rate of change in the pressure in the aorta is

$$\frac{dP_a}{dt} = \frac{1}{C}\frac{dV_a}{dt} = (Q_h - Q_p)/C \tag{2.50}$$

where Q_h is flow out of the heart and, Q_p is the flow into the peripheral vessels, given by

$$Q_p = P_a G_p, \tag{2.51}$$

if we assume zero venous pressure, this is the relation for a resistance vessel. Peripheral conductance, G_p is the inverse of peripheral resistance.

Solving equation (2.50) with (2.51) and remembering that the inflow, Q_h is added to the volume reservoir, V_0 in one go called the stroke volume, SV. The volume of the aorta becomes

$$V_a = V_0 e^{-tG_p/C} \tag{2.52}$$

The outflow Q_p is the result of an exponential pressure-dependent volume decay.

2.3.2 Parallel peripheral arteries

The general formula for peripheral conductance, G is affected by a sympathetic barosignal, B_{sp}. This signal will be introduced in the baroreflex chapter. For now we set

$$G = G(G_0, B_{sp}) \tag{2.53}$$

where G_0 is the mean of total peripheral conductance at rest, calculated from measurements. Exercising and resting tissues are divided into two vessels in parallel. From the experiment set up we have that the exercising part is the legs, and the resting part is the rest of the systemic circulation. The effective conductance of the peripheral vessels, G_p is

$$G_p = G_r + G_e \tag{2.54}$$

The equation (2.53) determines both G_r and G_e before exercise, so that the conductance of resting tissue, G_r is given by

$$G_r = (1 - Ex_{Cond})G \tag{2.55}$$

where Ex_{Cond} is the fraction of the total conductance that is through exercising tissue.

During exercise, local metabolites will override sympathetic stimuli, and result in vasodilation of the muscles that need more oxygen. We will come back to this in section 3.1. In the article [4] this is solved in the following way

$$G_e = \begin{cases} Ex_{Cond}G & \text{before and,} \\ \\ Q_{mf}/MAP & \text{after onset of exercise.} \end{cases} \tag{2.56}$$

This model would be correct if Q_{mf} was the entire flow though the exercising muscles, but in the program Q_{mf} is set to be the increase in flow compared to baseline.

Increase in flow in exercising muscles, Q_{mf} is modeled as an exponential function. There is a small delay, d_{mf} between start of exercise and increase

in flow.

$$Q_{mf} = \begin{cases} Q_{mf,0} & t < t_{ex} + d_{mf}, \\ \\ Q_{mf,0} + Q_{mf,max}(1 - e^{(t_{ex}+d_{mf}-t)/T_{c,mf}}) & t \geq t_{ex} + d_{mf}. \end{cases} \tag{2.57}$$

This relation for Q_{mf} is displayed in article [4] figure 2B and in the program in the appendix. Since Q_{mf} models the increase in flow then $Q_{mf,0} = 0$. Time of onset of exercise is t_{ex}.

When we define Q_{mf} as the increase in flow during exercise, we need to include the baseline conductance, $G_{e,0}$ when we model conductance through exercising muscles. This is done in the program and the equation is given below

$$G_e = \begin{cases} Ex_{Cond}G & \text{before and,} \\ \\ G_{e,0} + Q_{mf}/MAP_0 & \text{after onset of exercise.} \end{cases} \tag{2.58}$$

where $G_{e,0} = Ex_{Cond}G_0$. Compare this equation with equation (2.56) that is taken from the article [4]. If we don't include $G_{e,0}$, then equation (2.57) is going to make $G_e = 0$ right after the start of exercise.

There is also another difference between the article and the program. In equation 2.58, we see that the conductance in exercising muscles is given by baseline mean arterial pressure, MAP_0, while in the article this relation instead includes the mean arterial pressure at the given heart beat, MAP. Conductance is defined as the flow divided by the pressure difference, since it is the inverse of the resistance, see equation (2.1). We assume that the systemic venous pressure is negligible. This means that the program should be changed to fit the original intension of the article.

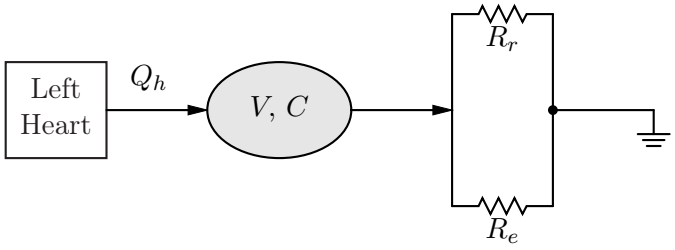


Figure 2.3: Circulation in the Toska model, representing the heart, the aorta/large arteries and the peripheral vessels.

Chapter 3

Baroreflex models

3.1 Physiology of the Baroreflex

The function of the baroreflexes is to stabilize the blood pressure. The baroreflexes are a part of the central nervous system (CNS).

Nerve cells that act as receptors are called afferent neurons. The CNS then interpret the information, and an appropriate response is sent through efferent neurons to target cells. Neurons have a cell body, axons and dendrites. At the end of the axon, where the neuron reach the target cell it ends in a synapse.

It is common to divide the CNS in two blocks. The somatic division includes that which we are conscious of. Examples include afferent receptors of temperature, and efferent control of the skeletal muscles. The autonomic division makes automatic, unconscious responses to changes in the body's internal environment.

The autonomic division is subdivided into the sympathetic and parasympathetic pathways. Autonomic pathways control smooth and cardiac muscles and some glands.

The sympathetic branch controls fright, fight and flight responses, but also serves other functions like control of blood flow to tissues. The parasympathetic pathways dominate when the body is at rest.

Most internal organs are under antagonistic control by the autonomic pathways. For instance sympathetic inervation increases heart rate while parasympathetic decreases it.

The smooth muscle in most blood vessels are innervated only by the sympathetic branch. Most vessels contain one type of adrenergic receptor that causes smooth muscle contraction (vasoconstriction), but some contain a second type of receptor that causes smooth muscle to relax (vasodilation). Both receptors are activated by the sympathetic neurotransmitter.

Most sympathetic pathways secrete the neurotransmitter norepinephrine also called noradrenalin, at the synapse. Parasympathetic pathways secrete

the neurotransmitter acetylcholine. The target cells responds differently to each transmitter.

The baroreflex belongs to the autonomic nervous system, and is the primary reflex for homeostatic control of blood pressure. Baroreceptors are stretch sensitive and respond to changes in pressure. There are baroreceptors on the vessel wall of the aorta and the carotid artery. When the blood pressure drops the baroreceptor firing rate decreases, and the reverse is true for increasing pressure. The response of the baroreflex is rapid; changes in cardiac output and peripheral conductance occur within two heart beats of the stimuli.

There are four important baroreflex pathways, three of which we have already mentioned. Increased sympathetic signal causes vasoconstriction. For the heart, increased sympathetic signal will increase heart rate at the sinoatrial node where the activation starts, and shorten conduction time through the AV node. Increased sympathetic signal will also enhance the force of myocardial, that is cardiac muscle, contraction. Increased parasympathetic signal will slow the heart rate. Parasympathetic signal have very little influence on contraction, so this is not included in the models.

Baroreflexes are responsible for changes that happen during exercise. When muscles work they need more oxygen, and the cardiac output must rise. Initially the heart rate increases due to decreased parasympathetic signal, then from increased sympathetic signal. The increase in heart rate gives the heart less time to relax and protect it from overfilling. Sympathetic stimuli also increases contractility and thus stroke volume.

During exercise the blood is redistributed to the exercising muscles through vasodilation in skeletal muscle arterioles and vasoconstriction in other tissues. The sympathetic stimuli causes vasoconstriction. Vasodilation can be caused by local decrease in O_2 concentration, and increase in CO_2 among other factors. Metabolites is a general term for products of the metabolism.

The result of all these changes is that the blood pressure rises slightly during exercise. One theory of why this is allowed by the baroreflex is that the baroreflex setpoint is reset to a higher pressure.

The content of this section is based on chapters 5, 6, 8, 10, 11, 15 and 25 in [10].

3.2 CwB baroreflex

The baroreceptor signal, N_{br} depends on the pressure in the aorta, and the rate of change in this pressure. The equation of the barosignal in the CwB model is taken from article [11]. It is developed from measurements, on the form of a transfer function, a Laplace transform,

$$\frac{N_{br}(s)}{P_{aod}(s)} = K\frac{(1+a_1 s)}{(1+a_2 s)(1+as)}. \tag{3.1}$$

The corresponding differential equation is given in the article. It can be found from reverse Laplace transform. The program uses the differential equation form of the baroreceptor signal, N_{br}

$$aa_2\frac{d^2N_{br}}{dt^2} + (a + a_2)\frac{dN_{br}}{dt} + N_{br} = KP_{aod} + a_1K\frac{dP_{aod}}{dt} \tag{3.2}$$

where the time constants a_1 and a_2 are average values from measurements, found from plotting the gain, defined as amplitude of the baroreceptor response divided by the amplitude of the pressure wave. The time constant a is included, because in the article it is expected that pressure waves of high frequency will not result in similar rapid strain of the vessel walls. The baroreceptor signal will go to zero in this case, a time constant $a < a_2$ should have this effect. In the program $a = 0.001s$, and this is chosen without reference to measurements. The parameter K accounts for differences in units between impulse frequency and pressure. The equation is based on measurements of the carotid baroreceptors in dogs, in our case it is used for the aortic baroreceptors in humans.

In the article [8] they use the baroreceptor above, as well as the signal for each pathway at the central nervous system, CNS and the efferent pathway. The pathway discharge frequency at CNS, N_x is described by a transfer function in figure 2 in the article. This is a model for the medullary cardiovascular control center in the brain, and x is exchanged for the four different signal pathways. The signal pathways are sympathetic and parasympathetic control of heart rate ,N_{hrs} and N_{hrv} respectively, and sympathetic control of contractility, N_{con} and peripheral conductance, N_{vaso}.

$$\frac{N_x(s)}{N_{br}(s)} = \frac{K_xe^{-L_xs}}{(T_xs + 1)}. \tag{3.3}$$

Each pathway, N_x react to stimuli, N_{br} with a delay, L_x, a time parameter, T_x, and a gain K_x. The corresponding differential equation from reverse Laplace transform is given in the program.

$$\begin{cases} T_x\frac{dN_x}{dt} + N_x(t) = K_xN_{br}(t - L_x) & t > L_x, \\ \\ \frac{dN_x}{dt} = 0 & t \leq L_x. \end{cases} \tag{3.4}$$

F_x is the normalized outgoing efferent signal to the affected organs

$$F_x = a_x + \frac{b_x}{e^{\tau_x(N_x(t)-N_{x,0})} + 1} \tag{3.5}$$

The parameters τ_x and $N_{x,0}$ are fitted to representative data. The equation gives a sigmoidal relationship between central neuron activity, N_x and discharge frequency of efferent neurons, F_x. Since increase in baroreceptor signal causes increased parasympathetic signal, the corresponding τ_{hrv}

is negative. Baroreceptor signal and sympathetic signals change in opposite directions, and τ_x is positive in these cases.

Heart rate is regulated by both a sympathetic, F_{hrs} and parasympathetic F_{hrv} efferent signal. The following equation characterizes a three dimensional heart rate response surface to sympathetic and parasympathetic stimuli.

$$HR = h_1 + h_2 F_{hrs} - h_3 F_{hrs}^2 - h_4 F_{hrv} + h_5 F_{hrv}^2 - h_6 F_{hrv} F_{hrs} \tag{3.6}$$

where $h_1 - h_6$ are constants. In the CwB program, the heart rate is changed once every heart beat according to the formula above. Otherwise the heart beat gets interrupted and become erratic, as noted in the program, [1]. This also applies to the contraction baroreflex.

The force of contraction of the heart is controlled by the sympathetic efferent signal, F_{con}. Increase in the sympathetic signal increases the maximum elastance through this linear equation

$$a(F_{con}) = a_{min} + K_a F_{con} \tag{3.7}$$

with minimal value a_{min} and scaling K_a.

In the program the function $a_2(F_{con})$ is sampled from $a(F_{con})$ once every heart beat, and kept constant during the beat. The following equation is for the pressure in the left ventricle, P_{lvc} and a similar expression exists for the right ventricle, equation (2.45).

$$P_{lvc} = a_2(F_{con}) E_{lv}(V_{lv} - V_{lv,r}) - \frac{P_{x2}}{e^{V_{lv}/V_{x8}} - 1} + P_{pcdc} \tag{3.8}$$

the elastance, E_{lv} is defined as the inverse of the compliance. The CwB model employs a different activation function of the ventricle than the article [8]. In the article it is included a formula so that increased signal F_{con} affects the activation function by shortening ventricular systole, this is not included in the CwB model.

A high value for sympathetic signal, F_{vaso} increases the resistance in the systemic arteries. This is called vasoconstriction. When smooth muscle in the arteries constrict, the vessel diameter decreases and the muscle wall stiffen. Therefore, the sympathetic signal affects both resistance and and compliance. The equation for resistance we recognize from Section 2.2.2.

$$R_{sa} = R_{sa,0} + K_r e^{4F_{vaso}} + K_r \left(\frac{V_{sa,max}}{V_{sa}} \right)^2 \tag{3.9}$$

Figure 3.1 shows how resistance depends on the volume for two values of F_{vaso}.

The pressure is determined by an active, $P_{sa,a}$ and a passive, $P_{sa,p}$ pressure-volume relation, see equations (2.40) and (2.41) respectively. A high signal F_{vaso} favours the active, slightly higher pressure

$$P_{sa} = F_{vaso} P_{sa,a} + (1 - F_{vaso}) P_{sa,p} \tag{3.10}$$

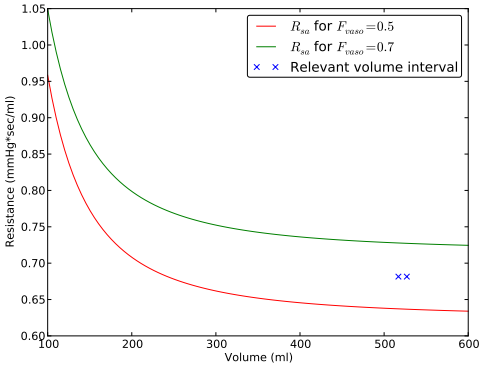


Figure 3.1: Resistance of systemic arteries

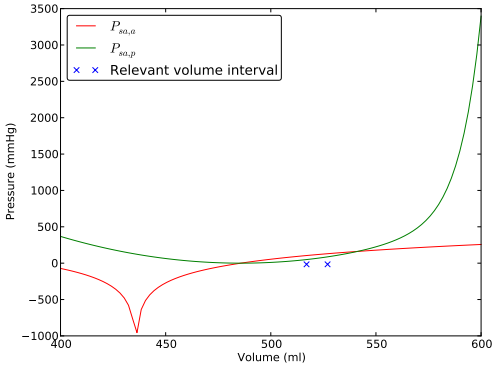


Figure 3.2: P-V relations of systemic arteries

When there is no sympathetic signal, the pressure in the arteries are described by $P_{sa,p}$, article [8]. Figure 3.2 shows the two pressure-volume relations for active and passive respectively. We note that if the volume falls outside a specific range, the passive pressure formula becomes the largest. In this case the baroreflex will loose its purpose, and the results will be unpredictable. In the given parameter setting the volume does not fall outside the permitted range.

3.3 Toska baroreflex

3.3.1 Modeling of baroreflex signal: article contra code

The baroreflex is modeled as a continuous function [4], even though the circulation variables are sampled beat-to-beat. The baroreceptor registers deviation in mean arterial pressure from setpoint, but not the rate of change in pressure, which would be difficult in a beat-to-beat model. This integral for baroreflex signal is given in the two articles that present this model.

$$B(t) = \frac{1}{T_c} \int_{-\infty}^{t-d} [P(t') - P_s(t')] e^{[(t'+d)/T_c]} dt' \tag{3.11}$$

The signal that the program actually produces is quite different, and more realistic. This is described in detail in Section 4.2.4.

Both versions of the barosignal feature a delay, d and a time constant, T_c to model how fast the baroreflexes react to changes in arterial blood pressure. The time constant T_c and delay d is different for each signal pathway. The Toska model includes three efferent pathways. Those are the parasympathetic influence on heart rate and the sympathetic influence on contraction and peripheral conductance. The signal represents the firing frequency of

the respective nerves, but it is expressed in mmHg which are the units for pressure.

The baroreflex setpoint, P_s is found from program and Figure 2A in article [4].

$$P_s = \begin{cases} P_{s,0} & t < t_{count}, \\ \\ P_{s,0} + P_{s,step} F(1 - e^{(t_{count}-t)/T_{c,set}}) & t \geq t_{count}. \end{cases} \tag{3.12}$$

$P_{s,step}$ is the maximum increase in baroreflex setpoint, and $T_{c,set}$ is the time constant that determines how fast the setpoint increases. The model proposes that the setpoint increases when the body is getting ready for exercise, here represented by t_{count} which is the time when they start a countdown in the exercise experiments. There is a parameter in the program called the before fraction, left over from the development of the code, here called F. This parameter is always $F = 1$, and changing it would give the wrong increase in setpoint, $P_{s,step}$. For each subject the initial baroreflex setpoint is the mean arterial pressure at rest, $P_{s,0} = MAP_0$ found from measurements.

3.3.2 Baroreflex for the Heart

The duration of one heart beat is influenced by the parasympathetic signal, B_{ph}. The sensitivity to the signal is K_{ph}, this parameter is sometimes called the gain of the signal.

$$RR = RR_0 + K_{ph} B_{ph} \tag{3.13}$$

RR_0 is the average RR-interval at rest, found from individual measurements. An earlier version of the model also included sympathetic influence on RR-interval. This reflex did not improve the fit between the simulated and the recorded time courses [4].

The sympathetic signal to the heart muscle changes the force of contraction, resulting in a change in stroke volume, SV. The stroke volume also depend on afterload and preload, modeling the effect of aortic pressure and filling of the ventricle respectively.

$$SV = SV_0 + K_a P_{d(n-1)} + Q_m(RR_{(n-1)} - RR_0) + K_{sh} B_{sh} \tag{3.14}$$

SV_0 is the average stroke volume at rest, from measurements. The three other terms model the factors that cause deviation from this baseline. The sympathetic signal, B_{sh} result in a slight decrease in stroke volume when mean arterial pressure is below the setpoint, the gain is K_{sh}. This is the opposite of the CwB model where a decrease in aorta pressure will result in increasing sympathetic barosignal and increased contraction.

Preload is the degree of heart muscle stretch, and that depends on the filling, [10]. Q_m is the mitral flow into the ventricle, and is assumed to be constant in the late diastole, [4]. The difference in RR-interval determines

the increase or decrease in filling volume compared to baseline. $RR_{(n-1)}$ is the value at the previous beat. The Frank-Starling law states that the end-systolic volume of the ventricle is constant, so the blood returned to the ventricle is ejected.

Afterload is the ventricular force that must be used to overcome the resistance created by the blood filling the aorta, [10]. $P_{d(n-1)}$ is the end-diastolic pressure in the aorta at the previous beat. The afterload sensitivity, K_a is a negative constant. Increased pressure will decrease stroke volume.

In the program the equation for the stroke volume is slightly different

$$SV = SV_0 + K_a(P_{d(n-1)} - P_{d(0)}) + Q_m(RR_{(n-1)} - RR_0) + K_{sh}B_{sh} \quad (3.15)$$

The difference lies in the term that represents the effect of afterload on stroke volume. The version in the program is the most plausible since it adds the deviation from baseline in end-diastolic pressure, P_d. It also resembles the filling term, where it is the change in RR-interval that determines if this term increases or decreases the stroke volume. The SV_0 term includes both the effect of afterload and filling at baseline.

3.3.3 Parallel peripheral arteries

In general the peripheral conductance, G is altered by sympathetic vasoconstriction. If the mean arterial pressure is below the setpoint then the sympathetic signal B_{sp} is negative. The conductance G decreases and we have vasoconstriction.

$$G = G_0(1 + K_{sp}B_{sp}) \quad (3.16)$$

G_0 is the mean of total peripheral conductance at rest, calculated from measurements. K_{sp} is the gain or sensitivity of the vessel to a given barosignal. This equation will substitute equation (2.53) in the model of the parallel peripheral arteries.

Chapter 4

Conversion of code

The CwB model was written in JSim's MML (Mathematical Modeling Language), and the Toska model was programmed in Matlab. Both were converted into Python in the beginning of the project to facilitate integration. In the case of the CwB model, the equations needed to some extent to be rearranged since MML is not a sequential language. Due to this, the equations were ordered according to physiology. The conversion of the Toska model was more straightforward after one had untangled the core model from the MATLAB GUI that was included to simplify model parameter adjustments for non-programmers.

4.1 CwB program

We start with a short introduction to programming in JSim and MML. In MML one can write differential equations almost like one would write them on paper. Every variable and parameter is declared with units like ml, sec and so on, and JSim has automatic unit conversion. In MML, the time span of the simulation is declared as a *realDomain*, which is the construct for the independent variables of ODEs. It is a fixed, evenly spaced grid.

```
realDomain t sec; t.min=0; t.max=20.0; t.delta=0.01;
```

We are using the baroreflex for contractility control as an example. We give a reminder of the equations (3.4) and (3.5) below for the contractility case.

$$\begin{cases} T_{con}\frac{dN_{con}}{dt} + N_{con}(t) = K_{con}N_{br}(t - L_{con}) & t > L_{con}, \\ \frac{dN_{con}}{dt} = 0 & t \leq L_{con}. \end{cases} \tag{4.1}$$

$$F_{con} = a_{con} + \frac{b_{con}}{e^{\tau_{con}(N_{con}(t) - N_{con,0})} + 1} \tag{4.2}$$

We observe that in JSim parameters are declared as *real*.

```
real K_con=1 dimensionless;//CNS gain for contractility control
real T_con=10 sec;//CNS time parameter, contractility control
```

State variables are declared as an array with the same dimensions as the independent variable *realDomain*, here time.

```
real N_con(t) sec^-1;// Sympathetic discharge rate at CNS for
                     // contractility.
```

Other variables that change continuously through the calculation, are also declared as an array of time.

```
real F_con(t) dimensionless;  // Normalized discharge rate for
                              // contractility control
real afs_con(t) dimensionless;// Dynamic heart contractility
                              // scaling function
```

Variables that are updated discretely are given a different label; *realState*, but are still declared as an array of time, with the same spacing.

```
realState afs_con2(t) dimensionless;
// Discrete heart contractility scaling function
```

These declarations result in both discrete and continuous variables being saved when applying the ODE-solver, and they can be plotted in JSim. In Python the only output of the scipy.odeint function are the state variables, which is the standard for ODE solvers.

Initializing these arrays of time, is done using an MML operation called *when*. Which is, at least from the users point of view, the same as an if-test.

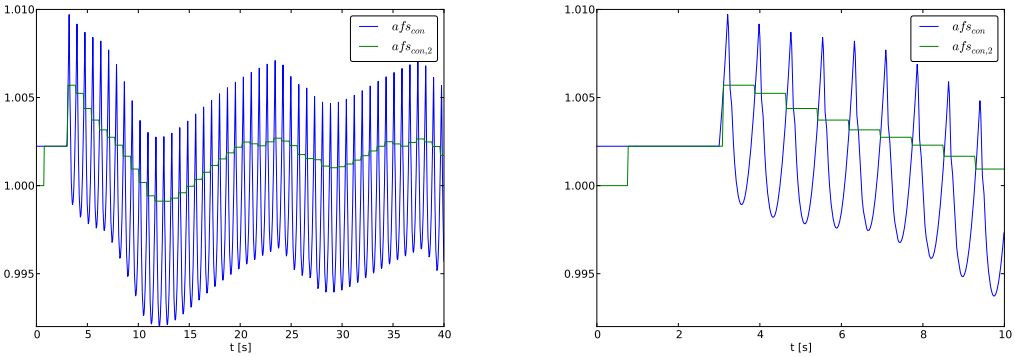
```
when(t=t.min) {
 N_con = 94.97;
 afs_con2 = 1;}
```

The discrete variables are changed at the end of every heart beat, with an operation called *event*, which also resembles an if-test. This kind of event handling is a common feature of many ODE-solvers, but it is not supported in scipy.odint or other available solvers in python.

```
event (t+t.delta>=(tshift+(1/HR))){
 HR = HRcont;
 tshift = t;
 afs_con2 = afs_con;}
```

Here $a(F_{con})$ or *afs_ con* is the continuous variable calculated from equation (3.7). Before $a(F_{con})$ is used in equation (3.8) as a baroreflex, it is sampled at the end of every heart beat in *afs_ con2*, see Figure 4.1. Notice that it is not necessary to write something like $HR(t) = HRcont(t)$, to access the current element of an array.

A simple first order ODE, represented by equation (2.27), is written like



(a) Simulation time 40s

(b) Simulation time 10s

Figure 4.1: The variables $a(F_{con})$ and $a_2(F_{con})$ calculated by CwB model in JSim, the discreate $a_2(F_{con})$ scale the compliance of the ventricle.

```
Vsv:t = Qsc - Qsv;
```

The differential operator is given by a colon. This is the entire code needed for such a differential equation.

The contraction baroreflex is a delay first order ODE.

```
//Discharge frequency controlling contractility of heart
N_con:t=if (t.min+t>L_con) (-N_con +(K_con*Nbr(t-L_con)))/T_con
        else 0;
```

We notice that normal if-tests in JSim only work on one variable, here the dN_{con}/dt is defined from an if-test. To access what is stored in the N_{br} state variable from an earlier time, just subtract the delay from the current time.

We extract data from a JSim run so that we can compare plots from JSim and python more conveniently.

JSim and MML are highly integrated. In the MML source code all parameters and initial conditions are declared and given a value, but these values are always replaced by the values entered into the JSim GUI. For a while we used values from the source code in the new python program, since this was the starting point of the conversion. These values were not optimized, which meant it looked like there was something wrong with the translation process.

4.1.1 JSim has automatic unit conversion

Because of the automatic unit conversion, some equations had to be carefully translated since python does not include units, only plain numbers. The most prevalent units are millilitres (ml) for volumes, seconds (sec) for time and millimetres of mercury (mmHg) for pressure. We make certain litres (L)

and minutes (min) are accurately transferred to ml and sec. All parameters related to the heart rate were given in minutes, which had to be converted to seconds.

In addition to the equations that include the heart rate, the two inertance equations have mixed units. The equations that involve inertance are the proximal aorta flow and proximal pulmonary arterial flow. Here the flow is given in L/min, while the other entities have the prevalent units. The proximal aorta flow is shown as an example.

```
real Raop=0.0001  mmHg*sec/ml;  // Proximal aortic resistance
real Laop=3.5E-4  mmHg*sec^2/ml;// Proximal aorta inertance

real Qaop(t)  L/min;// Proximal aorta flow
real Paopc(t) mmHg; // Proximal aorta chamber pressure
real Paodc(t) mmHg; // Distal aorta chamber pressure

when(t=t.min){Qaop  =0.69034755;}

Qaop:t = (Paopc - Qaop*Raop - Paodc) / Laop;
```

In the python program we use the conversion 1L/min=1000ml/60sec in python and obtain the initial value:

```
Qaop_0   =  11.5057925
```

The automatic unit conversion is the reason that there is no conversion factor that we need to change. We plot Q_{aop} simulated by both python and JSim, see Figure 4.2, and we observe that the initial condition is correct.

4.1.2 Handling discrete variables in scipy.odeint

The scipy.odeint function only accepts as input a function containing the right hand side of the differential equations, initial conditions for state variables, time and a tuple for parameters. The parameters must be immutable, they cannot be changed. The continuous variables, like pressures, are completely determined by state variables and constant parameters, and are calculated in each time step, so there is no problem here. However, the discrete variables are dependent on values calculated at the previous heart beat, and these are in scipy.odeint either not saved, or in the case of state variables, not accessible until the integration is done.

The problem lies in the variables that are declared *realState* in JSim, and changed by *event*. The quick solution is to create a class where we can store the necessary variables during an odeint-run. We make an object of this class in main. It is not possible to send this object into the scipy.odeint function since an object is certainly not immutable. The reason this works regardless, is that any variable declared in main in python is a global variable. If a global

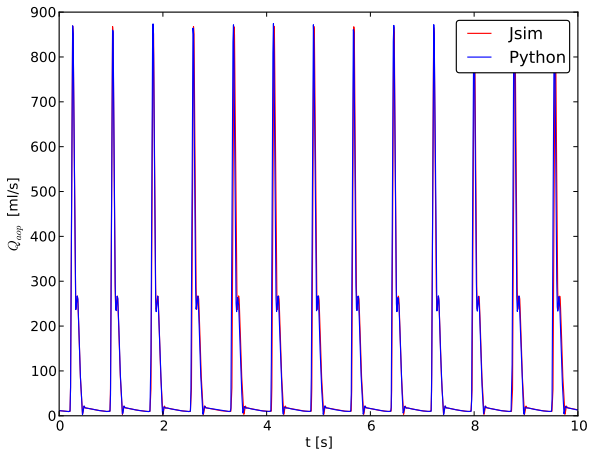


Figure 4.2: Flow through the proximal aorta, Q_{aop} in the CwB model, both in Jsim and python to show the result of the conversion.

variable defined in this way is going to be changed inside a function, it needs to either be inside a class or an array. This is not good programming, and should be avoided if at all possible. In this case it was the simplest way to circumvent the restrictions of scipy.odeint. The result is that all discrete *realState* variables in JSim, become class variables in our program. Without tampering like this we would have had to write a new ode-solver.

4.1.3 Choice of delay in the baroreflex differential equation

From equation (3.4) we see that when we want to calculate the pathway signal at CNS, N_x, we need to access the N_{br} variable from a fixed delay L_x back in time, depending on the reflex. This is called a delay differential equation.

The delay means we must save $N_{br}(t)$. The scipy.odeint function does not use a fixed time step while integrating, it is adaptive. The reason to include a time step, dt is to specify which values to save to the resulting state variable array. To save $N_{br}(t)$ for every tiny adaptive time step would be too time consuming. JSim, like python, only saves N_{br} with a sampling spacing dt, and this is what they can access when writing $N_{br}(t - L_x)$. The "MML Reference Manual" which can also be found from the Cwb page [1], says that the return value is determined by interpolation.

We do not include interpolation in our program. Instead we choose to hold the value of N_{br} constant for each time step, dt. This will be more effective than saving for every variable time step. We are saving fewer values, and it is much easier to backtrack when the time step is fixed. We still chose to use a list for saving N_{br}, even though an array is possible in this case.

The list will be as long as the number of saved time steps. Going back a delay L_x in time, is always the same number of steps from the end of the list. Python has short hand notation so $N_{br}[-1]$ is the last element in the list N_{br}. The form of the delay differential equation in python then becomes

```
if(t_min+t>L_con):
     dN_con =(-N_con+ K_con*hr.Nbr[-int(L_con/t_delta)]))/T_con
else:dN_con =0
```

As explained in the previous section, we use an object to save values. Even though N_{br} is a state variable, we can't access the array where odeint stores output.

```
if( t>= hr.t_shift_Nbr+t_delta):
        hr.Nbr.append(Nbr)
        hr.t_shift_Nbr+=t_delta
```

4.1.4 Sorting equations

In the CwB program the equations are separated in sections for each part of the circulation. For instance, all equations for the pulmonary circulation are grouped together. MML is not a sequential language, which means that the order of equations is irrelevant as long as all variables are defined in the end.

A typical example is the equations for the left atrium. Observe that the equations are given in almost the reverse order of what is necessary in python.

```
// Left Atrium, la, and its contraction:
Vla:t= (Qpv-Qla);
Qla  = if (Plac>Plvc) (Plac-Plvc)/Rla  // Eq. C
       else 0;
Pla  = (Vla-Vlar)*Ela-Px2*(1/(exp((Vla)/Vx8)-1)); // Eq. B,F
Plac = Pla + Ppcdc;
```

We go through the positions of the variables that are involved, but not defined in the above program excerpt. V_{lar} and E_{la} are given further up in the program, together with the other equations for the activation of the heart. P_{lvc} is defined in the paragraph directly below in the program, concerning the left ventricle. To find the equation for Q_{pv}, we search further down in the section with equations for the pulmonary system. P_{pcdc} is given in the section in the program with equations of the pericardium.

For the python program, we tried to keep as much as possible of the sections in which the equations are grouped. For instance we could move all the equations of the pericardium up and place them right under the

activation of the heart. This was necessary since many chamber pressures in the torso are influenced by the pericardial chamber pressure, P_{pcdc}.

While sorting the equations, we found a problem in the code. The suffix $_{sap}$ has been assigned two different meanings. It stands for both systemic arterioles and proximal systemic arteries. From physiology the proximal systemic arteries are between the aorta and the systemic arteries, while the systemic arterioles are between the systemic arteries and the systemic capillaries.

First we get this definition:

```
real Vsap(t)  ml;              // Systemic arterioles volume
real Psap(t)  mmHg;            // Systemic arterioles pressure
```

But then the definition changes:

```
real Rsap=0.025 mmHg*sec/ml;//Prox.systemic arteries resistance
real Csap=1.484 ml/mmHg;   //Prox. systemic arteries compliance
```

The reason it becomes proximal systemic arteries is because following the conventions of variable names in the rest of the program, the last p means proximal. As a result, these equations are placed before the systemic arteries in the flow equations. Since the compliance relation for P_{sap} is modeled like the smaller vessels, on the form of equation (2.26), it is more likely that this is indeed the systemic arterioles.

In the CwB python program, we tested what would happen if we changed the six affected equations. The initial conditions for V_{sa} and V_{sap} can not be interchanged because the given P-V relation for systemic arteries only apply for volumes in the interval [≈450-550ml], where active pressure is above passive pressure, see Figure 3.2. Another reason not to switch is that we want the volume of the systemic arteries to be the same. We want to interchange two vessels, but we want their respective volumes maintained.

Though the results were not unreasonable, it was clear that the parameters needed to be fitted again for this new case. For instance, all volumes in the pulmonary system increased. In all subsequent runs of the CwB-model we use the original order of equations, with $_{sap}$ meaning proximal systemic arteries.

4.1.5 Differences between the CwB-model in JSim and Python

The results of the CwB-model shows some differences between the JSim and Python programs. The main differences are in the baroreflex variables. From Figure 4.3 we observe the differences in the baroreceptor signal N_{br} and the contraction pathway signal at CNS, N_{con}.

In figure 4.3a we observe that the N_{con} signal from the Python run increase to a level while the JSim run also oscillate with large period. The other signal pathways N_{hrs}, N_{hrv} and N_{vaso} also have these oscillations in

the JSim run. The oscillations have a period of approximately 15s. In Figure 4.1 we observe that the oscillation in N_{con} cause a similar period to the discrete $a_2(F_{con})$ variable which is the one that actually influences the contraction of the ventricle.

The signal N_{con} is entirely dependant on N_{br} and some constant parameters. If we compare the Figures 4.3b and 4.3d we observe that the differences in N_{con} are more pronounced than differences in N_{br} leading to a suspicion that it is the implementation of the delay differential equation that causes the differences.

The flow in the proximal aorta, Q_{aop} is affected by the consequences of both the sympathetic contraction barosignal N_{con} and the parasympathetic and sympathetic heart rate barosignal N_{hrv} and N_{hrs}. We see from Figure 4.2 that the differences in Q_{aop} between the JSim and the Python run are very small. Another interesting difference is the MAP variable, or mean arterial pressure. This variable is discussed in Section 5.3 in the chapter about connecting the two models. In all variables there is an initial period before the baroreflexes land the system in a stable state. In Figure 4.4 we observe that after the initial change, the MAP variable in addition to varying with the heart rate, also oscillates with a larger period in time, when using the JSim program. In the python program this larger oscillation does not occur. This is similar to the oscillations in the signals N_{con} and the other pathways. One reason that this variable in particular is so affected can be that the arterial pressure P_{sa} is directly affected by N_{vaso}. Though when we look at P_{sa} in JSim we do not see a marked oscillation here.

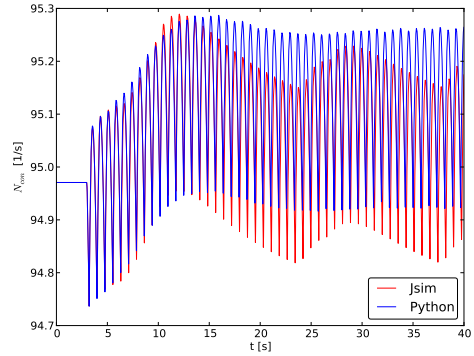
There are measured oscillations in atrial pressure and heart rate of around that of respiration, which are not well understood. The most studied is how heart rate varies with breathing. This is called respiratory sinus arrhythmia, [7, p. 496]. The effect of breathing on the circulation is not included in our model, the only related parameter is pressure in the lungs, P_{plc} which is modeled to be constant. During respiration the pressure in the lungs varies periodically.

In JSim the total volume is constant to machine precision, but in the python program we get an error of order of magnitude 0.01ml. The error does not get bigger with time. We checked a run of the python program that simulates 400s.

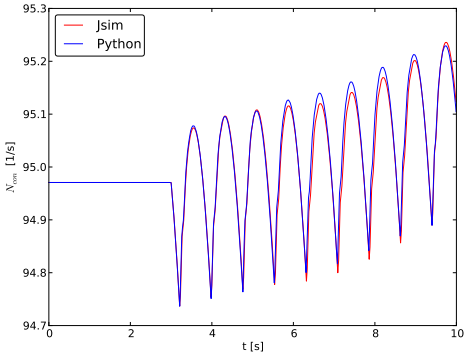
All other variables seem to be in accordance, to the same degree as Q_{aop} in Figure 4.2. Overall, the result of the conversion is satisfactory for our purpose.

The most likely cause of the discrepancies between the two versions of the model, is the delay in the baroreflex. As noted in Section 4.1.3, in JSim they find the delay from interpolation of the values at the two neighbouring time steps, while we chose an easier solution of using the same value of the $N_{br}(t - L_x)$ delay over an interval dt.

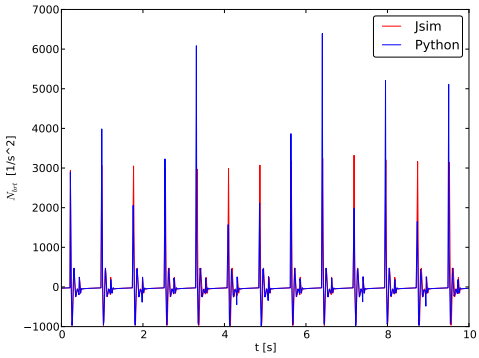
The event handling is also slightly different between the two programs,



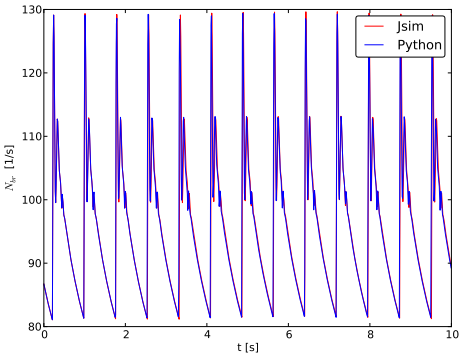
(a) N_{con} simulated 40s



(b) N_{con} simulated 10s



(c) dN_{br}/dt simulated 10s



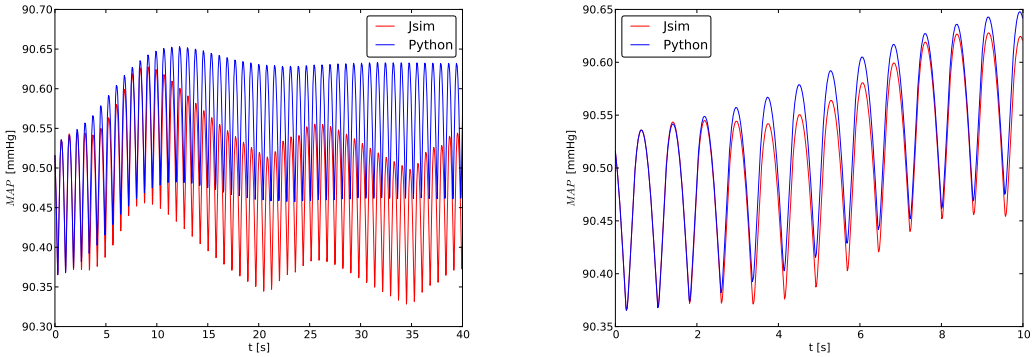
(d) N_{br} simulated 10s

Figure 4.3: Baroreceptors of CwB from JSim and Python

although we have tried to get as close as possible.

We use the scipy.odeint function for solving the CwB model. It is based on the ode-solver lsoda from Fortran library odepack. Another option would have been to use the scipy.ode function where the ode-solver is based on the Fortran library VODE. JSim uses CVODE, which is a translation of the VODE library into the C programming language. Both lsoda and VODE uses a method based on backward differentiation formulas (BDF) for solving stiff ODE problems. We used the scipy.odeint's output option to find that the CwB model is a stiff ODE system, as opposed to a non-stiff system, which would be solved by a different method. This information is from Python documentation and the Physiome project web pages [1].

We have not investigated further to establish for certain the reason for the differences between the runs of the CwB-model for JSim and Python.



(a) MAP simulated 40s

(b) MAP simulated 10s

Figure 4.4: MAP of CwB from JSim and Python

4.1.6 CPU time differences of the CwB-model

We want to measure the CPU time of running the CwB-model in JSim and Python. While working on the conversion it seemed that the JSim program was about 200 times faster than the python program. According to the "Introduction to the JSim GUI" that can be found from page [1], JSim compiles the MML source code into executable Java code before the program can be run. So when we are comparing JSim and Python, we are actually comparing Python and Java. We did not work on optimizing the python program, but there are many ways to improve straightforward python code.

In JSim there is a section of the GUI that shows the time when the model is executed and the time where the run is terminated, down to the accuracy of seconds. This is not a very accurate measure of CPU time, but given the limited manuals on JSim on the JSim Project web pages, it is the only available option.

To get as accurate results of CPU time differences as possible, we use a simulation time of 400s. For shorter simulation, the JSim measure time too inaccurately, see Table 4.1. The output in the JSim GUI for 400s simulation time looks like this:

```
12:27:49 Job "Run model MODEL" in progress...
12:28:43 terminated normally
```

We tried to run the JSim program with a simulation time of 4000s, but JSim wouldn't compile the code because there was insufficient memory available. By just looking at the error messages it seems that there was an overload of the data structures in Java, that JSim uses. The results for CPU-time and the longest simulation time that would compile is presented in Table 4.1.

The JSim GUI takes a long time when plotting the results of the 3500s run. The second time we adjusted the axis it actually crashed the JSim GUI.

Language	Simulation time	CPU-time	CPU-time/Simulation time
Python	4s	83.32s	20.830
JSim	4s	1s	0.250
Python	40s	958.44s	23.961
JSim	40s	6s	0.150
Python	400s	9745.77s	24.364
JSim	400s	54s	0.135
JSim	3000s	415s	0.138
JSim	3500s	531s	0.152
JSim	3600s		did not compile

Table 4.1: CPU-times for the CwB-model

We did get a look at the results though, and the N_{con} and MAP variables where stable, there was no divergence of the solution.

We only make one change to make the Python program faster. In JSim all variables can be plotted, not just the state variables. For instance there is a variable representing a very simple ECG signal, based only on heart rate. There are a few more of these equations just for plotting that do not affect the rest of the equations. These will not become part of the output of python integration, unless we specifically save them, so we remove them. JSim includes calculation of transmural and chamber pressures, in cases where the pressure outside the vessel is different from ground. Only chamber pressures are used in further calculations.

4.2 The Toska program

The original program was written in Pascal and operated from DOS. The program we consider here, is a modified version of an earlier model published in [14]. The earlier model was designed to explain the cardiovascular changes on thigh cuff inflation. The version we use, investigate changes at the onset of exercise. The medical data for the exercise experiments were gathered at two trials. In [13] they measure arterial pressure, stroke volume and heart beat, and in [3] they also measure femoral flow. An optimization algorithm is used to minimize the deviations between the recorded data and the model's predictions.

The conversion from Pascal to Matlab, was done by Sjur Urdson Gjerald, years before the start this project. All work related to this thesis is based on the Matlab program, if not stated otherwise. The original Matlab program can be found printed in a report [12]. During the work on this thesis on the conversion to Python, the program has been slimmed down to the essentials. The Matlab program runs slower than the original Pascal program. The

Pascal program is the version that they used in the articles [4, 14].

The new element in the Matlab program, compared to the Pascal program, is the added functionality needed to include measurements of femoral flow.

4.2.1 Untangle essential model code from the GUI

The Matlab program is organized so that it is operated from a graphic user interface (GUI). In order to have more control over parameter input, we want to remove the GUI, which constrain us to entering parameter values manually. For extensive testing of the program, we need to be able to start with randomized parameters. We also want to be able to reload files with optimized parameters to visualize results.

We start with exploring the model using the GUI. In Matlab we start Systemic_Circulation_Model.m, which basically sets up the GUI and ensures the user input results in calls to the right programs. The GUI includes sliders for changing all the parameters. Every time a parameter is changed, the model is rerun and plots are updated with the new result. This is very useful for getting an understanding of which parts of the model each parameter influences.

The optimization is quite slow in Matlab, but we get some plots of what happens to the function through the optimization process. The plots are updated in every iteration, along with current error and number of iterations done. This is one of the things that slow the optimization down.

The functions in the Matlab program that include GUI-commands are Intervall_skjekk.m, Optimaliser.m and Systemic_Circulation_Model.m.

Opening the Systemic_Circulation_Model.m file, we observe that it consists of almost entirely GUI commands. The different parts are ordered by a switch-case setup. This makes it easy to ensure that we keep all useful functionality while discarding everything else. Unlike most of the other functions, Systemic_Circulation_Model.py is written entirely from scratch. The only part of the GUI we kept were the plot routines. During the process we also made a Matlab version of the program without GUI. This made it easier to compare with the Python program.

The core model is the function ExecuteModel.m. From here the objects containing the model equations are initialized. Classes in Matlab are constructed so that files with class functions are put in a folder called @Classname. The present model contains eight classes representing the different parts of the circulation. The original Matlab program had only to a minor degree used the opportunity to put more than one function in each file. The entire model ends up containing in total 46 files. Programming with classes in Python is more practical than in Matlab, as we can easily put all eight classes into one file in Python while Matlab needs eight folders.

▶ SyPerTc	Sympathetic peripheral time constant	$T_{c,sp} = 10$s
▶ SyPerDel	Sympathetic peripheral delay	$d_{sp} = 6.00$s
▶ SyRrGain	Sympathetic rr-interval gain/sensitivity	$K_{sh,RR} = 0$
▶ SyRrTc	Sympathetic rr-interval time constant	$T_{c,sh} = 10$s
▶ SyRrDel	Sympathetic rr-interval delay	$d_{sh} = 1.00$s
▶ ParaRrTc	Parasympathetic rr-interval time constant	$T_{c,ph} = 0.25$s
▶ ParaRrDel	Parasympathetic rr-interval delay	$d_{ph} = 0.25$s
▶ MuscFlDly	Muscle flow delay	$d_{mf} = 1.8$s

Table 4.2: Constant parameters of the Toska model.

4.2.2 Parameters to the model's equations

The program needs two input files, the measured data for the given subject and a standard input file for model parameters. The model parameters can be subdivided depending on what we do with them after they have been imported from the input file. Some we set as constant, and five of the parameters are not used. Others we change in relation to the given subject's medical data. The remaining are the ones we optimize for. We include lists of the parameters to make it easier to check the program against the equations.

Parameters that are held constant during optimization, are the time constants and delays of the baroreflexes, and the delay in increase in muscle flow after the start of exercise, see Table 4.2. For reasons already discussed, we set the sympathetic RR-interval gain to zero, which means we choose to ignore the sympathetic influence on heart rate.

Parameters that depend on individual subjects are changed right after the input files are imported, they are listed in Table 4.3. When there is not any recorded data for muscle flow, it is assumed that Ex_{Cond}=0.15. The Windkessel compliance is the compliance of the artery, this is assumed to be related to the mass of the subject. The initial values are the average of the first 10s of the medical data of the respective subject. This is the period of rest before they start counting down to start of exercise.

The parameters that we focus on are the ones adjusted during optimization, listed in Table 4.4.

We find individual values for these parameters so that the model comes as close as possible to the medical data.

4.2.3 Old parameters from the development of the code

The input parameter file has a format that includes parameters to functions that is either not included in the original program, or has been dismissed as

▶ RestExFrac	Exercising fraction of body	Ex_{Cond}
▶ WkComp	Windkessel compliance	C
▶ MAP0	Mean arterial pressure	MAP_0
▶ RR0	Rr-interval	rr_0
▶ SV0	Stroke volume	SV_0
▶ MF0	Muscle flow	MF_0

Table 4.3: Parameters of the Toska model that depend on individual measurements.

▶ SyPerGain	Sympathetic peripheral gain/sensitivity	K_{sp}
▶ SyCoGain	Sympathetic contractivity gain/sensitivity	K_{sh}
▶ ParaRrGain	Parasympathetic rr-interval gain/sensitivity	K_{ph}
▶ AfterLoadSens	Sensitivity to afterload on stroke volume	K_a
▶ MitrFl	Mitral flow	Q_m
▶ MuscFlow	Muscle flow	Q_{mf}
▶ MuscFlTc	Muscle flow time constant	$T_{c,mf}$
▶ SetpStep	Baro set point step	$P_{s,step}$
▶ Barotccount	Baro set point tc countdown	$T_{c,set}$

Table 4.4: Parameters of the Toska model that we optimize against measurements.

▶ VenousFrac	Venous fraction
▶ VenousDly	Venous delay
▶ Barotc	Baro set point time constant
▶ BeforeFrac=1	Baro set point before fraction
▶ AdaptFac=0	Adapt factor for mean arterial pressure deviation

Table 4.5: Parameters of the Toska model from the development of the code.

not relevant to the system. These parameters are not mentioned in any of the articles, we list them in Table 4.5.

The venous parameters were supposed to model the venous return's effect on stroke volume. Two baroreflex setpoint time constants are included, but only one of them is used. The last two parameters are still a part of the program, and will influence the solution if they are changed. When AdaptFac is zero, the current mean arterial pressure's deviation from baroreflex set point is stored for use in the baroreflex signal. This is the same as if the T_BaroAdapt class is not present. Changing BeforeFrac will give the wrong increase in setpoint, $P_{s,step}$ for the baroreflexes.

Since these unused parameters are included in all the parameter input

files, we consider compatibility backwards in time and choose to keep them. We make it clear in the program that these are not in use and should not be optimized for. These parameters are more of a problem in the Matlab GUI where there are sliders to change parameters.

4.2.4 TimeConstDelay, the baroreflex signal function

The baroreflex signal in the program is calculated in an object of the class TimeConstDelay. Inside this class, the delay in the baroreflex was stored as a circular FIFO queue, and programmed as an object of the class DelayFifo. Stored in the FIFO queue is the MAP deviation from setpoint and the corresponding time, once for each heart beat. We can replace the DelayFifo object with a python dictionary, and still retain all the useful functionality. We choose a dictionary and not an array because the RR interval is not constant, so it will not have a fixed length. We could easily have used a list, though a dictionary makes it look more like the original program.

The reason why they used a circular FIFO queue, was storage and memory constraints when the original Pascal program was made, around 1994. The circular FIFO queue resets the write pointer to zero after a hundred values have been saved, and starts overwriting previous values to save memory. This could have been improved further by observing that the largest delay in the article, for the sympathetic influence on peripheral vessels, is $d_{sp} = 6.0$ s. If we consider the minimal duration of a heart beat, about 0.7s, then saving the values for twenty heart beats is more than sufficient.

We remove some tests on the values of the pointers in the FIFO queue, because the incidents that there are concerns about, will not happen. For instance, to get a circular FIFO queue overflow and start writing over values that has not been used yet, the delay must be $d > 100$ s, since the circular queue saves a hundred values.

The reason we decided to forego the circular FIFO queue is that a normal run of this program includes less than 300 heart beats, given by the duration of the experiments. Since having enough memory to run the program is no longer an issue, resetting the value of the write and read keys in a circular manner, will add complexity to reading the code with minimal effect on efficiency.

The baroreflex signal reacts to disturbance in arterial pressure. The time course and corresponding difference between arterial pressure and setpoint, called MapError, we now store in a python dictionary.

As mentioned in Section 3.3.1 the baroreflex signal in the program is not the same as the integral presented in the articles [4, 14]. We need an analysis of how the ”flow” through if-tests in TimeConstDelay determines what is stored in the signal, $B(t)$. The program is included in the appendix.

It is easy to see what happens before the simulation time passes the delay

time;

$$B(t_n) = \text{MapError}(t = 0) = P(0) - P_s(0), \quad \text{if} \quad t_n \leq d. \tag{4.3}$$

The signal is equal to the initial pressure deviation from setpoint, during the time before it is possible to go a delay d back.

The next equation is only exact if the RR-interval is constant, which is never the case, but it illustrates how the signal is formed.

$$B(t_n) \approx f\text{MapError}(t_{n-1} - d) + (1 - f)\text{MapError}(t_n - d) \quad \text{if} \quad t_n > d \tag{4.4}$$

The signal is based on two, or at the most three adjacent MapError, the deviations from setpoint. The delay is there to model the time between detection at the baroreceptor and response through efferent pathway. The weight of each term is decided from the factor f, where

$$f = e^{-\left(\frac{t_n - d - (t_{n-1} - d)}{T_c}\right)} = e^{-\frac{RR_n}{T_c}} \tag{4.5}$$

RR_n is the RR-interval at the current time, and T_c is a time constant depending on the pathway. The integral version of the signal sum up all previous MapError before the delay.

To describe the difference between the article and the code, we plot the two cases. We make a very simple program that calculate this integral

$$B(t) = \frac{1}{T_c} \int_0^{t-d} [P(t') - P_s(t')] e^{[(t'+d)/T_c]} dt' \tag{4.6}$$

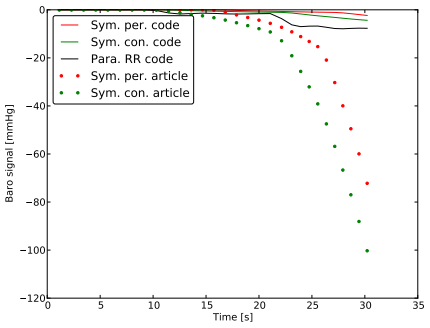
which is the same as in the article, only assuming the pressure is at the setpoint before we start our simulation.

One option for comparison is to use the time sequence and the corresponding MapError taken from a previous run of the program as input.

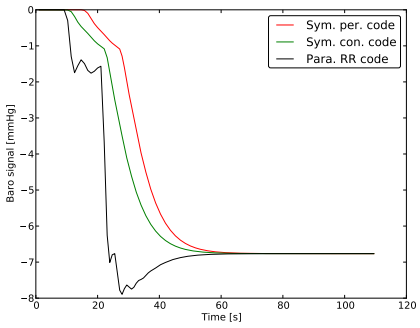
We use an optimized run for the subject GT from the newer measurements [3], the input to the baroreflexes are MapError and the responding time series, beat-to-beat. In Figure 4.5a we observe that the integral from the article is rapidly diverging. The parasympathetic signal to the heart is diverging so fast that it can not be plotted in a reasonable way. This is because of a small time constant, $T_{c,ph} = 0.25s$. We have not tried to use the integral in an actual run of the program, but would assume that the results would be unpredictable. Looking at the program's baroreflex in detail in Figure 4.5b we see that here a small time constant preserve the variation in the MapError for the parasympathetic signal, while a time constant of $T_{c,sh} = T_{c,sp} = 10s$ smoothes out the sympathetic signals.

Another option is to study the two signals with an artificial MapError input. In Figure 4.5c and 4.5d we set

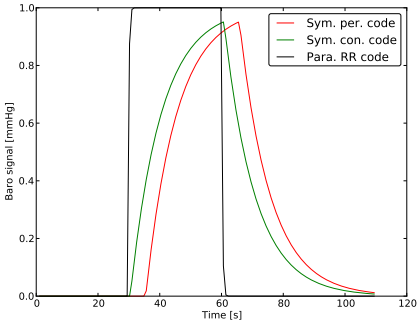
$$MapError = \begin{cases} 1 & 30\text{s} \leq t \leq 60\text{s}, \\ \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$



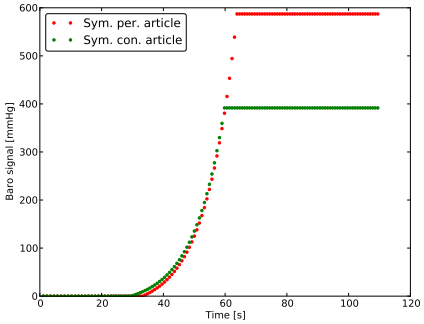
(a) MapError input from optimized run



(b) MapError input from optimized run



(c) Square MapError input



(d) Square MapError input

Figure 4.5: Baroreflex for subject GT

We observe that the signal from the code will decrease toward zero when there is no input, while the signal from the article will only go down if we have MapError of the opposite sign. If there is some other reason in the model for the mean arterial pressure never reaching the setpoint, so that it is either always below or above, then the signal from the article will diverge.

4.2.5 Include flow measurements from exercising muscle

In the original version of the program without measurements for femoral flow, the muscle flow parameter Q_{mf} represent the increase in flow in reaction to exercise, see Section 2.3.2. Since in [13] the femoral flow was not measured, it was estimated that the fraction of total peripheral conduction representing the legs was $Ex_{Cond} = 0.15$ in the original Pascal program.

When we have measurements for femoral flow, we want to find the fraction of conductance through the legs from these, instead of an estimate. In the experimental data files, Column 7 is the femoral flow as it was measured. Column 8 is the femoral flow data with mean at rest, MF_0, subtracted.

We want to optimize the results against the data in Column 8, since this

is the increase in flow, but first we need to find Ex_{Cond} for each subject from the fraction of total femoral flow to total flow. To do this we use mean values at rest. In the Matlab program Ex_{Cond} only became correct if we imported Column 7. If we imported Column 8 that we should optimize against, then Ex_{Cond} became zero. That means the program was run as if no part of the peripheral arteries were influenced by local metabolites when exercising.

To fix this problem, we import column 7 and find the mean at rest, MF_0. Then we calculate Ex_{Cond}. We need to subtract MF_0 from the measured femoral flow data from Column 7 to obtain the increase in flow, seen in Column 8. Another solution would have been to import both these columns.

The suggestion in practice:

1) Import Column 7 where $MF_0 \neq 0$

2) $Ex_{Cond} = \frac{MF_0}{SV_0 RR_0} \approx 0.15$

3) Column 7 $\rightarrow$ Column 7/ MF_0

4) Set MF_0=0

In the Matlab program it was implemented this way:

1) Import Column 8 where $MF_0 = 0$

2) $Ex_{Cond} = \frac{MF_0}{SV_0 RR_0} = 0$

Another suggestion is to make it identical to the case without data for femoral flow:

1) Import Column 8 where $MF_0 = 0$

2) Set $Ex_{Cond} = 0.15$

4.2.6 Constrain intervals for model parameters

We use an optimization function to fit the parameters in the model to actual data for individual patients. More information about the optimization algorithm can be found in Section 6.1, where we analyse the results.

A practical problem about this function is that it does not have any built in possibility to constrain the parameters that are optimized. We want to constrain the parameters to physiologically relevant values. Many of the parameters can not be negative.

The problem is more generally called a constrained optimization problem. There are many methods for solving these, but we chose not to go into this topic in this thesis. The scipy.optimize library only include a one-dimensional constrained optimization algorithm, and our problem is multi

dimensional. We choose the same optimization algorithm that is used in the Matlab program.

In the original program, intervals are only checked for sliders and inside some objects. The sliders in the GUI present the results. This means that while it seems that parameters we have found are within limits, the values we see on the screen might not actually be the ones that the optimization ended up with. We get the opposite problem when the parameters are changed inside objects. The function value that we are minimizing, will then not necessarily be based on parameters that the algorithm is using for optimization.

There is no perfect solution to this problem in python. One option is to check if the parameters are within the interval immediately after they have been suggested by the optimization algorithm. The parameters are then changed before the optimization of the set starts.

4.2.7 Create routines for different scenarios

The data presented in [4, 13], had measurements of mean arterial pressure, heart rate and stroke volume, and the model was optimized against these. In the newer measurements published in [3] they also measured the femoral flow of the subjects.

There has not been published an article where the program is used on the newer measurements that include femoral flow yet. So these results will be presented for the first time in this thesis. In this case the program should optimize against all four measurements.

It is important to have a simple user interface for changing conditions in the program for running different scenarios. We want the program to be able to work with both the older and newer measurements. As mentioned in section 4.2.5, we have two data columns with femoral flow, for the newer measurements. One of the actual measurements, and one where the average at rest is subtracted. In order to test the program, we want to be able to use both. For the old measurements we should not import, or optimize against, femoral data. Another thing we would like to change at will, is whether or not the Windkessel compliance is dependent on the mass of the subject.

4.2.8 Test program with random start values for optimizing

In the two articles [4, 14], they tested the program for each subject with only two different parameter starting points.

To test if the optimization is independent of initial parameters, we make a random number generator pick them. We use the random.uniform function, and choose intervals for the parameters that are narrower than those we allow for the optimization results. The reason for this is that many combinations of parameters make the model oscillate or diverge. We realised this while

using the sliders in the Matlab GUI, and then looked for intervals of each parameter where this did not happen. There might still be combinations of parameters within the intervals chosen that oscillate, but we do not want the intervals to be too narrow. If the optimization starts at such a point, it will normally not converge to a relevant result.

We also make a Matlab version of the program without GUI, to make it easier to compare the results with the python program. We save the random variables from the python program that we start the optimization with, to file. The intention is to start the Matlab version with the same random variables.

Chapter 5

Model development

5.1 Connecting Toska baromodel and CwB circulation

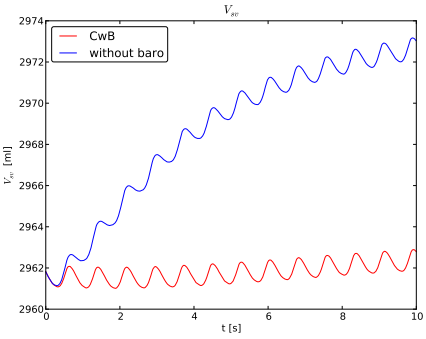
The Toska model is made specifically to investigate what role the baroreceptors play in regulating the body when it goes from a resting state to exercising. The CwB model simulate how baroreceptors deal with other kinds of stress to the system, like the Valsava manoeuvre, [8], but it does not include how metabolites influence the conductance in exercising muscles.

In the simple Toska circulation, the peripheral vessels are divided into a resting part and an exercising part which is the legs, connected in parallel. The CwB circulation has parallel vessels for the head, over the heart and the rest of the body in the systemic circulation. Since there is no division into parallel vessels between exercising and resting muscles in the CwB circulation, we will not activate the changes during exercise when we use the Toska baroreflexes. To accomplish this we set the parameter ExerciseState to be zero which represent rest, during the entire simulated time. This means that the baroreflex setpoint in the Toska model will be constant, and metabolites influence on conductance in exercising muscles will not be triggered.

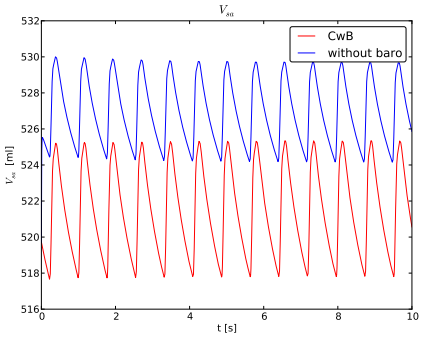
The parameters from the Toska model that we need are the ones related to the sympathetic and the parasympathetic pathways, plus the initial values MAP_0 and RR_0. The rest of the parameters are either related to exercise, or to the beat-to-beat circulation.

5.2 Running CwB circulation model without any baroreceptors

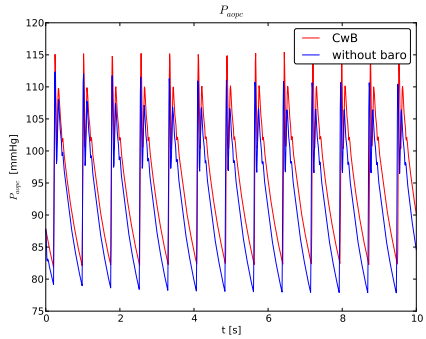
Disconnecting the baroreflexes is quite straight forward. In the case of the contraction the baroreflexes affects the ventricles but not the atria. We choose to set $a(F_{con}) = 1$, which makes the equations for atria and ventricles



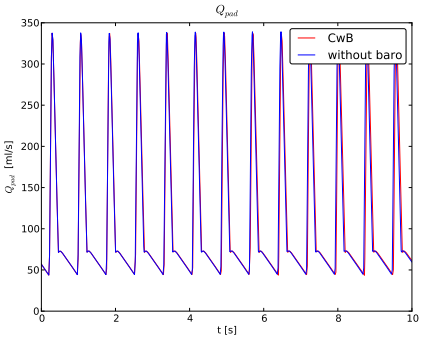
(a) Blood volume in systemic veins



(b) Blood volume in systemic arteries



(c) Pressure in the proximal part of the aorta



(d) Flow in the distal pulmonary artery

Figure 5.1: The CwB model in python with original baroreceptors and no baroreceptors

on the same form. We choose to keep the heart rate constant at the initial value. Vasoconstriction involves the most intricate equations. We choose to set the efferent sympathetic signal to $F_{vaso} = 0$ since this gives the passive arterial pressure: $P_{sa} = P_{sa,p}$.

We observe from Figure 5.1 that some parts of the system are more affected by baroreflexes than others. We will keep an eye on the systemic arteries and veins when we later try to fit the Toska baroreceptors to the model, since these are among the state variables that differ the most. The systemic veins are very compliant, so if blood volumes get shifted in the system, the veins will act like a reservoir. The systemic arteries are affected in two ways from the vasoconstriction, as both the resistance and the pressure involves F_{vaso}. In Figure 5.1c we note that the blood pressure is lower when the baroreflex is turned off.

5.3 Averaging beat for beat

The Toska baroreflex uses mean arterial pressure over a heart beat as input. Since all pressures in the CwB circulation vary we need to integrate them to

find average values. To find the mean arterial pressure we integrate P_{sa} over a heart beat using the simplest square method, adding for every variable time step, then divide by the current RR-interval.

The CwB program include formulas for the mean arterial pressure and cardiac output. They estimate MAP and CO by including the following differential equations. This models the rate of change, though we could not find a justification for the choice of time constant. The time constant smoothes out the pulsatile behaviour, so that we get closer to mean values of P_{sa} and Q_{lv}.

```
real COtau = 15  sec;  //Cardiac output equation time constant

COutput:t = (Qlv-COutput)/COtau;              //Cardiac output

MAP:t = (Psa-MAP)/COtau;              //Mean arterial pressure
```

Since this version of mean arterial pressure varies over the heart beat, it can not be used as input in the Toska baroreflex.

We chose to average the arterial pressure P_{sa} in the Toska baroreflex even though it is the pressure in the distal aorta P_{aod} that is driving the CwB baroreflexes, and the actual baroreceptors are located in the aorta.

The next step is to introduce each baroreflex to the circulation, observe differences between models, and make adjustments.

5.4 Contraction

In the Toska model, the sympathetic barosignal for contraction is included as a term in equation (3.14) that determines the stroke volume. The equation also includes a term for how afterload and filling time will affect the stroke volume. In a pulsatile model these two terms will be an integrated part of the flow out of the ventricle. It is then only the sympathetic barosignal that need to be added to the CwB model. In the CwB model the flow out of the left ventricle, Q_{lv} is:

$$Q_{lv} = \frac{P_{lvc} - P_{aopc}}{R_{av}} \tag{5.1}$$

from the pressure difference between the pressure of the left ventricle, P_{lvc} and the proximal aorta, P_{aopc} over the aortic valve resistance, R_{av}.

Stroke volume is defined as total flow output during one heart beat, or the RR-interval. For a pulsative system this is the integral of the flow out of the left ventricle, Q_{lv}

$$SV = \int_{RR} Q_{lv} dt \tag{5.2}$$

In the Toska model the way of including the baroreflex of contraction, is to add the resulting change to what would otherwise be the stroke volume,

SV_{orig}, see equation (3.14).

$$SV_{baro} = SV_{orig} + K_{sh}B_{sh} \tag{5.3}$$

We want to add the Toska contraction baroreflex to a pulsatile system, the barosignal, B_{sh} is constant over the duration of a heart beat. The following is the effect we want to accomplish

$$SV_{baro} = \int_{RR} Q_{lv,orig}dt + K_{sh}B_{sh} = \int_{RR} (Q_{lv,orig} + K_{sh}B_{sh}/RR)dt \tag{5.4}$$

We need to find a way that does not violate the conservation of volume, while giving a similar change in stroke volume.

The baroreflex influence the force of contraction, which determines the chamber pressure in the ventricle, P_{lvc}. This in turn change the flow out of the ventricle, Q_{lv} , see equation (5.1). Since Q_{lv} depends linearly on P_{lvc}, the first proposition on how to include the change in stroke volume is to change the ventricular pressure from the CwB model in the following way

$$P_{lvc} = P_{lvc,orig} + K_{sh}B_{sh}R_{av}/RR \tag{5.5}$$

and similarly for the right ventricle

$$P_{rvc} = P_{rvc,orig} + K_{sh}B_{sh}R_{puv}/RR \tag{5.6}$$

This is not acceptable because just raising the pressure in the ventricle by a constant value is not the same as changing the force with which the heart can contract.

In the CwB model the baroreflex signal for contraction is multiplied with the varying elastance of the ventricle, here E_{lv}, hence changing the elastance that determines the force of contraction, see equation (3.8).

In order to make the models as similar as possible, we propose this form when including the Toska baroreflex. The signal changes the varying elastance.

$$P_{lvc} = [1 + K_{sh}B_{sh}/RR]E_{lv}(V_{lv} - V_{lv,r}) - \frac{P_{x2}}{e^{V_{lv}/V_{x8}} - 1} + P_{pcdc} \tag{5.7}$$

We want the two baromodels to give equal pressure in the ventricle when there is no input signal. In the Toska model, $B_{sh} = 0$ is the case of no baroreflex signal, when the blood pressure is at the preferred level. In equation (3.8) from the CwB model $a(F_{con}) = 1$ is the signal that make the ventricular pressure on the same form as the atrial contraction which does not have baroreflex inervation. We include the barosignal in the right ventricle in the same way.

5.5 Heart rate

Changing the baroreceptors for heart rate is simple. In both cases the barosignal is influencing the heart rate directly. We do not need to make any adjustments, as we did in the contraction case. The heart rate that is used in the CwB model is only updated at the end of the heart beat anyway. Heart rate is the inverse of the RR-interval. We exchange equation (3.6) in the CwB model for equation (3.13) from the Toska model.

5.6 Peripheral vessels

In the CwB baroreflex the signal to the peripheral vessels affects both the pressure and the resistance of the systemic arteries. Setting $F_{vaso} = 0$ means that the arterial pressure is passive, $P_{sa} = P_{sa,p}$ and we have no baroreflex, like we see in Section 3.2. There is no logical way to make the Toska baroreflex affect pressure directly, as the model only applies to peripheral conductance. We therefore use the formula for passive pressure, $P_{sa,p}$ as the arterial pressure.

The Toska model has a total peripheral conductance that is the sum of two conductances through parallel vessels representing different parts of the body, see equation (2.54). We decide not to include metabolite influence in reaction to exercise. The reason is that to accomplish this in the CwB model, we would have to divide the systemic circulation into two equal branches, and then fit all the parameters in these two branches from scratch. This simplification means that we only include the sympathetic peripheral signal from the Toska model.

Conductance is defined as the inverse of resistance. For an electrical circuit, we can always find an equivalent resistor that gives the same current and potential drop as any combinations of resistors, see [15, p. 981-982]. It follows that equivalent resistance of a system is the inverse of the equivalent conductance.

$$R_{eq} = \frac{1}{G_{eq}} \tag{5.8}$$

In general for resistors in parallel, the equivalent conductance is the sum of the conductance of the components.

$$G_{eq} = G_1 + G_2 + \dots \tag{5.9}$$

The Toska model has two parallel components. Equation (5.8) ensures that the relation between the corresponding resistances is

$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \dots \tag{5.10}$$

In the systemic circulation of the CwB model we have vessels in series. Since the vessels have alternating resistance and compliance components,

using the equivalent resistance for resistors in series is an approximation. The volume change of a compliance vessel means that the two adjoining resistance vessels can have different flow. The equivalent resistance of resistors in series is derived from the fact that in series the current through each component must be the same. Ohm's law then gives

$$R_{eq} = R_1 + R_2 + \dots \tag{5.11}$$

The definition of conductance gives this relation for resistors in series

$$\frac{1}{G_{eq}} = \frac{1}{G_1} + \frac{1}{G_2} + \dots \tag{5.12}$$

When we do not include reactions to exercise, the Toska model affect the total peripheral conductance, G_{eq} in the following way, see equation (3.16)

$$\frac{1}{G_{eq}} = \frac{1}{G_{eq,orig}(1 + K_{sp}B_{sp})} = \frac{1}{G_{eq,orig}} \left(\frac{1}{1 + K_{sp}B_{sp}} \right), \tag{5.13}$$

where $G_{eq,orig}$ is the original total peripheral conductance without baroreflex. Since the CwB systemic circulation features vessels with resistance in series, we use equation (5.8) and (5.11). We find that the equivalent resistance of the systemic circulation, R_{eq} becomes

$$R_{eq} = R_{eq,orig} \left(\frac{1}{1 + K_{sp}B_{sp}} \right) = \left(\frac{1}{1 + K_{sp}B_{sp}} \right) (R_1 + R_2 + \dots) \tag{5.14}$$

where $R_{eq,orig}$ is the original total systemic resistance.

This means that if we were to get the same equivalent resistance of the systemic circulation for the two models, we should include the baroreflex term in the formula for all the resistances.

From physiology we know that it is the peripheral arteries that are affected by vasoconstriction. We propose to include the factor only in the formula for arterial resistance. From equation (3.9) the new form of the arterial resistance then becomes

$$R_{sa} = \left(\frac{1}{1 + K_{sp}B_{sp}} \right) \left(R_{sa,0} + K_r e^{4F_{vaso}} + K_r \left(\frac{V_{sa,max}}{V_{sa}} \right)^2 \right). \tag{5.15}$$

This can be justified since in the CwB model, the vasoconstriction signal only influences R_{sa} and P_{sa}. The efferent signal from the original baromodel is $F_{vaso} = 0$ so that the second term is also a constant.

Chapter 6

Results

This thesis is the first extensive testing of the Toska model. In the articles [3, 14] they tested for only two different starting values.

We divide this section into two parts. The first one contain the data from testing the Toska model, and the second shows the results from using the Toska baroreflexes with the CwB model.

6.1 Running the Toska model

When the conversion was complete, the first thing we wanted to do was to check it against the published results in article [4]. Like they noted in the article when commenting the constraints on the parameters, they chose two different sets of starting values for the nine adjusted parameters. First the mean of several model runs, and then secondly some above and some below within a physiologically reasonable range.

That they could start with the mean of previous model runs, imply that the model does not optimize to exactly the same result every time. We needed to check how these differences influence the results. In the article [4], they explained that the parameters were identical in all but three individuals in the two simulation runs. The parameters in question were Q_m and K_{sh} (the articles states B_{sh}, but that is the barosignal not an optimized parameter). Both of them affect stroke volume.

In the python program we chose the scipy.optimize.fmin function, which employs the Nelder-Mead downhill simplex algorithm. This is the same algorithm that the fminsearch function in Matlab uses, and it was the one included in the model. In the article [14], the minimization algorithm is also the downhill simplex method of Nelder and Mead, so this algorithm was also used in Pascal.

The convergence criteria xtol and ftol can not be set too strict given the noise in the recorded data, like we see in Figure 6.1. The optimization is terminated when reaching an accepted relative error in the parameters, xtol,

and a corresponding relative error in the function we want to minimize, ftol. In the Matlab program the values are set to xtol=10 and ftol=1, this ensure that in our case the ftol criteria terminates the optimization.

During the initial testing we found that if we made the ftol criteria slightly stricter, we got better results without having a greater fraction of results ending up on parameter boundaries. We choose tolerance ftol=0.1 for the error function.

The error function that we optimize over, is the sum of the squared differences between the model and the recorded data for mean arterial pressure, *RR*-interval and stroke volume. For the measurements from [3] it similarly includes femoral muscle flow. To get a dimensionless function, each sum is divided by the variance of the recorded data at rest, see the program in the appendix. The optimization procedure is therefore trying to reach a least squares solution. In the data there is a 10s rest period before counting down to start of exercise, we select only the 40s after rest to be optimized for, see program. According to [4] this was decided because after that time there will be humoral influence and other possible neural reflexes with a long time delay.

As already noted in Section 4.2.8, we choose relatively narrow parameter interval when we pick random start parameters. For the optimized parameters we allow wider intervals. Both intervals can be found in the program in the appendix.

We run optimization on 100 random parameter sets for each subject, and we apply the optimization algorithm four times on each set, with output from one being input to the next. The reason for this is that between the third and fourth optimization the number of function calls are notably smaller, and we have then usually reached a local minima even though the tolerance ftol is high.

For four of the subjects the optimization occasionally ends up with parameters that create an index error in the array that sums up the squared differences. This probably happens because of the *RR*-interval, but we have not managed to find such a result to visualize. It is a rare problem, usually occurring only once for every hundred set. This means that since we have to start the program again, there are more than 100 optimized parameter sets in the result files.

Before we can judge the results, we need to remove the sets where one of the parameters ended up on the boundary, since they are not actual local minima. In Tables 6.1 and 6.5, we show the fraction of sets that we accept for each subject, for the old and new subjects respectively. In addition to parameters on the boundary, we also dismiss solutions with an error function of $\epsilon = 700$ as not relevant. We observe that either more than half of the sets are accepted or almost all end up on the boundary.

We allow for parameters to be zero, even though this is the lower boundary for most parameters. The parameters that usually ends up on the upper

boundary, are the increase in baroreflex setpoint, $P_{s,step}$, the corresponding time constant, $T_{c,set}$ and the time constant for increase in muscle flow, $T_{c,mf}$.

The results with parameters on the boundary do in general make good curves, which may imply that the intervals could be widened.

The overall impression of any file of accepted parameter sets, is that usually a portion of the ones with the best error functions, has parameters that are equal to the precision of one significant digit. For some parameters it is slightly worse. This is not a very stable result, though it is certainly not random. In the articles [14] and [4] the parameters are published to a precision of two or three significant digits depending on the parameter.

In Tables 6.2, 6.3 and 6.4 we show the best accepted parameter set, then an average of all the accepted results are sent as input to the program to find the error function for the average parameters. Since there are many local minima, some of the results are better than others. We want to also make an average of the best results, and after studying the data we chose to select those with error function $\epsilon < \epsilon_{min} + 3$ to be certain that the results are close.

In Table 6.2 we also include the published results from the article [4]. We note that when we do this more thorough optimization, we end up with a better error function for all subjects.

For the subject MK the optimization ends up on the boundary every time. In the article [4] it is also noted that the time constant, $T_{c,mf}$ is constrained. They set it to $T_{c,mf} = 13.2s$ so here too we have a boundary solution, see Table 6.2.

The published parameters are in general quite different from those we reach. To show how this affects the solution, we choose to view the results of Table 6.2 for two subjects. We choose subjects that have a high acceptance ratio, while having different time courses. In Figure 6.1 we have subject EL. First we note how equal our three optimized and averaged curves are. We can also see this from the table. We choose this figure because the published values and the average results were quite similar.

The subject PL in Figure 6.2 is one of the subject where we have the greatest difference between our optimized results and the published values.

A trend in the data of Table 6.2, is that the set point increase $P_{s,step}$ is at least twice as large for our optimized curves compared to the published values. We observe that when setpoint increase doubles, then the sensitivity or gains, K_{ph},K_{sp} and K_{sh}, are halved. On the other hand there is not much difference in muscle flow $Q_{mf,max}$, the time constant for muscle flow increase $T_{c,mf}$ and afterload sensitivity, K_a. In the tables we have called $Q_{mf,max}$ for Q_{mf}. This is the notation in the tables of article [4]. The reason that $P_{s,step}$ affects the gains of the baroreflexes is quite simply that a large increase will give a large barosignal. A way to get similar time courses is to lower the gains so that the large barosignal has less effect.

There are too many dependent variables in the model to get a unique solution. The published values are reasonable solutions of the system just

not the best optimized ones. The fact that the set point increase is in general optimized to such a high value might not be physiologically plausible.

We also did an experiment with setting set point increase to $P_{s,step} = 7.0$ mmHg. This is the optimized value for the average of time course data for all the subjects, called global average response in the article [4]. We optimized over the eight remaining parameters. In this case all parameters settled closer to that of the published values. The ones where we saw the greatest difference, were KY and RJ since the published $P_{s,step}$ is higher here. We do not include the data from this experiment.

The results that are based on the newer measurements that also involves femoral flow, are displayed in Tables 6.3 and 6.4. We include simulations of two scenarios. In order to find out how predictive the model is, we run it with the same error function as the old measurements. Then we see if the optimized femoral flow increase matches the new measurements. The other case we are interested in, is when we include the squared difference between measurement and simulation of the femoral flow in the sum in the error function that we minimize over. We want to see the effect this has on the parameters compared to the previous optimization.

In Table 6.5 we see that there is a greater fraction of the newer subjects that are difficult to optimize. We observe in Table 6.3 the bad result for subjects C and G when the parameters we average over are far apart. When we run the model, we end up in a part of the parameter space that cause oscillations, hence the really high error functions.

We concentrate on the subjects where we have a good fraction of accepted results. We observe from Table 6.3 and 6.4 that the error function is larger when we also optimize for femoral flow. This is understandable since it involves an extra sum.

The interesting parameter when regarding different optimizing criteria, is $Q_{mf,max}$. Subject I is one of the poorer fit for $Q_{mf,max}$. In Figure 6.3, where the original error function is used, the increase in femoral flow is almost twice that of the measurements. In 6.4 we see the same subject for the new error function. We note that the original error function gives a better fit for the three other measurements; SV, MAP and RR-interval. From Table 6.3 we similarly see a doubling of Q_{mf} for the original error function. The other parameters are slightly shifted to accommodate this change.

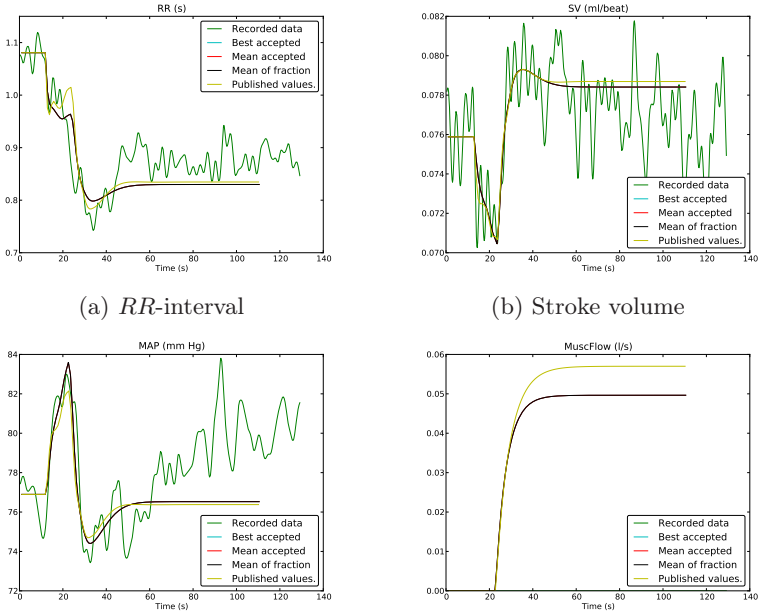
In Figures 6.5 and 6.6 we show the subject M. Here we see one of the best predictive results for the original error function. This we also note from the Table 6.4. In subject M we have the closest resemblance for all the parameters for the two error functions.

In section 2.3.2 we noted that the formula for conductance in exercising muscle in the programs involve baseline mean arterial pressure, MAP_0, while in the article they use MAP. In all the measurements that we have shown here, we kept the program the way it was, to facilitate comparisons.

We did an experiment with using MAP with the old measurements. The

	EL	HG	HV	KY	LJ	LM	MK	PL	RJ	TK
Fraction	0.78	0.92	0.69	0.05	0.88	0.64	0	0.62	0.35	0.71

Table 6.1: Fraction of accepted parameter sets when model is run with the measurements from [13].



(a) *RR*-interval (b) Stroke volume

(c) Mean arterial pressure (d) Flow in exercising muscle

Figure 6.1: Plot of the model run in Table 6.2 for subject EL

difference in the parameters was minor compared to the differences within a run of 100 parameter sets. There was no parameter where we could clearly see that the two files was not related to the same subject.

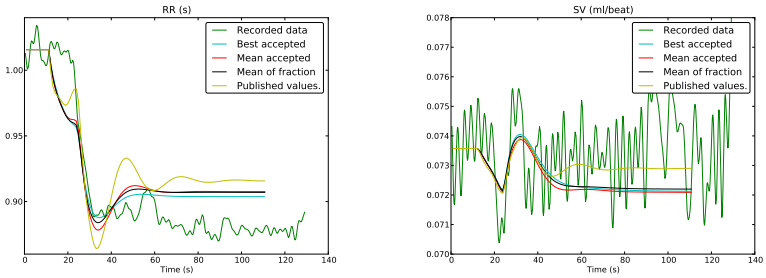
6.2 Connecting the two models

We are interested in finding out if a baroreflex made for the simple beat-to-beat circulation model is general enough to work for another circulation.

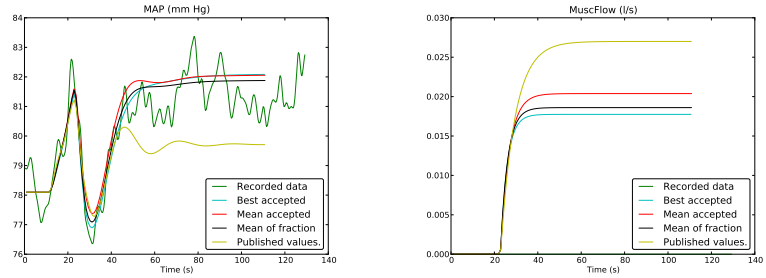
We start by testing the default Toska model parameters, the ones that we always initialize the model with before optimizing. The parameters that are relevant to change are shown in Table 6.6. In order to see the effect that the baroreceptors have on the system, we plot three state variables with the default parameters, and then the same state variables when the parameters are fitted to the CwB circulation. Fitting the Toska parameters to the CwB circulation is similar to fitting them to a given subject. This is always a part

	Weight (kg)	Setpoint Increase (mmHg)	$T_{c,set}$ (s)	K_{ph} (ms /mmHg)	K_{sp} (l /mmHg)	K_{sh} (ml /mmHg)	Q_{mf} (l/s)	$T_{c,mf}$ (s)	Q_m (ml/s)	K_a (ml /mmHg)	Error
EL											
a	64	18.843	5.773	13.234	0.022	0.145	0.049	4.951	0.000	-0.780	176
b	64	17.168	5.178	15.149	0.026	0.185	0.050	5.052	0.762	-0.794	188
c	64	17.810	5.523	14.411	0.025	0.171	0.050	5.131	0.545	-0.792	182
d	64	7.500	2.950	30.700	0.068	0.360	0.057	6.410	0.000	-0.980	205
HG											
a	73	9.540	8.130	15.589	0.028	0.296	0.022	1.065	24.159	-0.551	134
b	73	9.415	7.856	16.699	0.029	0.240	0.022	1.048	21.552	-0.526	148
c	73	9.553	8.069	15.616	0.028	0.285	0.022	1.076	23.614	-0.543	134
d	73	5.000	6.960	26.800	0.053	0.400	0.023	1.900	21.600	-0.810	157
HV											
a	56	28.804	15.672	14.507	0.008	0.314	0.048	1.541	19.457	-0.368	270
b	56	19.603	12.881	21.390	0.016	0.355	0.051	2.013	17.331	-0.431	390
c	56	26.752	15.537	15.454	0.010	0.334	0.049	1.724	19.541	-0.391	271
d	56	5.000	1.790	40.700	0.027	0.350	0.050	2.560	11.600	-0.600	323
KY											
a	65	27.915	21.857	20.278	0.056	0.390	0.048	7.372	0.000	-0.571	116
b	65	25.391	18.027	35.256	0.034	0.723	0.037	5.474	0.075	-0.546	369
c	65	28.939	24.642	20.894	0.047	0.518	0.040	6.297	0.000	-0.678	115
d	65	13.500	16.280	55.200	0.139	1.210	0.040	5.160	0.000	-0.660	186
LJ											
a	73	16.725	9.673	15.564	0.031	0.177	0.031	2.406	12.733	-0.588	238
b	73	16.162	9.131	16.380	0.032	0.108	0.031	2.368	8.715	-0.586	241
c	73	15.919	9.418	16.420	0.033	0.166	0.032	2.492	11.614	-0.593	239
d	73	8.000	8.650	33.500	0.077	0.400	0.035	3.510	6.800	-0.890	289
LM											
a	61	16.320	7.484	17.846	0.000	0.220	0.021	8.378	0.000	-0.739	249
b	61	14.594	7.128	21.007	0.002	0.316	0.023	9.213	2.175	-0.720	256
c	61	15.127	7.197	20.046	0.000	0.286	0.021	8.741	1.568	-0.725	254
d	61	8.500	5.330	39.800	0.016	0.540	0.028	11.460	0.000	-0.770	270
MK											
d	62	6.000	4.510	27.200	0.075	0.020	0.042	13.200	3.400	-0.980	253
e	62	30.000	10.704	5.776	0.028	0.078	0.067	15.000	14.861	-0.728	135
PL											
a	83	17.191	13.480	8.366	0.017	0.018	0.017	2.908	0.000	-0.396	236
b	83	13.571	13.764	14.942	0.033	0.033	0.021	4.318	0.632	-0.414	409
c	83	15.008	13.225	10.151	0.023	0.025	0.019	3.301	0.378	-0.390	248
d	83	5.500	8.610	25.700	0.090	0.060	0.027	6.640	0.000	-0.470	263
RJ											
a	75	19.999	17.126	17.317	0.001	0.665	0.009	0.102	0.000	-0.844	280
b	75	17.208	22.050	42.286	0.010	1.732	0.009	0.471	0.820	-0.865	439
c	75	20.229	18.069	18.669	0.002	0.776	0.009	0.144	0.201	-0.880	282
d	75	11.000	13.000	38.400	0.023	1.130	0.012	0.900	0.000	-0.670	297
TK											
a	57	17.462	3.063	10.939	0.023	0.129	0.038	2.369	2.074	-0.717	147
b	57	17.095	2.853	11.376	0.024	0.130	0.038	2.373	1.814	-0.719	150
c	57	16.941	3.008	11.390	0.024	0.133	0.039	2.410	2.001	-0.719	147
d	57	7.500	1.730	24.200	0.058	0.230	0.041	3.400	0.000	-0.890	194

Table 6.2: Model applied on the measurements from [13]. a) Best accepted optimization reached, b) Mean of accepted parameter sets, c) Mean of fraction of accepted parameter sets, d) Published values, e) Best optimization reached

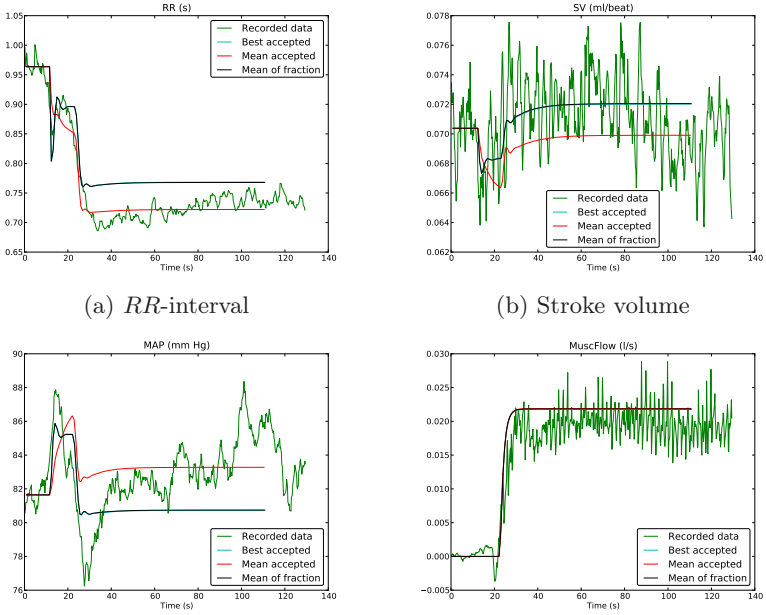


(a) *RR*-interval (b) Stroke volume



(c) Mean arterial pressure (d) Flow in exercising muscle

Figure 6.2: Plot of the model run in Table 6.2 for subject PL



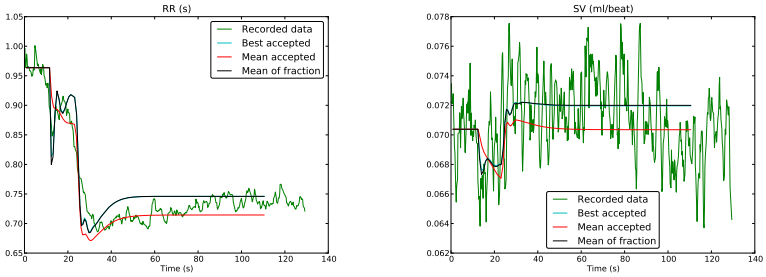
(a) *RR*-interval (b) Stroke volume

(c) Mean arterial pressure (d) Flow in exercising muscle

Figure 6.3: Plot of the model run in Table 6.3 for subject I, with including muscle flow data in error function

	Weight (kg)	Setpoint Increase (mmHg)	$T_{c,set}$ (s)	K_{ph} (ms /mmHg)	K_{sp} (l /mmHg)	K_{sh} (ml /mmHg)	Q_{mf} (l/s)	$T_{c,mf}$ (s)	Q_m (ml/s)	K_a (ml /mmHg)	Error
B											
d	55	1.210	0.128	82.032	0.068	0.000	0.008	8.965	22.799	0.000	419
e	55	1.210	0.128	82.032	0.068	0.000	0.008	8.965	22.799	0.000	419
f	55	1.210	0.128	82.032	0.068	0.000	0.008	8.965	22.799	0.000	419
C											
a	63	6.219	6.132	113.351	0.000	6.802	0.025	0.225	45.236	-0.759	576
b	63	7.608	13.831	168.890	0.002	8.426	0.027	1.749	18.244	-1.777	11449
c	63	6.219	6.132	113.351	0.000	6.802	0.025	0.225	45.236	-0.759	576
d	63	23.146	28.006	43.746	0.024	3.515	0.040	0.100	50.235	-1.147	446
e	63	27.643	26.767	31.377	0.048	2.585	0.069	6.275	30.934	-2.278	533
f	63	23.146	28.006	43.746	0.024	3.515	0.040	0.100	50.235	-1.147	446
E											
a	82	6.964	0.605	9.768	0.000	0.136	0.006	0.562	36.216	-0.437	458
b	82	6.835	0.573	10.278	0.000	0.158	0.006	0.619	33.705	-0.508	458
c	82	6.973	0.558	9.942	0.000	0.143	0.006	0.591	34.698	-0.469	458
d	82	9.260	0.721	7.432	0.018	0.000	0.023	0.100	6.065	-0.824	222
e	82	9.186	0.538	7.504	0.018	0.008	0.023	0.103	6.997	-0.823	223
f	82	9.149	0.529	7.527	0.018	0.005	0.023	0.103	6.753	-0.823	223
F											
a	90	4.540	0.101	45.967	0.000	0.000	0.015	0.104	19.259	0.000	367
b	90	2.627	0.107	92.822	0.000	0.145	0.017	1.804	19.586	-0.901	450
c	90	4.540	0.101	45.967	0.000	0.000	0.015	0.104	19.259	0.000	367
d	90	17.484	29.500	26.565	0.020	0.000	0.036	2.400	9.137	-0.346	146
e	90	9.527	16.123	70.815	0.012	0.456	0.030	1.938	7.291	-0.531	264
f	90	17.484	29.500	26.565	0.020	0.000	0.036	2.400	9.137	-0.346	146
G											
a	72	1.112	0.111	116.529	0.000	1.293	0.043	2.897	0.080	-2.661	437
b	72	0.898	3.692	131.122	0.000	4.127	0.037	3.514	6.211	-1.505	13608
c	72	1.112	0.111	116.529	0.000	1.293	0.043	2.897	0.080	-2.661	437
d	72	29.950	27.747	19.265	0.066	0.346	0.147	9.544	0.148	-0.777	135
e	72	17.422	15.944	49.290	0.085	0.592	0.120	8.229	0.341	-1.067	1435
f	72	28.768	28.121	20.217	0.068	0.376	0.145	9.385	0.613	-0.808	135
H											
a	55	9.252	12.182	65.763	0.083	0.182	0.035	6.367	0.000	-0.315	319
b	55	9.247	12.200	66.404	0.082	0.203	0.035	6.329	0.564	-0.284	319
c	55	9.261	12.172	65.593	0.083	0.182	0.035	6.347	0.365	-0.294	319
d	55	9.335	12.083	63.701	0.015	0.125	0.021	3.902	0.000	-0.279	279
e	55	9.108	12.427	69.665	0.012	0.364	0.020	4.030	2.042	-0.250	283
f	55	9.210	12.199	65.476	0.012	0.209	0.020	3.940	1.265	-0.251	279
I											
a	68	5.963	0.121	28.402	0.000	0.310	0.022	1.542	3.450	-0.576	416
b	68	10.027	6.202	28.780	0.000	0.332	0.022	1.566	5.115	-0.760	758
c	68	5.932	0.276	28.673	0.000	0.309	0.022	1.552	3.136	-0.591	416
d	68	5.454	0.859	34.857	0.038	0.248	0.039	2.168	0.000	-0.606	302
e	68	12.103	8.521	25.099	0.024	0.228	0.038	1.971	1.479	-0.600	498
f	68	5.538	0.651	34.515	0.038	0.274	0.039	2.141	0.633	-0.611	303
J											
a	50	22.149	9.354	25.165	0.019	0.143	0.021	4.239	0.000	-0.670	234
b	50	18.104	10.294	52.612	0.037	0.622	0.021	4.462	0.983	-0.725	358
c	50	21.414	9.471	27.177	0.021	0.175	0.021	4.434	0.294	-0.675	235
d	50	17.739	10.757	37.851	0.065	0.240	0.038	9.234	0.000	-0.681	205
e	50	16.396	11.406	67.754	0.041	0.928	0.022	5.586	0.725	-0.791	299
f	50	19.090	10.617	34.565	0.056	0.207	0.036	8.569	0.072	-0.670	210

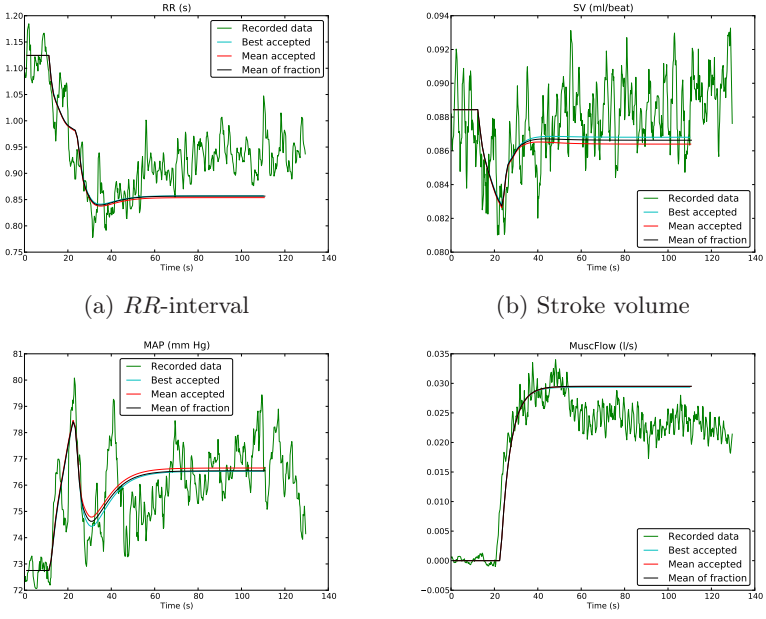
Table 6.3: Model applied on the measurements from [3]. a) Best accepted optimization reached, b) Mean of accepted parameter sets, c) Mean of fraction of accepted parameter sets, d) Best accepted optimization reached, no muscle flow data, e) Mean of accepted parameter sets, no muscle flow data, f) Mean of fraction of accepted parameter sets, no muscle flow data



(a) *RR*-interval (b) Stroke volume

(c) Mean arterial pressure (d) Flow in exercising muscle

Figure 6.4: Plot of the model run in Table 6.3 for subject I, without including muscle flow data



(a) *RR*-interval (b) Stroke volume

(c) Mean arterial pressure (d) Flow in exercising muscle

Figure 6.5: Plot of the model run in Table 6.3 for subject M, with including muscle flow data in error function.

	Weight (kg)	Setpoint Increase (mmHg)	$T_{c,set}$ (s)	K_{ph} (ms /mmHg)	K_{sp} (l /mmHg)	K_{sh} (ml /mmHg)	Q_{mf} (l/s)	$T_{c,mf}$ (s)	Q_m (ml/s)	K_a (ml /mmHg)	Error
L											
a	54	26.153	29.119	41.974	0.034	0.756	0.018	4.228	0.187	-0.543	272
b	54	22.035	26.413	67.726	0.052	0.791	0.018	2.896	6.228	-0.225	286
c	54	26.680	26.975	37.071	0.030	0.652	0.019	4.205	0.123	-0.535	273
d	54	18.582	24.715	82.941	0.000	1.646	0.008	0.100	12.078	-0.229	246
e	54	19.823	27.800	86.214	0.002	1.394	0.007	0.101	14.218	-0.096	247
f	54	19.823	27.800	86.214	0.002	1.394	0.007	0.101	14.218	-0.096	247
M											
a	63	16.688	8.787	20.690	0.013	0.413	0.029	4.467	10.895	-0.821	335
b	63	15.645	8.569	23.053	0.015	0.383	0.029	4.544	7.572	-0.875	340
c	63	15.607	8.569	22.675	0.015	0.413	0.030	4.516	8.973	-0.860	336
d	63	24.478	10.031	13.508	0.003	0.342	0.023	3.830	16.307	-0.736	319
e	63	15.319	8.943	27.528	0.019	0.544	0.031	5.374	6.803	-0.934	413
f	63	18.353	9.202	19.216	0.006	0.382	0.024	4.126	10.751	-0.829	325
TO											
a	65	28.008	20.693	15.546	0.001	0.256	0.021	2.578	3.204	-0.759	239
b	65	22.346	21.212	32.989	0.003	0.636	0.021	2.577	1.417	-0.871	790
c	65	26.881	20.459	16.382	0.001	0.249	0.021	2.611	1.673	-0.787	240
d	65	20.418	17.078	21.156	0.017	0.269	0.032	3.559	1.694	-0.700	224
e	65	16.326	18.537	35.880	0.045	0.532	0.039	4.710	0.927	-0.793	281
f	65	18.553	17.330	25.550	0.027	0.313	0.034	4.018	0.595	-0.744	240

Table 6.4: Model applied on the measurements from [3]. a) Best accepted optimization reached, b) Mean of accepted parameter sets, c) Mean of fraction of accepted parameter sets, d) Best accepted optimization reached, no muscle flow data, e) Mean of accepted parameter sets, no muscle flow data, f) Mean of fraction of accepted parameter sets, no muscle flow data

Fraction	B	C	E	F	G	H	I	J	L	M	TO
With	0	0.03	0.85	0.02	0.02	0.92	0.52	0.71	0.07	0.89	0.54
Without	0.01	0.03	0.79	0.03	0.09	0.71	0.70	0.65	0.04	0.85	0.60

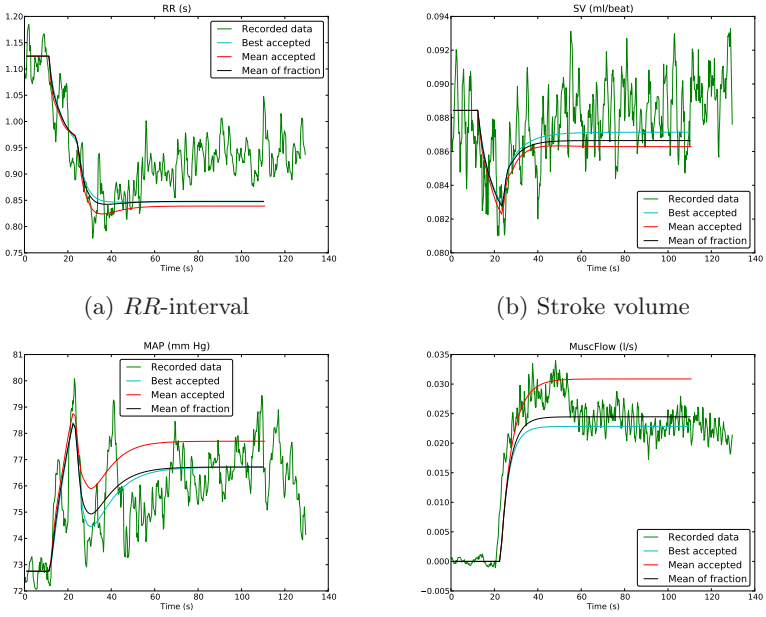
Table 6.5: Fraction of accepted parameter sets when model is run with the measurements from [3], with and without including the muscle flow measurements.

of the Toska model, since it is patient specific.

We choose the state variables V_{aod}, V_{sv} and V_{sa} since when the Toska parameters have been fitted, the largest discrepancies between the two baroreflex models are in these state variables. All the other state variables have small deviation in heart rate as the major difference.

Figure 6.7 shows the blood volume of the systemic veins. We look at each individual baroreflex and the combined effect for both the default Toska parameters and CwB. We see that the effect of the Toska baroreceptors are too strong for this case, the blood volume is shifted so that the veins either gain or loses extensive amounts of blood.

In Figure 6.11 we observe the main reason for the drastic changes to the system. The default Toska baroreceptors are correcting what is conceived to be a high blood pressure for the systemic arteries. Decrease in volume



(a) *RR*-interval (b) Stroke volume

(c) Mean arterial pressure (d) Flow in exercising muscle

Figure 6.6: Plot of the model run in Table 6.3 for subject J, without including muscle flow data

means decrease in pressure, here in the distal aorta, V_{aod}.

When one compares Figure 6.9 and 6.10 that show arterial volume, V_{sa}, it seems that the default gives the better fit. This is the only state variable where that happens. The reason is that we have not included the peripheral baroreflex effect on arterial pressure. This means that while the volumes in Figure 6.9 are close, the corresponding pressure, P_{sa} will be lower in the case of the Toska baroreflex, see Figure 3.2. The default baroreflex is trying to lower the mean arterial pressure which is calculated from P_{sa}.

We want to fit the baroreflex parameters to the CwB circulation, but we realize that applying the optimization algorithm will be too time consuming. We need to simulate at least 10s to see the full effect of the baroreflex, and an average run of the optimization algorithm makes about 300 function calls.

The models are in any case quite different so a perfect match is not to be expected. There are five parameters in the baromodel that we need to change. These are listed in Table 6.6.

We make the fit by considering the steady state of the CwB model and visual comparison of the plots of all the state variables.

We need to change RR_0 and MAP_0 because we want to drive the system toward the solution that is stable in the original CwB-model. From the JSim program we find that the heart rate varies little during the simulation. The

initial value is $HR = 77/60s$. To find a good choice for baro setpoint, MAP_0 we look at the variables P_{sa} and MAP in JSim.

In order to figure out the gains of the barosignal, we look at one reflex at the time with the new estimation for baro setpoint. We find that the biggest difference is now in the peripheral vasoconstriction. It is too sensitive, so we turn the gain down. The value we end up with is well within the interval containing the optimized SyPerGain/K_{sp} values for the subjects, see Table 6.2 and 6.6. For a reasonable good fit, the other variables can remain unchanged. This result is the second column in Table 6.6.

Since we want to have a better understanding of all the parameters, we see how close the two baromodels can get. From the fitted parameters in Table 6.6, we show the plots for the same state variables that we present with the default parameters.

When it comes to heart rate, we find that keeping RR_0 the same as the initial condition in the CwB model, gives the overall best results. Even though the resulting RR-intervals are slightly shorter than that of the CwB baroreflex, see Figure 6.12 of the volume in the aorta. Of the three state variables we are plotting, this one is most directly affected by heart rate. We see from Figure 6.12a that when the heart rate baroreflex is not included, the heart rate is more similar. The reason for the RR-intervals getting shorter with baroreflex, is that the mean arterial pressure is slightly lower than the set point. This means that in order to make it a closer fit, it is the set point, MAP_0 that should be changed, not RR_0.

The sympathetic signal that influence contraction, has a small contribution to the system when the system is close to right mean arterial pressure for both baromodels, compare Figure 6.8c and the case with no baroreflex in Figure 5.1a. On the other hand we know from Figure 6.7c that when the system is not close to the baro setpoint, this gain will have a marked effect.

We conclude the fitting when we are content with the result in Figure 6.8d of the systemic veins.

The next step when testing if the Toska baromodel works similar to the CwB baromodel, is to introduce a perturbation in an interesting parameter.

A healthy valve has low resistance to forward flow and block backflow. If a valve defect increase resistance it is called valve stenosis. Valve defects can be a congenial malfunction or a result of certain diseases, as explained in [6, p 45]. We change aortic valve resistance from $R_{av} = 0.0001$ mmHg s/ml to $R_{av} = 1$ mmHg s/ml . We test out different values in JSim, and for lower values of R_{av} there are little difference to the system. The choice is not based on medical data for valve stenosis. In Figure 6.13 we see the results for the CwB baroreflex, the fitted Toska baroreflex and without baroreflexes.

We see in Figure 6.13c that the baroreflex quickly counter the effect of less flow out of the left ventricle. When there is no baroreflex, the blood volume in the distal aorta falls to a lower plateau and settles. Both the baroreflex models react to the change, and within 10s the blood volume and

Parameter	Starting point	First attempt	Fitted parameters
K_{sp}	0.054100590	0.030000000	0.028000000
K_{sh}	0.000345964	0.000345964	0.000345964
K_{ph}	0.042343168	0.042343168	0.010000000
MAP_0	74.740546667	90.000000000	91.000000000
RR_0	1.102551111	0.779220779	0.779220779

Table 6.6: Connecting the Toska baroreflex to the CwB-model

Python	CPU-time	CPU-time/Simulation time
CwB baroreceptors	231.61	23.16
No baroreceptors	43.84	4.38
Toska baroreceptors	43.88	4.39

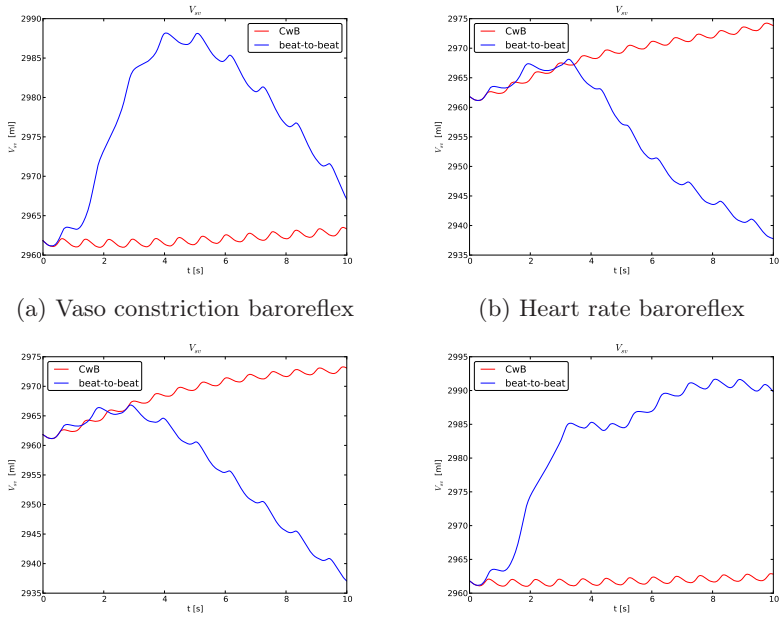
Table 6.7: CPU-times for the CwB-model with different baroreflexes, simulation time is 10s.

hence the blood pressure in the distal aorta is back to the level of the initial value. The Toska baroreflex reacts slightly faster than the CwB baroreflex. To find out which is the most realistic we would have to compare with measurements. The fact that the volume stabilize toward the same value for both baroreflexes shows that the merging of the Toska model with the CwB circulation can work well.

In Figure 6.13b we find that the Toska baromodel keep the blood volume in the systemic arteries higher than the case without baroreflex, to keep up the arterial blood pressure. The CwB baroreflexes react to the fall in blood volume, but reach a lower new level. The reason for the lower blood volume is again that the peripheral baroreflex also alters the pressure P_{sa}.

With aortic valve stenosis, parts of the blood volume is shifted from the systemic circulation to the pulmonary circulation. In Figure 6.13a we see that the volume in the systemic veins decrease with on average 110ml for the three cases. The blood volume in the pulmonary vein in Figure 6.13a in reaction increase with 150ml. Studying all the state variables, we observe in general that the pulmonary system is less effected by the baroreflexes.

One reason to choose the Toska baroreflex, is that it does not involve ODEs, which means that the CPU time for a simulation is much faster than for the CwB baroreflex, see Table 6.7. We note that the N_{br} barosignal in Figure 4.3d gets a sharp spike every heartbeat which is very difficult to solve for the ODE solver.



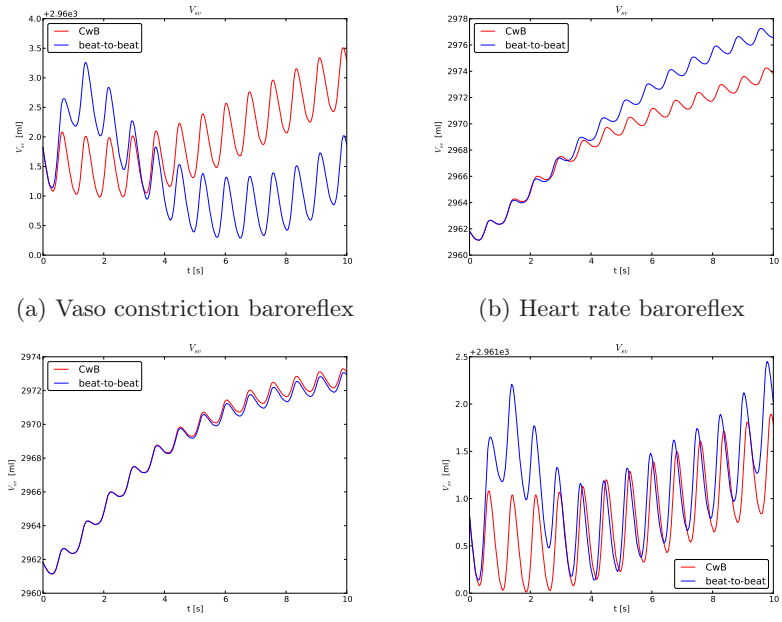
(a) Vaso constriction baroreflex

(b) Heart rate baroreflex

(c) Contraction baroreflex

(d) All baroreflexes

Figure 6.7: The difference between the Toska and the CwB baroreflexes, default Toska parameters, shown for the volume of the systemic veins.



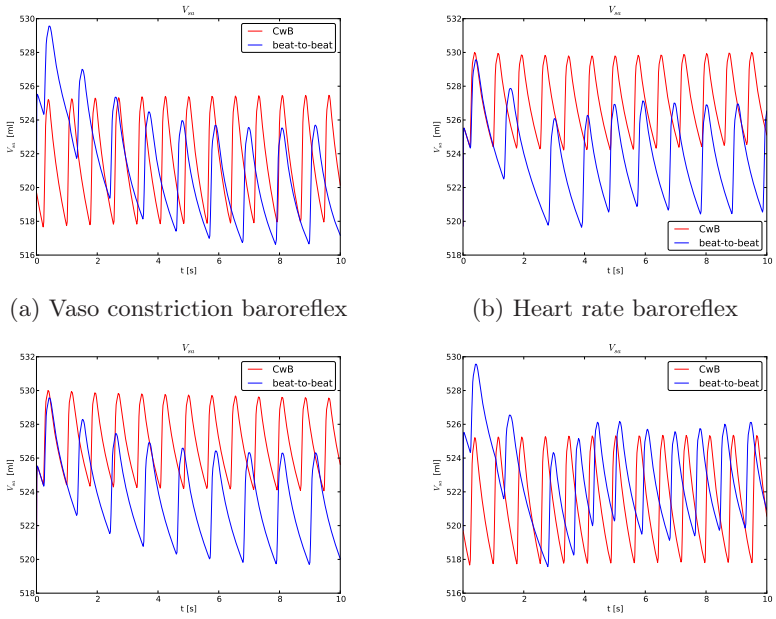
(a) Vaso constriction baroreflex

(b) Heart rate baroreflex

(c) Contraction baroreflex

(d) All baroreflexes

Figure 6.8: The difference between the Toska and the CwB baroreflexes, fitted Toska parameters, shown for the volume of the systemic veins.



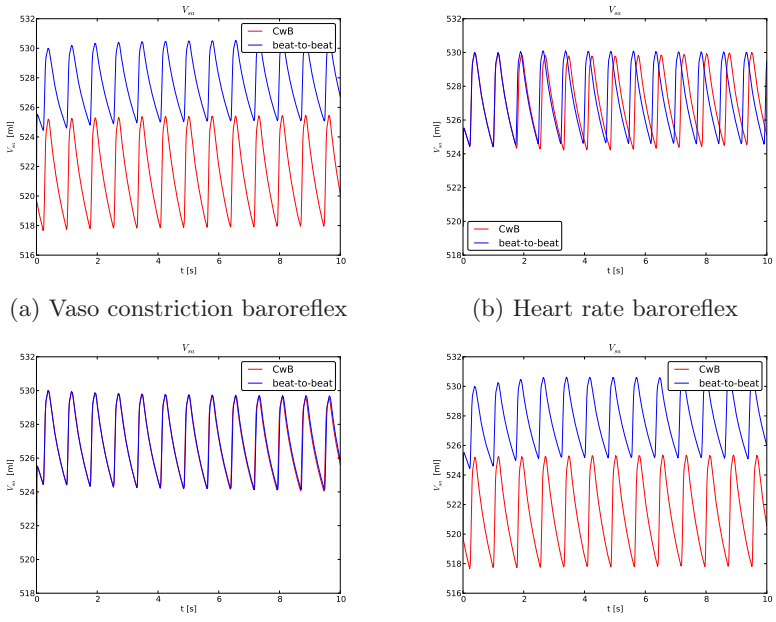
(a) Vaso constriction baroreflex

(b) Heart rate baroreflex

(c) Contraction baroreflex

(d) All baroreflexes

Figure 6.9: The difference between the Toska and the CwB baroreflexes, default Toska parameters, shown for the volume of the systemic arteries.



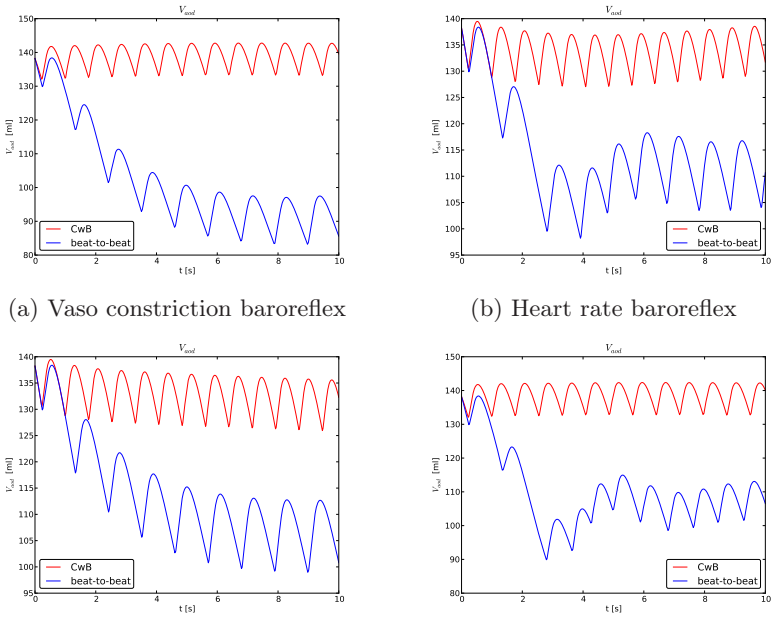
(a) Vaso constriction baroreflex

(b) Heart rate baroreflex

(c) Contraction baroreflex

(d) All baroreflexes

Figure 6.10: The difference between the Toska and the CwB baroreflexes,fitted Toska parameters, shown for the volume of the systemic arteries.



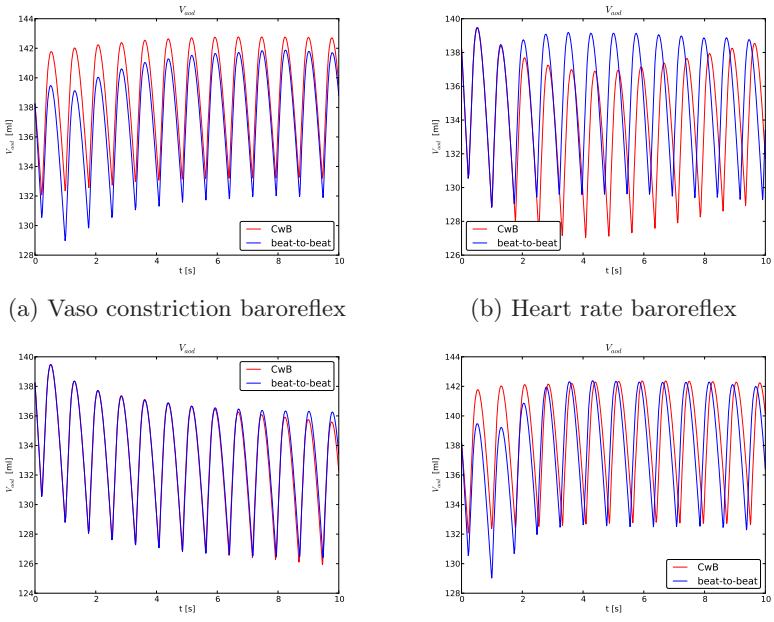
(a) Vaso constriction baroreflex

(b) Heart rate baroreflex

(c) Contraction baroreflex

(d) All baroreflexes

Figure 6.11: The difference between the Toska and the CwB baroreflexes, default Toska parameters, shown for the volume of the distal aorta.



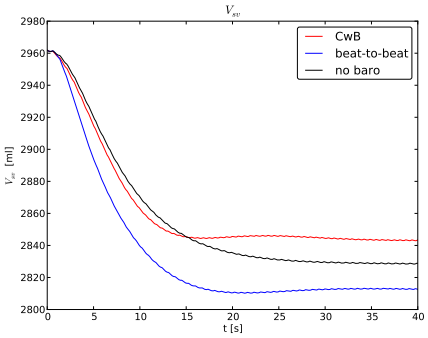
(a) Vaso constriction baroreflex

(b) Heart rate baroreflex

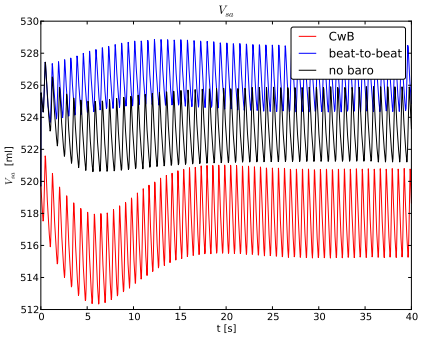
(c) Contraction baroreflex

(d) All baroreflexes

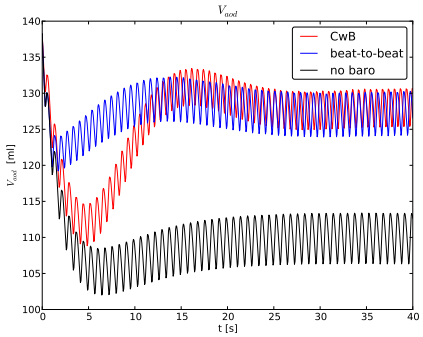
Figure 6.12: The difference between the Toska and the CwB baroreflexes, fitted Toska parameters, shown for the volume of the distal aorta.



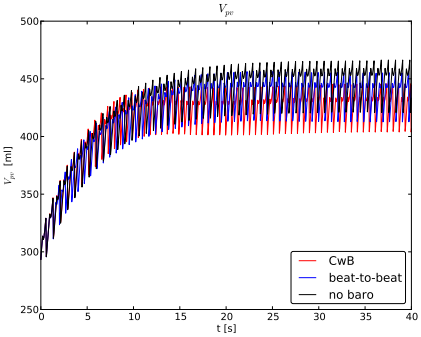
(a) Blood volume in systemic veins



(b) Blood volume in systemic arteries



(c) Volume in the distal part of the aorta



(d) Blood volume of pulmonary veins

Figure 6.13: The CwB model in python with original baroreceptors, fitted Toska baroreceptors and no baroreceptors, for aortic valve resistance, $R_{av} = 1$ mmHg s/ml

Chapter 7

Concluding remarks

The Toska baromodel gives more varied results than previously thought. Optimizing the model to recorded data will not give a conclusive result. Average results of many optimized runs, is preferred instead. A test sample of 100 runs, is too small for statistical reasons, particularly when many runs end up on the boundary and are therefore not actual local minima.

The time constants for setpoint, $T_{c,set}$ and increase in muscle flow, $T_{c,mf}$ should probably be less constrained, since many optimized runs end up on the upper boundary. Another option is to make them constant, like the time constants of the signal pathways. The two time constants $T_{c,set}$ and $T_{c,mf}$ are the parameters that change the solution the least, we found this from experimenting in the Matlab GUI. Since the system is not very sensitive to these parameters, it is hard to find minimum points for them.

The Toska baromodel can be coupled to the CwB circulation, using gains that are in range of those for the recorded subjects. This model manage to regulate the circulation system in qualitatively the same way as the CwB baroreflexes. The Toska model involves less parameters to fit, and is less complex, though equations are not based on baroreceptor nerve measurements. It might be possible to include Toska baroreflex influence on stiffness of the arterial wall given by P_{sa}, if we find a function that take the B_{sp} signal and return a value on the interval [0,1] in a similar way that the CwB efferent pathway signal F_x is a normalized version of the CNS signal N_x. Then we can use this function to scale which of P_{sa}^p and P_{sa}^a is the most dominant, instead of just using P_{sa}^p for the arterial pressure.

We mention some other opportunities for further study:

- Possible to test for other perturbations, validate the baromodels
- Connect the CwB circulation model to a more realistic 3D heart model
- Include the possibility of local metabolites when exercising in the CwB circulation

Appendix A

Python code

We include all the python code developed in this project. This is done mainly for documentation purposes. First we present the Toska model, and then the CwB model with original and Toska baroreflexes.

A.1 The Toska model in python

The Toska model consists of six files. We make a short introduction so that it should be easier to find the parts of the model mentioned in the thesis. The Systemic_Circulation_Model.py file runs the model. The InitialSimulation.py file load the parameter and experimental files and calculate the initial values. The ExecuteModel.py initialize objects of the of the classes of parts of the circulation, contains some of the model equations, and include the acceptable intervals for optimized model parameters. The objects_in_baromodel.py contain classes for each part of the circulation, and the model equations. The class TimeConstDelay for the barosignal function that we discuss in Section 4.2.4 is in this file. The Optimaliser.py file contains calls to the optimize algorithm, and the error function of the difference between the model solution and experiments. The RandomNewParam.py file includes routines for running the model for random parameters, routines for accessing optimized parameter sets and functions for statistics, plots and tables.

```
""" ----------------------------
    Systemic_Circulation_Model.py
    ----------------------------"""

from InitialSimulation import *
from Optimaliser import *

def plotter(result,expdata,subject):
```

```
    titles=['RR (s)','SV (ml/beat)','MAP (mm Hg)','MuscFlow (l/s)',
            'TPC  (l/(s*mmHg))']
    names=['rr','sv','map','mf','tpc']

    for i in xrange(len(names)):
        plot(result['time'], result[names[i]], 'r',
             expdata['exptime'], expdata['exp%s'%names[i]], 'g',
             title=titles[i], xlabel='time (s)');
        hardcopy('%s%s.eps'%(subject,names[i]))

    plot(result['time'],result['totcond'],
         result['time'], result['tpc'], 'r',
         title='Total peripheral conductance (l/(s*mmHg))',
         xlabel='time (s)',legend=['totcond','tpc'])
    hardcopy('%stotcond.eps'%subject)

def plotter2(result,expdata,subject):
    titles=['RR (s)','SV (ml/beat)','MAP (mm Hg)','MuscFlow (l/s)',
            'TPC  (l/(s*mmHg))']
    names=['rr','sv','map','mf']
    #figure(1)
    for i in xrange(len(names)):
        subplot(2,2,i+1)
        plot(result['time'], result[names[i]], 'r',
             expdata['exptime'], expdata['exp%s'%names[i]], 'g',
             title=titles[i], xlabel='time (s)');
    hardcopy('%smodel.eps'%(subject))
    """
    figure(2)
    plot(result['time'],result['totcond'],
         result['time'], result['tpc'], 'r',
         expdata['exptime'], expdata['exptpc'], 'g',
         title='Total peripheral conductance (l/(s*mmHg))',
         xlabel='time (s)',legend=['totcond','tpc','exptpc'])
    hardcopy('%stotcond.eps'%subject)
    """
def write_model_results_to_file(result,modeloutfile):
    outfile=open(modeloutfile,'w')
    outfile.write('Time\t\t  RR\t\t  SV\t\t  MAP\t\t  MF\t\t  TPC\n')
    for i in xrange(len(result['time'])):

        outfile.write('%15.9f%15.9f%15.9f%15.9f%15.9f%15.9f\n'
                      %(result['time'][i],result['rr'][i],
                        result['sv'][i],result['map'][i],
```

```
                        result['mf'][i],result['totcond'][i],))
    outfile.close()


def write_to_file_and_screen(ModParam,ModParamNames,errorNow,paramoutfile):
    outfile=open(paramoutfile,'w')

    print '\n\n%15s\t%15f'%('errorNow',errorNow)
    for i in xrange(len(ModParam)-1):
        outfile.write('%15.9f\t%15s\n'%(ModParam[i], ModParamNames[i]))
        print '%15s\t%15f'%(ModParamNames[i], ModParam[i])

    print '%15s\t%15f\n\n'%('MF0', ModParam[-1])
    outfile.write('%15.9f\t%15s\n'%(errorNow, ModParamNames[-1]))
    outfile.close()


def run_simulation(extra,time_param,FILES,MODEL):
    time,Map,rr,sv,nbeats,mf,totcond=ExecuteModel(extra['ModParam'],time_param)
    TPC=sv/(rr*Map)
    extra['result']={'time':time,'map':Map,'rr':rr,'sv':sv,'Nofbeat':nbeats,
                     'mf':mf,'tpc':TPC,'totcond':totcond}

    if MODEL['plot1']:
        plotter(extra['result'],extra['expdata'],FILES['subject'])
    if MODEL['plot2']:
        plotter2(extra['result'],extra['expdata'],FILES['subject'])

    errorNow=ErrFvalVec([],extra['ModParam'],extra['expdata'],[],time_param,
                        MODEL['mf_data'])
    extra['errorNow']=errorNow
    write_to_file_and_screen(extra['ModParam'],extra['ModParamNames']
                             ,errorNow,FILES['paramoutfile'])
    write_model_results_to_file(extra['result'],FILES['modeloutfile'])


def Systemic_Circulation_Model(FILES,newparam,inds,MODEL):
    SIMTIME     =110#129.4 #Total duration of simulation
    EXERCISETIME= 20.0     #Time of onset of exercise
    COUNTTIME   = 10.0     #Time of counting down
    SYSTDUR     =  0.15    #Half of duration of systole, must be < MINRR
    DT          =  0.2     #Sampling time of template
    time_param=[SIMTIME,EXERCISETIME,COUNTTIME,SYSTDUR,DT]
```

```
"""case 1: Initialising"""
extra=InitialSimulation(FILES,time_param,MODEL)

if MODEL['run_newparam']:
    """case 2: Put new values into Modparam vector"""
    extra['ModParam'][inds]=newparam
    run_simulation(extra,time_param,FILES,MODEL)

if MODEL['optim_newparam']:
    """reset"""
    extra['ModParam']=extra['OrigModParam']
    """case 3: Callback function for Optimaliser"""
    extra=Optimaliser(extra,newparam,inds,time_param,MODEL)
    run_simulation(extra,time_param,FILES,MODEL)

return extra #['ModParam'],extra['errorNow']
"""
SyPerGain_min       =0;      SyPerGain_max       =0.5;
SyPerTc_min         =0.1;    SyPerTc_max         =30;
SyPerDel_min        =0;      SyPerDel_max        =20;
SyRrGain_min        =0;      SyRrGain_max        =1;
SyCoGain_min        =0;      SyCoGain_max        =1;
SyRrTc_min          =0.1;    SyRrTc_max          =30;
SyRrDel_min         =0;      SyRrDel_max         =5;
ParaRrGain_min      =0;      ParaRrGain_max      =1;
ParaRrTc_min        =0.1;    ParaRrTc_max        =1;
ParaRrDel_min       =0;      ParaRrDel_max       =1;
AfterLaodSens_min   =-0.0065;AfterLaodSens_max   =0;
MitrFl_min          =0;      MitrFl_max          =1;
RestExFrac_min      =0;      RestExFrac_max      =1;
WkComp_min          =0;      WkComp_max          =1;
AdaptFac_min        =0;      AdaptFac_max        =1;
MuscFlow_min        =0;      MuscFlow_max        =1;
MuscFlTc_min        =0;      MuscFlTc_max        =10;
MuscFlDly_min       =0;      MuscFlDly_max       =10;
VenousFrac_min      =0;      VenousFrac_max      =1;
VenousDly_min       =0;      VenousDly_max       =1;
SetpStep_min        =0;      SetpStep_max        =30;
BeforeFrac_min      =0;      BeforeFrac_max      =1;
BaroTc_min          =1.1;    BaroTc_max          =30;
BaroTcCount_min     =1.1;    BaroTcCount_max     =30;
MAP0_min            =0;      MAP0_max            =120;
RR0_min             =0;      RR0_max             =3;
SV0_min             =0;      SV0_max             =1;
```

```
    MF0_min             =-1;     MF0_max             =1;
    """

if __name__=='__main__':

    newparam=[0.082,0.00030,0.0372,-0.000789,0.00112,0.026,5.9, 6.57]
    inds=[0,4,7,10,11,15,16,23]
    #newparam=[0.082,0.00030,0.0372]
    #inds=[0,4,7]
    subj='TO'
    subj='GT'
    FILES={'expfile':'Data/%sLM.MED'%subj,'paramfile':'RELC_NEW2.PTX',
           'subject':'%sLM'%subj, 'paramoutfile':'RELC_NEW3.PTX',
           'modeloutfile':'time_Map%soptim.dat'%subj}

    MODEL={'mf_data':True, 'mf_column':7,'plot1':False,'plot2':True,'mass':None,
           'run_newparam':True,'optim_newparam':True,'opt_ftol':0.1}

    extra=Systemic_Circulation_Model(FILES,newparam,inds,MODEL)
    Modparam,errorNow=extra['ModParam'],extra['errorNow']

""" ----------------------------
    InitialSimulation.py
    ----------------------------"""

from ExecuteModel import *

def InitialSimulation(FILES,time_param,MODEL):
    SIMTIME  =time_param[0]
    COUNTTIME=time_param[2]
    DT       =time_param[4]

    """read experimental data from file"""
    expdata=ReadExpFile(FILES['expfile'],SIMTIME,DT,MODEL)

    """read initial parameters from file"""
    ModParam,ModParamNames,prevError=ReadParamFile(FILES['paramfile'])

    """Averaged experimental values from before countdown"""
    ModParam[24] = Average(expdata['expmap'],1, COUNTTIME,DT)
    ModParam[25] = Average(expdata['exprr'], 1, COUNTTIME,DT)
    ModParam[26] = Average(expdata['expsv'], 1, COUNTTIME,DT)

    if MODEL['mass'] is not None:
```

```
        ModParam[13]=MODEL['mass']*0.00005   #Windkessel compliance

    if MODEL['mf_data']:
        ModParam[27] = Average(expdata['expmf'], 1, COUNTTIME,DT)
        ModParam[12] = ModParam[27]/(ModParam[26]*ModParam[25])
        if MODEL['mf_column']==8:
            expdata['expmf']=expdata['expmf']-ModParam[27]
            ModParam[27]=0
    else:
        ModParam[27]=0
        ModParam[12]=0.15

    extra ={'ModParam':ModParam,'OrigModParam':ModParam,
            'ModParamNames':ModParamNames,'expdata':expdata,'result':{},
            'experimental_file':FILES['expfile'],
            'parameter_file':FILES['paramfile'],'prevError':prevError}
    return extra

def ReadParamFile(paramfile):
    infile=open(paramfile);  lines =infile.readlines(); infile.close()
    ModParam=[]
    ModParamNames=[]
    for line in lines:
        ModParam.append(float(line.split()[0]) )
        ModParamNames.append( line.split()[1] )

    prevError= ModParam[-1]
    return array(ModParam),ModParamNames, prevError

def ReadExpFile(expfile,SIMTIME,DT,MODEL):
    time=0    #Time            (s)
    rr=1      #RR              (s)
    sv=2      #SV              (ml/beat)
    Map=3     #MAP             (mmHg)
    fv=4      #FV              (ml/beat)  (only right femoral artery)
    fv_sum=5  #FV+FV2sec-delay (ml/beat)  (ml blood to both legs per beat)
    #mf (l/s)      (liter blood per sec to both legs)
    if MODEL['mf_data']: mf=7
    else:                mf=MODEL['mf_column']

    infile=open(expfile);  lines=infile.readlines(); infile.close()
    data=zeros((len(lines),9))

    for cnt in xrange(len(lines)):
```

```
        line=array(lines[cnt].split(),float)
        data[cnt,:]=line

    data[:,sv]=data[:,sv]*0.001

    expdata={'exprr':data[:,rr],'expsv':data[:,sv],'expmap':data[:,Map],
             'exptime':data[:,time],'expfv':data[:,fv],
             'expfv_sum':data[:,fv_sum],'expmf':data[:,mf]}

    expdata['exptpc']=data[:,sv]/(data[:,rr]*data[:,Map])
    expdata['rrwh'] = 1 / Variance(data[:,rr], 1, SIMTIME,DT)
    expdata['svwh'] = 1 / Variance(data[:,sv], 1, SIMTIME,DT)
    expdata['mapwh']= 1 / Variance(data[:,Map],1, SIMTIME,DT)

    if MODEL['mf_data']:
        expdata['mfwh'] = 1 / Variance(data[:,mf], 1, SIMTIME,DT)
    else:expdata['mfwh'] = 0
    return expdata

def Variance(data,t1,t2,DT):
    avg = Average(data, t1, t2,DT)
    it1 = int(round(t1 / DT))
    it2 = int(round(t2 / DT))
    s = 0.0
    for cnt in xrange(it1-1,it2-1):
        TEMP = data[cnt] - avg
        s =s+ TEMP * TEMP
    return s/(it2 - it1)

def Average(data,t1,t2,DT):
    it1 = int(round(t1 / DT))
    it2 = int(round(t2 / DT))
    s = 0.0
    for cnt in xrange(it1-1,it2-1):
        s = s + data[cnt]
    return s/(it2 - it1)



""" ----------------------------
    ExecuteModel.py
    ----------------------------"""

from objects_in_baromodel import *
```

```
def Newparam_within_limits(x,inds):
    for k in xrange(len(x)):#after each iteration, x is newparam
        j=inds[k]
        mini,maxi= Interval(j)
        if   x[k]<mini:  x[k]=mini
        elif x[k]>maxi:  x[k]=maxi


def Param_within_limits(x):
    for k in xrange(len(x)):
        mini,maxi= Interval(k)
        if   x[k]<mini:  x[k]=mini
        elif x[k]>maxi:  x[k]=maxi


def Interval(index):
    min_param=[0,0.1,0,0,0,0.1,0,0,0.1,0,-0.0065,
               0,0,0,0,0,0.09,0,0,0,0,0,1.1,0.1,0,0,0,-1]
    max_param=[0.2,30,20,1,0.02,30,5,0.2,1,1,0,
               0.1,1,1,1,1,15,10,1,1,30,1,30,30,120,3,1,1]
    #max_param=[0.2,30,20,1,0.02,30,5,0.2,1,1,0,
    #          0.1,1,1,1,1,30,10,1,1,40,1,30,40,120,3,1,1]
    return min_param[index],max_param[index]


def ExecuteModel(ModParam,time_param):
    SIMTIME     =time_param[0]
    EXERCISETIME=time_param[1]
    COUNTTIME   =time_param[2]
    SYSTDUR     =time_param[3]
    rest    = 0
    downcnt = 1
    exerc   = 2

    Param_within_limits(ModParam)


    """Model parameters: adjusted =A"""
    symppergain   = ModParam[0]  #Sympathetic peripheral gain/sensitivity (A)
    symppertc     = ModParam[1]  #Sympathetic peripheral time constant
    sympperdel    = ModParam[2]  #Sympathetic peripheral delay
    symprrgain    = ModParam[3]  #Sympathetic rr-interval gain/sensitivity
    sympcontgain  = ModParam[4]  #Sympathetic contractivity gain/sensitivity(A)
```

```
symprrtc      = ModParam[5]  #Sympathetic rr-interval time constant
symprrdel     = ModParam[6]  #Sympathetic rr-interval delay
parasympgain  = ModParam[7]  #Parasympathetic rr-interval gain/sensitivity(A)
parasymprrtc  = ModParam[8]  #Parasympathetic rr-interval time constant
parasymprrdel = ModParam[9]  #Parasympathetic rr-interval delay

afterloadsens = ModParam[10] #Sensitivity to afterload on stroke volume (A)
mitralflow    = ModParam[11] #Mitral flow     (A)
restexfrac    = ModParam[12] #Exercising fraction of body
windcompl     = ModParam[13] #Windkessel compliance
adaptfac      = ModParam[14] #Adapt factor for mean arterial pressure error
muscflow      = ModParam[15] #Muscle flow     (A)
musctc        = ModParam[16] #Muscle flow time constant (A)
muscdelay     = ModParam[17] #Muscle flow delay
venousfrac    = ModParam[18] #Never used: venous fraction (on sv)
venousdelay   = ModParam[19] #Never used: venous delay
setpstep      = ModParam[20] #Baro set point step
beforefrac    = ModParam[21] #Baro set point before fraction
barotc        = ModParam[22] #Never used: baro set point time constant
barotccount   = ModParam[23] #Baro set point time constant countdown (A)


"""Averaged experimental values from before countdown"""
map0          = ModParam[24]  #Mean arterial pressure
rr0           = ModParam[25]  #Rr-interval
sv0           = ModParam[26]  #Stroke volume
MF0           = ModParam[27]  #Muscle flow

RunTime = 0.0                 #Running time of simulation
ExerciseState = rest          #State of exercise process

"""Initialize function for changes in baroreflex set point """
BaroSetpoint=T_BaroSetpoint(map0,setpstep,barotccount,beforefrac,COUNTTIME)
BaroAdapt   =T_BaroAdapt(adaptfac)

"""Initialize baroreflexes with appropriate delays and time constants"""
parasymp= TimeConstDelay(parasymprrdel,parasymprrtc)
symp=     TimeConstDelay(symprrdel,symprrtc)
sympperif=TimeConstDelay(sympperdel,symppertc)

"""Initialize the big arteries"""
WindKessel=T_WindKessel(windcompl,map0,sv0,rr0,SYSTDUR)
LastEdp=WindKessel.GetEdp()
LastMap = map0
```

```
    """Initialize the heart"""
    Heart=T_Heart(sv0,rr0,LastEdp,sympcontgain,symprrgain,parasympgain,
                  afterloadsens,mitralflow)

    """Total peripheral conductance (TPC)"""
    Conductance = sv0 /(rr0 * map0)

    """Devide peripheral vessels into an exercising and non-exercising part """
    nonexbody=ResistVessel(Conductance * (1.0 - restexfrac),symppergain)
    exbody   =ResistVessel(Conductance * restexfrac,symppergain)

    MuscFlowFunc=T_MuscFlowFunc(muscflow,musctc,muscdelay)
    Nofbeat = 0
    time=[];Map=[];rr=[];sv=[];mf=[];totcond=[]

    while (RunTime < SIMTIME):
        """MAP deviation from set point"""
        MapError=LastMap-BaroSetpoint.SetPoint(RunTime,ExerciseState==rest)

        MapError= BaroAdapt.Adapt(MapError)

        """Insert MAP deviation from set point into reflexes"""
        parasymp.SetMapError(RunTime,MapError)
        symp.SetMapError(RunTime,MapError)
        sympperif.SetMapError(RunTime,MapError)

        """Calculate baroreflex signals"""
        SympPerSig   =sympperif.GetSignal(RunTime)
        SympRrSig    =symp.GetSignal(RunTime)
        ParasympRrSig=parasymp.GetSignal(RunTime)

        """Stroke volume and duration of one heart beat, given baro signals"""
        NewSv,NewRR=Heart.NewBeat(LastEdp, SympRrSig, ParasympRrSig)

        if (RunTime >= COUNTTIME):
            if ExerciseState ==rest: ExerciseState = downcnt
            elif ExerciseState ==downcnt:
                if (RunTime >= EXERCISETIME):
                    ExerciseState = exerc
                    exbody.ExerciseOn()

        """Flow through exercising muscle"""
        NewMf = MuscFlowFunc.MuscFlow(RunTime,EXERCISETIME,MF0)
```

```
        """Total peripheral conductance (TPC)"""
        TotConductance=(nonexbody.Conductance(SympPerSig,NewMf,map0)+
                        exbody.Conductance(SympPerSig,NewMf,map0))
                        #map0 or LastMap


        WindKessel.Cycle(SYSTDUR, TotConductance)
        WindKessel.AddVolume(NewSv)
        WindKessel.Cycle(NewRR - SYSTDUR, TotConductance)

        """Mean arterial pressure during one heart beat"""
        LastMap=WindKessel.GetMap()
        """End diastolic pressure in the artery"""
        LastEdp=WindKessel.GetEdp()

        RunTime = RunTime+NewRR

        time.append(RunTime)
        sv.append(NewSv)
        Map.append(LastMap)
        rr.append(NewRR)
        mf.append(NewMf)
        totcond.append(TotConductance)
        Nofbeat=Nofbeat+1

    sv=array(sv);rr=array(rr);Map=array(Map)
    return time,Map,rr,sv,Nofbeat,mf,totcond

""" ----------------------------
    objects_in_baromodel.py
    ----------------------------"""

from scitools.std import *
class T_Heart:
    def __init__(self,sv0,rr0,edp0,sympcontgain,symprrgain,parasympgain,
                 afterloadsens,mitralflow):
        self.sv0 = sv0                  #initial stroke volume
        self.rr0 = rr0                  #initial rr-interval
        self.edp0 = edp0                #initial end-diastolic pressure
        self.lastrr = self.rr0
        """Only allow positive values"""
        self.sympcontgain = sympcontgain
        self.symprrgain   = symprrgain
```

```
        self.parasympgain = parasympgain
        self.mitralflow   = mitralflow
        """Only allow negative values"""
        self.aftl = afterloadsens

    def NewBeat(self,LastEdp,sympsignal,parasympsignal):
        """Normal SV + Filling time effect on SV
        + Afterload effect on SV + Sympathetic effect on contraction"""
        NewSv = (self.sv0 + (self.lastrr - self.rr0)*self.mitralflow +
                 (LastEdp - self.edp0)*self.aftl - sympsignal*self.sympcontgain)

        """Normal rr interval + Parasympathetic effect + Sympathetic effect"""
        NewRR =(self.rr0 + parasympsignal*self.parasympgain +
                sympsignal*self.symprrgain)

        MINRR = 60.0/190    #Lowest possible rr-interval duration
        if (NewRR < MINRR): NewRR = MINRR

        self.lastrr = NewRR #For next beat...
        return NewSv,NewRR

class T_WindKessel:
    """Representing large arteries"""
    def __init__(self,compliance,Map0,sv0,rr0,SYSTDUR):
        self.compliance=compliance
        self.tc    =compliance*Map0*rr0/sv0
        self.f     =exp(-(rr0/self.tc))
        self.f2    =exp(SYSTDUR/self.tc)
        """Calculate exact initial end-diastolic volume"""
        self.volume=sv0*self.f2*self.f/(1-self.f)

        self.pint=0
        self.tsum=0

    def Cycle(self,rr,TPC):
        self.tc=self.compliance/TPC #calculate time constant of volume decay
        exf=exp(-(rr/self.tc))      #compute exponential function coefficient
        newmap=self.volume*(1-exf)*self.tc/(rr*self.compliance)
        #newmap=self.volume*(1-exf)/(rr*TPC)

        self.volume=self.volume*exf #compute volume at end of cycle
        """Compute mean arterial pressure by integration"""
        self.pint=self.pint+newmap*rr
        self.tsum=self.tsum+rr
```

```
    def AddVolume(self,NewSv):
        self.volume=self.volume+NewSv

    def GetMap(self):
        Map=self.pint/self.tsum
        self.pint=0
        self.tsum=0
        return Map

    def GetEdp(self):
        """Calculate end-diastolic pressure"""
        return self.volume/self.compliance

class ResistVessel:
    """Representing peripheral vessels"""
    def __init__(self,G0,symppergain):
        """G0:  Total peripheral conductance before exercise"""
        self.G0 = G0
        self.exercise = False

        """Do not allow a negative value of this reflex gain;
        innervation/sympathetic vasoconstriction sensitivity """
        self.symppergain = symppergain

    def Conductance(self,innervate,muscflow,mapsetp):

        if (self.exercise):
            """Local metabolites of exercising muscles override
            sympathetic nervous input"""
            return  self.G0 + (muscflow/mapsetp)
        else:
            """Sympathetic vasoconstriction reduces conductance"""
            return  self.G0 * (1.0 + innervate*self.symppergain)

    def ExerciseOn(self):
        self.exercise = True


class T_MuscFlowFunc:
    def __init__(self,muscflow,tc,delay):
        self.tc = tc
        self.muscflow = muscflow
        self.delay = delay
```

```
    def MuscFlow(self, Runtime, ExerciseTime, MF0):
        """Muscle flow start increasing after the onset of exercise
        up to a maximum value"""
        if (Runtime < ExerciseTime + self.delay):
            return  MF0                             #Original value:0.0
        else:
            return (MF0 + self.muscflow *
                    (1.0-exp((ExerciseTime +self.delay -Runtime)/self.tc)))

class T_BaroSetpoint:
    def __init__(self,mapsetp,setpstep,barotccount,beforefrac,COUNTTIME):
        self.COUNTTIME=COUNTTIME
        self.mapsetp = mapsetp
        self.setpstep = setpstep
        self.barotccount = barotccount
        self.beforefrac = beforefrac

    def SetPoint(self,Runtime,Before_COUNTTIME):
        """The baroreflex set point starts increasing when the countdown
        begins, increases up to a maximum value """
        if (Before_COUNTTIME):
            return  self.mapsetp
        else:
            return (self.mapsetp +  self.beforefrac*self.setpstep *
                (1- exp((self.COUNTTIME - Runtime)/self.barotccount)))


class T_BaroAdapt:
    def __init__(self,adaptfac):
        self.adaptfac  = adaptfac
        self.LastMapError= 0

    def Adapt(self,MapError):
        AdaptError = MapError + self.adaptfac * (MapError - self.LastMapError)
        self.LastMapError = MapError
        return AdaptError

class TimeConstDelay:
    def __init__(self,delay,timeconst):
        self.Delay={}           #Delay FIFO queue
        self.wrp = 0            #Set write pointer to zero
        self.rep = 0            #Set read pointer to zero
```

```
        self.signal= 0.0        #Time constant filter cell set to zero
        self.lastt = 0.0        #Set last time value to zero

        """Do not allow negative delays, remember which delay to use"""
        self.delay = delay

        """Do not allow too small a time constant in reflexes,
        this may cause floating point overflow"""
        self.tc = timeconst

    def CalcNewExp(self,t,MapError):
        dt = t - self.lastt
        if (dt <= 0):f = 0.0
        else:        f = exp(-dt / self.tc)

        self.signal = self.signal * f + MapError*(1.0 - f)
        self.lastt = t

    def GetSignal(self,RunTime):
        td = RunTime - self.delay   #Compute delayed time
        before=True
        while (before):
            tm,MapError = self.Delay[self.rep]
            before=(tm < td)
            if (before):
                self.CalcNewExp(tm,  MapError)
                self.rep=self.rep+1 #Increment read pointer
            else:
                self.CalcNewExp(td, MapError)
        return self.signal

    def SetMapError(self,RunTime,MapError):
        """Insert value at write pointer position"""
        self.Delay[self.wrp]=[RunTime,MapError]
        self.wrp=self.wrp+1    #Increment write pointer




""" ----------------------------
    Optimaliser.py
    ----------------------------"""

import scipy.optimize
```

```
from ExecuteModel import *

def Optimaliser(extra,newparam,inds,time_param,MODEL):
    """create local copies of some variables"""
    expdata=extra['expdata']
    ModParam=extra['ModParam']

    """set optimization options and do optimization:
    Find minimum of unconstrained multivariable function
    using derivative-free method"""
    newparam,errorNow,itera,funcalls,flag= \
        scipy.optimize.fmin(ErrFvalVec,newparam,
                            args=(ModParam,expdata,inds,time_param,
                                  MODEL['mf_data']),
                            xtol=10,ftol=MODEL['opt_ftol'],full_output=1,
                            callback=
                            eval('Newparam_within_limits(newparam,inds)'))

    """put new value into Modparam vector"""
    ModParam[inds]=newparam
    extra['ModParam'] = ModParam #contain mean values
    return extra


def ErrFvalVec(newparam,ModParam,expdata,inds,time_param,musc_flow_data):
    ModParam[inds] = newparam
    time,Map,rr,sv,nbeats,mf,totcond=ExecuteModel(ModParam,time_param)

    Err_map= ErrSum(time, Map,expdata['expmap'],time_param)
    Err_sv = ErrSum(time, sv, expdata['expsv'], time_param)
    Err_rr = ErrSum(time, rr, expdata['exprr'], time_param)
    """To obtain dimensionless numbers, devide by respective variances"""
    Err=(expdata['mapwh'] * Err_map + expdata['svwh']*Err_sv +
         expdata['rrwh'] * Err_rr)

    if musc_flow_data:
        Err_mf = ErrSum(time, mf, expdata['expmf'], time_param)
        Err=Err+expdata['mfwh'] * Err_mf
    return Err


def ErrSum(time,simvals,expvals,time_param):
    SIMTIME=time_param[0]
    DT     =time_param[4]
```

```
    t1     =time_param[2] #COUNTTIME
    t2     =SIMTIME  -60
    it1 = int(round(t1/DT))
    it2 = int(round(t2/DT))

    EXP_SAMP=int(round(SIMTIME/DT))#Samples in averaged experimental result
    intp=zeros(EXP_SAMP)

    for cnt in xrange(len(time)):
        if (time[cnt] > t1 and time[cnt-1] < t2):
            AddInt(simvals[cnt], time[cnt-1], time[cnt],intp,DT)
    errs=0

    for cnt in xrange(it1,it2):             #(it1-1,it2+3):
        TEMP = intp[cnt] - expvals[cnt]     #Least squares
        errs = errs + TEMP * TEMP
    return errs

def AddInt(v,t1,t2,intp,DT):
    s1 = t1 / DT;   is1 = int(s1);    s1 = s1-is1
    s2 = t2 / DT;   is2 = int(s2);    s2 = s2-is2
    for p in xrange(is1,is2+1):
        intp[p] = intp[p] + v
    intp[is1] = intp[is1] - v * s1
    intp[is2] = intp[is2] + v * (s2 - 1.0)


"""-------------------------
   RandomNewParam.py
   ----------------------"""
from Systemic_Circulation_Model import *
import random

def random_new_param(inds):
    newparam=[]
    for i in xrange(len(inds)):
        mini,maxi= Interval_start_values(inds[i])
        newparam.append(random.uniform(mini,maxi))
    return newparam

def Interval_start_values(index):
    min_param=[0,0.1,0,0,0,0.1,0,0.02,0.1,0,-0.0057,
               0,0,0,0,0,0.1,0,0,0,5,0,1.1,1.1,0,0,0,-1]
    max_param=[0.1,30,20,1,0.01,30,5,0.1,1,1,0,
```

```
            0.1,1,1,1,0.18,10,10,1,1,10,1,30,15,120,3,1,1]
    return min_param[index],max_param[index]

def param_on_interval_edge(x,inds,zero_edge):
    edge=False
    for k in xrange(len(x)):
        mini,maxi= Interval(inds[k])
        if zero_edge:
            if   (x[k]==mini or x[k]==maxi): edge=True
        else:
            if   (x[k]==mini or x[k]==maxi) and not x[k]==0: edge=True
    return edge


def write_interior_file(outfilename,Interior_param,mn,statistics):
    kl=Interior_param.shape
    fraction_interior=kl[0]/float(mn[0])

    outfile=open(outfilename,'w')
    write_to_file(Interior_param,outfile)
    outfile.write('\n%15.9f\n'%fraction_interior)
    write_to_file(statistics,outfile)
    outfile.close()

def write_to_file(x,outfile):
    for i in xrange(x.shape[0]):
        for j in xrange(x.shape[1]):
            outfile.write('%15.9f'%x[i,j])
        outfile.write('\n')

def write_sorted_file(Sorted_Adjusted,outfilename):
    outfile=open(outfilename,'w')
    write_to_file(Sorted_Adjusted,outfile)
    outfile.close()

def read_optimized_param_file(name):
    infile=open(name,'r');lines=infile.readlines(); infile.close()
    Adjusted_param=[]

    for line in lines:
        line=array(line.split(),float)
        Adjusted_param.append(line)

    Adjusted_param.sort(key=lambda adjust:adjust[0])
```

```
    return array(Adjusted_param)

def experiment_subjects(year,subjects):
    """Subjects id and mass"""
    if year=='1996':
        experiments={'ELLA':64,'ELLC':64,'HGLA':73,'HGLC':73,'HVLA':56,
                     'HVLC':56,'KYLA':65,'KYLC':65,'LJLA':73,'LJLC':73,
                     'LMLA':61,'LMLC':61,'MKLA':62,'MKLC':62,'PLLA':83,
                     'PLLC':83,'RJLA':75,'RJLC':75,'TKLA':57,'TKLC':57}
        datadirname='DATA96/'
    elif year=='1996A':
        experiments={'ELLA':64,'HGLA':73,'HVLA':56,'KYLA':65,'LJLA':73,
                     'LMLA':61,'MKLA':62,'PLLA':83,'RJLA':75,'TKLA':57}
        datadirname='DATA96/'
    elif year=='1996C':
        experiments={'ELLC':64,'HGLC':73,'HVLC':56,'KYLC':65,'LJLC':73,
                     'LMLC':61,'MKLC':62,'PLLC':83,'RJLC':75,'TKLC':57}
        datadirname='DATA96/'
    else:
        experiments={'BTLM':55,'CTLM':63,'ETLM':82,'FTLM':90,'GTLM':72,
                     'HTLM':55,'ITLM':68,'JTLM':50,'LTLM':54,'MTLM':63,
                     'TOLM':65.2}
        datadirname='Data/'

    expname=experiments.keys()
    expname.sort()
    if subjects is not 'all':
        expname=subjects

    return experiments,datadirname,expname

def make_statistics(Interior_param,inds):
    statistics=zeros((6,len(inds)+1))
    for l in xrange(len(inds)+1):
        statistics[0,l]=Interior_param[:,l].mean()
        statistics[1,l]=Interior_param[:,l].var()
        statistics[2,l]=Interior_param[:,l].std()
        statistics[3,l]=Interior_param[:,l].min()
        statistics[4,l]=Interior_param[:,l].max()
        statistics[5,l]=median(Interior_param[:,l])
    return statistics

def find_interior_parameters(Adjusted,mn,zero_edge):
    Interior_param=[]
```

```
    for m in xrange(mn[0]):
        newparam=Adjusted[m][1:]
        edge=param_on_interval_edge(newparam,inds,zero_edge)
        if ((not edge) and Adjusted[m][0]<700):
            Interior_param.append(Adjusted[m,:])

    return array(Interior_param)

def find_mean_optimized_param(Interior_param,epsilon):
    """Find all result that have ErrorNow < min(ErrorNow) + epsilon"""
    kl=Interior_param.shape
    Final_param=[]
    for k in xrange(kl[0]):
        if Interior_param[k][0]>Interior_param[0][0]+epsilon: break
        Final_param.append(Interior_param[k,:])

    Final_param=array(Final_param)
    mean_optimized_param=zeros(len(inds)+1)
    for l in xrange(kl[1]):
        mean_optimized_param[l]=Final_param[:,l].mean()
    return mean_optimized_param


def make_outfilenames(name,ident,zero_edge,artname,modelname,dirname,
                      dirnameout):
    filename='%s%s.dat'%(name,ident)
    artfilename='%s%s.dat'%(artname,ident)
    modfilename='%s%s%s.dat'%(dirname,modelname,ident)
    outfilename1='%s%s'%(dirname,filename)
    outfilename4='%s%s'%(dirnameout,filename)
    namerand='%srand%s'%(dirname,filename)
    filename2='sorted%s'%filename
    outfilename2='%s%s'%(dirname,filename2)
    if zero_edge:
        outfilename3='%sinterior%s'%(dirname,filename2)
    else:
        outfilename3='%sinterior_w0%s'%(dirname,filename2)
    return (outfilename1,outfilename2,outfilename3,namerand,artfilename,
            modfilename,outfilename4)

def View_results(Sorted_Adjusted,show,FILES,inds,MODEL):
    mn=Sorted_Adjusted.shape
    MODEL['optim_newparam']=False
    MODEL['run_newparam']=True
```

```
    for m in xrange(mn[0]):
        newparam=Sorted_Adjusted[m][1:]
        if m==show: break

        extra=Systemic_Circulation_Model(FILES,newparam,inds,
                                         MODEL)
    return extra

def sort_param(inds3,newparam1,newparam2):
    newparam1=list(newparam1)
    newparam2=list(newparam2)
    newparam=array(newparam1+newparam2)
    indssort=array([inds3,newparam])
    indssort=list(indssort.transpose())
    indssort.sort(key=lambda a:a[0])
    indssort=array(indssort)
    return indssort[:,0], indssort[:,1]


def Old_results_and_added_param_optim_all(Sorted_Adjusted,FILES,inds1,inds2,
                                      MODEL,outfilename4):
    mn=Sorted_Adjusted.shape
    MODEL['optim_newparam']=True
    MODEL['run_newparam']=False
    outfile=open(outfilename4,'a')
    inds3=array(inds1+inds2)

    for i in xrange(mn[0]):
        newparam1=list(Sorted_Adjusted[i][1:])
        newparam2=random_new_param(inds2)  #remember sorting, outfiles
        inds,newparam=sort_param(inds3,newparam1,newparam2)

        for j in xrange(4):
            extra=Systemic_Circulation_Model(FILES,newparam,inds,
                                                    MODEL)
            Modparam,errorNow=extra['ModParam'],extra['errorNow']
            newparam=Modparam[inds]

        outfile.write('%20.9f'%errorNow)
        for k in xrange(len(inds)):
            outfile.write('%15.9f'%Modparam[inds[k]])
        outfile.write('\n')
```

```
    outfile.close()

def Old_results_optim_added_param(Sorted_Adjusted,FILES,inds1,inds2,MODEL,
                                  outfilename4):
    mn=Sorted_Adjusted.shape
    MODEL['optim_newparam']=True
    MODEL['run_newparam']=False
    FILES['paramfile']= FILES['paramoutfile']
    outfile=open(outfilename4,'a')
    inds3=array(inds1+inds2)

    for i in xrange(mn[0]):
        newparam1=list(Sorted_Adjusted[i][1:])
        newparam2=random_new_param(inds2)  #remember sorting, outfiles

        ModParam,ModParamNames,prevError=ReadParamFile(FILES['paramfile'])
        ModParam[inds1]=newparam1
        write_to_file_and_screen(ModParam,ModParamNames,prevError,
                                 FILES['paramoutfile'])
        for j in xrange(4):
            extra=Systemic_Circulation_Model(FILES,newparam2,inds2,
                                                          MODEL)
            Modparam,errorNow=extra['ModParam'],extra['errorNow']
            newparam2=Modparam[inds2]

        inds,newparam=sort_param(inds3,newparam1,newparam2)
        outfile.write('%20.9f'%errorNow)
        for k in xrange(len(inds)):
            outfile.write('%15.9f'%Modparam[inds[k]])
        outfile.write('\n')

    outfile.close()


def Run_simulation(outfilename1,namerand,N,FILES,inds,MODEL):
    outfile=open(outfilename1,'a')
    outfilerand=open(namerand,'a')

    for i in xrange(N):
        newparam=random_new_param(inds)
        for k in xrange(len(inds)):
            outfilerand.write('%15.9f'%newparam[k])
        outfilerand.write('\n')
```

```
        for j in xrange(4):
            extra=Systemic_Circulation_Model(FILES,newparam,inds,
                                                    MODEL)
            Modparam,errorNow=extra['ModParam'],extra['errorNow']
            newparam=Modparam[inds]

        outfile.write('%20.9f'%errorNow)
        for k in xrange(len(inds)):
            outfile.write('%15.9f'%Modparam[inds[k]])
        outfile.write('\n')

    outfile.close()
    outfilerand.close()

def change_units(newparam,inds):
    """Change units for presentation in table"""
    change={0:False,4:True,7:True,10:True,11:True,15:False,16:False,
            20:False,23:False}

    units=zeros(len(inds))
    for i in xrange(len(inds)):
        if change[inds[i]]:
            units[i]=newparam[i]*1000
        else:
            units[i]=newparam[i]

    return units

def make_line_in_table(param,inds,shuffle,table_error):
    newparam=param[1:]
    newparam_units=change_units(newparam,inds)
    """Present data in same order as in article Elstad et al. 2002"""
    new=newparam_units[shuffle]

    line=''
    for i in xrange(len(new)):
        line+='& %7.3f'%new[i]

    if table_error:
        line+='& %7.0f'%param[0]
    line=r"""%s\\
         """%line
    return line
```

```
def make_headline(inds,shuffle,name,table_error):
    header={0:['$K_{sp}$','','(l','/mmHg)'],
            4:['$K_{sh}$','','(ml','/mmHg)'],
            7:['$K_{ph}$','','(ms','/mmHg)'],
            10:['$K_{a}$','','(ml','/mmHg)'],
            11:['$Q_{m}$','','(ml/s)',''],
            15:['$Q_{mf}$','','(l/s)',''],
            16:['$T_{c,mf}$','','(s)',''],
            20:['Setpoint','Increase','(mmHg)',''],
            23:['$T_{c,set}$','','(s)',''],
            -1:['Error','','','']}

    inds_shuffled=array(inds)
    inds_shuffled=inds_shuffled[shuffle]

    if table_error:
       inds_shuffled=list(inds_shuffled)
       inds_shuffled.append(-1)


    line1='{|l|l|'
    line2='\n%s&Weight'%(name[0:2])
    line3='\n&'
    line4='\n&(kg)'
    line5='\n&'
    line6='\n&'

    for i in xrange(len(inds_shuffled)):
        line1+='l|'
        h=header[inds_shuffled[i]]
        line2+='&%s'%h[0]
        line3+='&%s'%h[1]
        line4+='&%s'%h[2]
        line5+='&%s'%h[3]
        line6+='&'

    line=r"""%s}%s\\%s\\%s\\%s\\%s\\
         """%(line1,line2,line3,line4,line5,line6)
    return line

def write_data_line(outfile,experiments,name,num_data,inds,shuffle,
                    letter,table_error):
    line=r"""%s & %d """
    outfile.write(line%(letter,experiments[name]))
```

```
    data=make_line_in_table(num_data,inds,shuffle,table_error)
    outfile.write(data)

def make_table(data,datakeys,experiments,name,dirname,inds,shuffle,
               table_error):
    filename='%s%s.tex'%(dirname,name)
    outfile=open(filename,'w')

    headline=make_headline(inds,shuffle,name,table_error)

    begin_table=r"""
    \begin{table}
    \begin{tiny}
    \begin{tabular}%s"""%headline

    caption=''

    line=r"""%s & %d """
    outfile.write(begin_table)
    letters=['a','b','c','d','e']
    for i in xrange(len(datakeys)):
        write_data_line(outfile,experiments,name,data[datakeys[i]][1],
                        inds,shuffle,letters[i],table_error)
        caption+='%s) %s'%(letters[i],data[datakeys[i]][3])

    end_table=r"""
    \end{tabular}
    \caption{%s}
    \end{tiny}
    \end{table}
    """%caption

    outfile.write(end_table)
    outfile.close()


def plot_differences(data,datakeys,name,dirname):
    titles=['RR (s)','SV (ml/beat)','MAP (mm Hg)','MuscFlow (l/s)',
            'TPC  (l/(s*mmHg))']
    names=['rr','sv','map','mf']
    expdata=data[0][0]['expdata']

    for i in xrange(len(names)):
        #subplot(2,2,i+1)
```

```
        leg=[]
        plot(expdata['exptime'], expdata['exp%s'%names[i]], 'g',
             title=titles[i], xlabel='Time (s)')
        leg.append('Recorded data')
        hold('on')
        for j in xrange(len(datakeys)):
            result=data[datakeys[j]][0]['result']
            plot(result['time'], result[names[i]],data[datakeys[j]][2] )
            leg.append('%s'%(data[datakeys[j]][4]))
        legend(leg)
        hold('off')
        hardcopy('%s%sall%s.eps'%(dirname,name,names[i]))

    #hardcopy('%s%sall.eps'%(dirname,name))

def prepare_plot(param,FILES,inds,MODEL):
    newparam=param[1:]
    extra=Systemic_Circulation_Model(FILES,newparam,inds,MODEL)
    param[0]=extra['errorNow']
    return extra


if __name__=='__main__':
    year='1996C'
    subjects='all'
    subjects=['TKLC']
    experiments,datadirname,expname=experiment_subjects(year,subjects)

    #dirname='TIRftol01_mass_notMF0_SPS/'
    dirnameout='TIRftol01_1996_mass_8thenSPS/'
    dirname='TwoIntervalResult_ftol01_1996mass_SPS/'

    name='Adjust_param_4x100'
    artname='ArticleResultsSPS/param'
    modelname='time_Map'

    FILES={'expfile':'','paramfile':'RELC_NEW2.PTX','subject':'',
           'paramoutfile':'RELC_NEW3.PTX','modeloutfile':''}

    MODEL={'mf_data':True,'mf_column':8,'plot1':False,'plot2':False,'mass':None,
           'run_newparam':True,'optim_newparam':False,'opt_ftol':0.1}

    inds=[0,4,7,10,11,15,16,20,23]
    inds2=[20]
```

```
shuffle=[7,8,2,0,1,5,6,4,3]
epsilon=3
show=1
N=100
zero_edge=False
mean_optimized=zeros((len(experiments),len(inds)+1))
table_error=True


for r in xrange(len(expname)):
    outfilename=make_outfilenames(name,expname[r],zero_edge,artname,
                                  modelname,dirname,dirnameout)

    FILES['expfile']='%s%s.MED'%(datadirname,expname[r])
    FILES['subject']='%s%s'%(dirname,expname[r])
    FILES['modeloutfile']=outfilename[5]
    #MODEL['mass']=None  #None gives compliance from parameter file
    MODEL['mass']=experiments[expname[r]]

    #Run_simulation(outfilename[0],outfilename[3],N,FILES,inds,MODEL)

    #Sort results
    Sorted_Adjusted=read_optimized_param_file(outfilename[0])
    write_sorted_file(Sorted_Adjusted,outfilename[1])

    #Old_results_optim_added_param(Sorted_Adjusted,FILES,inds,inds2,MODEL,
                                  #outfilename[6])

    mn=Sorted_Adjusted.shape
    extra_sa=View_results(Sorted_Adjusted,show,FILES,inds,MODEL)
    #Accepted parameter sets
    Interior_param=find_interior_parameters(Sorted_Adjusted,mn,zero_edge)

    #Statistcis: mean, var, std, fraction of accepted etc
    statistics=make_statistics(Interior_param,inds)
    write_interior_file(outfilename[2],Interior_param,mn,statistics)

    extra_ip=prepare_plot(Interior_param[0],FILES,inds,MODEL)
    #Mean of accepted parameter sets
    extra_st=prepare_plot(statistics[0],FILES,inds,MODEL)

    #Mean of fraction of accepted parameter sets
    mean_optimized[r,:]=find_mean_optimized_param(Interior_param,epsilon)
    extra_mo=prepare_plot(mean_optimized[r],FILES,inds,MODEL)
```

```
#Published values: 1996
#MODEL['mass']=experiments[expname[r]]

article=read_optimized_param_file(outfilename[4])
extra_a=prepare_plot(article[0],FILES,inds,MODEL)

data={}
data[0]=[extra_sa,Sorted_Adjusted[0][:],'b',
         'Best optimation reached. ','Best optim']
data[1]=[extra_ip,Interior_param[0][:], 'c',
         'Best accepted optimation reached.','Best accepted']
data[2]=[extra_st,statistics[0][:],     'r',
         'Mean of accepted parameter sets.','Mean accepted']
data[3]=[extra_mo,mean_optimized[r][:], 'k',
         'Mean of fraction of accepted parameters.','Mean of fraction']
data[4]=[extra_a, article[0][:],       'y',
         'Published values.','Published values.']

#Make tables and plots for report
datakeys=[1,2,3,4]
make_table(data,datakeys,experiments,expname[r],dirname,inds,
           shuffle,table_error)

datakeys=[0,2,3,4]
plot_differences(data,datakeys,expname[r],dirname)
```

A.2 The CwB model in python

For the occasion of including the program in this document we decided to join the CwB model with original and Toska baroreflexes in one program. This program is called JSim_heart_complete.py. In all the runs shown and timed, they where in separate programs for efficiency.

One way to connect them would be to include if-test everywhere were the baroreflexes are involved. There are two reasons why this is not the best idea. The most important is that the original program has an additional six state variables, since the baroreflexes are modeled as differential equations. We want to avoid them in the version with Toska baroreflexes. The other reason is that if-tests slow down the code, and it is already slow.

Since the programs are basically built round the same vessel components, all the parameters are equal. To avoid them being printed twice, we make a string that include all parameters. A string is immutable so it can be put in-

side a tuple, and accepted by the args parameter in the scipy.odeint function. Then at every time step the python function exec() is called on the string. This makes the parameters local inside the solve_system() function. The program slows down approximately by a factor three, compared to declaring the parameters directly inside the solve_system() function.

At the end of the original CwB-program there is a drawing, which use symbols from electrical circuits to represent different properties of the vessels in the circulation. This is appropriate since the mathematical representation of flow through these vessels are analogous of the equations of current through the respective circuit components. The exception is the compliance of the big arteries and veins already mentioned, but we still use capacitors to visualize them. The drawing of the CwB-circulation is Figure A.1.

```
"""
JSim_heart_complete.py

Modeled using: from scipy.integrate import odeint
Changed initial conditions to model input in Jsim
"""

from scitools.std import *
from scipy.integrate import odeint
from objects_in_baromodel import *
import sys,time

def print_to_outfile(t,z,outfilename):
    ofile=open(outfilename,'w')

    for i in xrange(len(t)):
        ofile.write('%12g  '%(t[i]))
    ofile.write('\n')

    for j in xrange(z.shape[1]):
        for i in xrange(z.shape[0]):
            ofile.write('%12g  '%(z[i,j]))
        ofile.write('\n')
    ofile.close()

class Heartrate:
    def __init__(self):
        self.HR       = 77.0/60
        self.afs_con2 = 1 #afs_con
        self.t_shift  = t_min
        #Scaler to set ventricular systolic fraction of heart cycle (sec)
```

```
        self.TsvK     = 0.35
        #Scaler to set atrial systolic fraction of heart cycle (sec)
        self.TsaK     = 0.2
        self.Tsv      = self.TsvK*sqrt(1.0/self.HR)
        self.Tsa      = self.TsaK*sqrt(1.0/self.HR)
        self.shift    = False
        self.baro_on  =[True,True,True,True]
        self.Nbr      =[]
        self.t_shift_Nbr =t_min


def InitialSimulation(paramfile):
    """read initial parameters from file"""
    infile=open(paramfile);  lines =infile.readlines(); infile.close()
    ModParam=[]
    ModParamNames=[]
    for line in lines:
        ModParam.append(float(line.split()[0]) )
        ModParamNames.append( line.split()[1] )

    ModParam =array(ModParam)
    ModParam[27] = ModParam[12]*(ModParam[26]*ModParam[25])
    extra ={'ModParam':ModParam,'ModParamNames':ModParamNames,'result':{}}
    return extra

class Baroreflex:
    def __init__(self,ModParam):
        self.COUNTTIME=10
        """Model parameters:"""
        self.symppergain = ModParam[0]#Sympathetic peripheral gain/sensitivity
        symppertc        = ModParam[1]#Sympathetic peripheral time constant
        sympperdel       = ModParam[2]#Sympathetic peripheral delay
        self.symprrgain  = ModParam[3]#Sympathetic rr-interval gain/sensitivity
        self.sympcontgain= ModParam[4]#Sympathetic contractivity gain/sensitivity
        symprrtc         = ModParam[5]#Sympathetic rr-interval time constant
        symprrdel        = ModParam[6]#Sympathetic rr-interval delay
        self.parasympgain= ModParam[7]#Parasympathetic rr-interval gain/sensitivity
        parasymprrtc     = ModParam[8]#Parasympathetic rr-interval time constant
        parasymprrdel    = ModParam[9]#Parasympathetic rr-interval delay
        setpstep         = ModParam[20]#Baro set point step
        beforefrac       = ModParam[21]#Baro set point before fraction
        barotccount      = ModParam[23]#Baro set point time constant countdown
        self.map0        = ModParam[24]#Mean arterial pressure
        self.rr0         = ModParam[25]
```

```
        self.HR          = 1/ModParam[25]#77.0/60
        self.ExerciseState=0
        """Initialize function for changes in baroreflex set point """
        self.BaroSetpoint=T_BaroSetpoint(self.map0,setpstep,barotccount,
                                         beforefrac,self.COUNTTIME)

        """Initialize baroreflexes with appropriate delays and time constants"""
        self.parasymp= TimeConstDelay(parasymprrdel,parasymprrtc)
        self.symp=     TimeConstDelay(symprrdel,symprrtc)
        self.sympperif=TimeConstDelay(sympperdel,symppertc)

        self.SympPerSig=0
        self.SympRrSig =0
        self.ParasympRrSig=0
        self.t_shift   = t_min
        self.t_shift_dt= t_min
        self.baro_on=[True,True,True,True]
        #Scaler to set ventricular systolic fraction of heart cycle (sec)
        self.TsvK     = 0.35
        #Scaler to set atrial systolic fraction of heart cycle (sec)
        self.TsaK     = 0.2
        self.Tsv      = self.GetTs(self.TsvK)
        self.Tsa      = self.GetTs(self.TsaK)
        self.pint=0
        self.tsum=0

    def SumMap(self,Psa,t_delta):
        self.pint=self.pint+Psa*t_delta
        self.tsum=self.tsum+t_delta
        self.t_shift_dt+=t_delta

    def GetMap(self):
        Map=self.pint/self.tsum
        self.pint=0
        self.tsum=0
        return Map

    def NewRR(self):
        """Normal rr interval + Parasympathetic effect + Sympathetic effect"""
        NewRR =(self.rr0 + self.ParasympRrSig*self.parasympgain +
                self.SympRrSig*self.symprrgain)

        MINRR = 60.0/190    #Lowest possible rr-interval duration
        if (NewRR < MINRR): NewRR = MINRR
```

```
        self.HR=1/NewRR

    def SympatheticContractivity(self):
        return self.sympcontgain*self.SympRrSig*self.HR

    def SympatheticPeripheral(self):
        return 1.0/(1+self.symppergain*self.SympPerSig)

    def NewBeat(self,t):
        self.t_shift = t
        Map=self.GetMap()
        """MAP deviation from set point"""
        MapError=Map-self.BaroSetpoint.SetPoint(t,self.ExerciseState==0)
        """Insert MAP deviation from set point into reflexes"""
        self.parasymp.SetMapError(t,MapError)
        self.symp.SetMapError(t,MapError)
        self.sympperif.SetMapError(t,MapError)

        """Calculate baroreflex signals"""
        self.SympPerSig   =self.sympperif.GetSignal(t)
        self.SympRrSig    =self.symp.GetSignal(t)
        self.ParasympRrSig=self.parasymp.GetSignal(t)

        if self.baro_on[0] and self.baro_on[1]:
            self.NewRR()

        self.Tsv      = self.GetTs(self.TsvK)
        self.Tsa      = self.GetTs(self.TsaK)

    def GetTs(self,TsK):
        return TsK*sqrt(1.0/self.HR)

    def Activation(self,trel,Ts):
        if (0.0<=trel and trel<Ts):
            y=(1.0 - cos(pi*trel/Ts))/2.0
        elif (Ts<=trel and trel<1.5*Ts):
            y=(1.0 + cos(2.0*pi*(trel-Ts)/Ts))/2.0
        else:
            y=0.0
        return y

    def Activation_function(self,t,PRint):
        trela = t-self.t_shift
        trelv = t-self.t_shift-PRint
```

```
        ya=self.Activation(trela,self.Tsa)
        yv=self.Activation(trelv,self.Tsv)
        return ya,yv



parameters=(r"""
#---------------------------------------------------------------------
#  I. PARAMETERS  OF  VARYING  ELASTANCE  HEART  MODEL
#---------------------------------------------------------------------

Vlvrd= 72.0           #Unstressed end-diastolic left ventricle volume (ml)
Vlvrs= 23.0           #Unstressed end-systolic left ventricle volume  (ml)
Vrvrd= 103.0          #Unstressed end-diastolic right ventricle volume (ml)
Vrvrs= 53.0           #Unstressed end-systolic right ventricle volume (ml)
Vlard= 10.0           #Unstressed end-diastolic left atrium volume (ml)
Vlars= 8.0            #Unstressed end-systolic left atrium volume (ml)
Vrard= 10.0           #Unstressed end-diastolic right atrium volume (ml)
Vrars= 8.0            #Unstressed end-systolic right atrium volume (ml)

Rra = 0.001           #Tricuspid valve resistance (mmHg*s/ml)
Rla = 0.001           #Mitral valve resistance (mmHg*s/ml)

PRint = 0.12         #Difference in atrial, venticular activation times(sec)

Emaxlv= 5.6           #Maximum elastance left ventricle  (mmHg/ml)
Eminlv= 0.186874659   #Minimum elastance left ventricle  (mmHg/ml)
Emaxrv= 0.67          #Maximum elastance right ventricle  (mmHg/ml)
Eminrv= 0.1041640922  #Minimum elastance right ventricle  (mmHg/ml)

Emaxra= 0.1091675077  #Maximum elastance right atrium (mmHg/ml)
Eminra= 0.0992431888  #Minimum elastance right atrium (mmHg/ml)
Emaxla= 0.1446191772  #Maximum elastance left atrium (mmHg/ml)
Eminla= 0.1314719793  #Minimum elastance left atrium (mmHg/ml)

Pbs  = 0              #Reference pressure at body surface (ground) (mmHg)
Vmyo = 238.0          #Volume of myocardium (ml)
#---------------------------------------------------------------------
#  II. PARAMETERS OF  SYSTEMIC  CIRCULATION
#---------------------------------------------------------------------

#Resistances:
```

```
Rav   = 0.0001        #Aortic valve resistance  (mmHg*s/ml)
Raop  = 0.0001        #Proximal aortic resistance  (mmHg*s/ml)
Rcrb  = 6.8284472205  #Cerebral circulation resistance  (mmHg*s/ml)
Raod  = 0.025         #Distal aortic resistance  (mmHg*s/ml)
Rtaop = 0.2           #Transmural proximal aortic resistance  (mmHg*s/ml)
Rtaod = 0.3           #Transmural distal aortic resistance  (mmHg*s/ml)
Rsap  = 0.025         #Prox. systemic arteries resistance  (mmHg*s/ml)
Rsc   = 0.1545054945  #Systemic capillaries resistance  (mmHg*s/ml)
Rsv   = 0.1381298227  #Systemic veins resistance  (mmHg*s/ml)

#Compliances:
Caop = 0.3445734208   #Aortic proximal compliance  (ml/mmHg)
Caod = 1.4544677036   #Aortic distal compliance  (ml/mmHg)
Csap = 1.4843409851   #Prox. systemic arteries compliance  (ml/mmHg)
Csc  = 7.9822364317   #Systemic capillaries compliance  (ml/mmHg)

#Inertances:
Laop =3.5E-4          #Proximal aorta inertance  (mmHg*sec^2/ml)
Laod =3.5E-4          #Distal aorta inertance  (mmHg*sec^2/ml)
#Other parameters:
Kc = 497.7852450367   #Active vasomotor tone scaling parameter for systemic
                          #arterial pressure (mmHg)
D0 = 50.0             #Active vasomotor tone volume parameter for systemic
                          #arterial pressure (ml)
Vsa0=485.7624931891   #Minimal volume of systemic arteries (ml)
Vsa_max=577.7106000108#Maximal luminal volume of systemic arteries (ml)
Kp1 = 0.03            #Passive vasomotor tone scaling parameter for systemic
                          #arterial pressure (mmHg)
Kp2 = 0.05            #Passive vasomotor tone scaling parameter for systemic
                          #arterial pressure (mmHg/ml^2)
Kr = 0.01             #Pressure scaling constant for systemic arterial
                          #resistance (mmHg*sec/ml)
Rsao = 0.5508058134   #Arteriolar resistance offset (mmHg*sec/ml)
tau_p = 0.1           #Passive vasomotor tone constant for systemic
                          #arterial pressure (ml^-1)
Ksv = 21.83           #Scaling factor for systemic venous pressure (mmHg)
Vmax_sv=3379.5450000637#Maximal volume of lumped systemic veins (ml)
D2 = -5.0             #Offsetting constant for partially collapsed Vena cava
                          #pressure (mmHg)
K1 =0.0968305478      #Scaling constant for unstressed and distended Vena
                          #cava pressure(mmHg/ml)
K2 = 0.4              #Scaling constant for partially collapsed Vena cava
                          #pressure (mmHg)
KR = 0.001            #Scaling factor for Vena cava resistance (mmHg*sec/ml)
```

```
R0 = 0.025            #Vena cava resistance offset parameter (mmHg*sec/ml)
V0 = 129.6486000024   #Unstressed volume of Vena cava (ml)
Vmax_vc=350.5314000065#Maximum volume of Vena cava (ml)
Vmin_vc=50.0107470009 #Mimimum volume of Vena cava (ml)

COtau = 15.0          #Cardiac output equation time constant
Pplc = -5.6           #Pleural chamber pressure (mmHg)
Px2 = 2.0             #P-V curve shaping parameter  (mmHg)
Px1 = 1.0             #P-V curve shaping parameter (mmHg)
Vx1 = 1.0             #P-V curve shaping parameter (ml)
Vx8 = 8.0             #P-V curve shaping parameter (ml)
Vx75 = 75.0           #P-V curve shaping parameter (ml)

#-------------------------------------------------------------------------
#  III. PARAMETERS OF  PULMONARY  CIRCULATION
#-------------------------------------------------------------------------

#Resistances:
Rpuv = 0.0001         #Pulmonary valve resistance  (mmHg*sec/ml)
Rtpap = 0.05          #Proximal pulmonary arterial transmural
                          #resistance(mmHg*sec/ml)
Rtpad = 0.05          #Distal pulmonary arterial transmural
                          #resistance  (mmHg*sec/ml)
Rpap = 0.0001         #Proximal pulmonary resistance  (mmHg*sec/ml)
Rpad = 0.03           #Distal proximal pulmonary resistance  (mmHg*sec/ml)
Rps = 4.2958026137    #Pulmonary shunt resistance  (mmHg*sec/ml)
Rpa = 0.0565149137    #Pulmonary arterioles resistance  (mmHg*sec/ml)
Rpc = 0.0309026688    #Pulmonary capillaries resistance  (mmHg*sec/ml)
Rpv = 0.0001          #Pulmonary veins resistance  (mmHg*sec/ml)

#Compliances:
Ctpap = 1.5365929068  #Proximal pulmonary arterial compliance (ml/mmHg)
Ctpad = 2.6893667388  #Distal pulmonary arterial compliance (ml/mmHg)
Cpa = 3.1321449506    #Pulmonary arterioles compliance (ml/mmHg)
Cpc = 7.7147016012    #Pulmonary capillaries compliance (ml/mmHg)
Cpv = 27.87028922     #Pulmonary veins compliance (ml/mmHg)

#Inductors:
Lpa =1.801907E-4      #Pulmonary arterial inertance (mmHg*sec^2/ml)
Lpad=1.932239E-4      #Distal pulmonary artery inertance (mmHg*sec^2/ml)


#-------------------------------------------------------------------------
# IV.  PARAMETERS  OF  BARORECEPTOR  MODEL
```

```
#-----------------------------------------------------------------------

#Baroreceptor Firing Rate is Nbr.
#    Firing rate sent to Central Nervous System (CNS).
#    CNS filters the Nbr signal and outputs efferent firing frequencies:
# 1. N_hrv - vagal pathway firing frequency
# 2. N_hrs - sympathetic pathway controlling heart rate firing frequency
# 3. N_con - sympathetic pathway controlling heart contractility
# 4. N_vaso - sympathetic pathway controlling vasomotor tone

a = 0.001      #Time constant for baroreceptor firing rate (sec)
               #NOTE: I made up the value for 'a' without any reference(DB)
a1 = 0.036     #Time constant for baroreceptor firing rate (sec)
a2 = 0.0018    #Time constant for baroreceptor firing rate (sec)
K = 1.0        #Baroreceptor gain (used to account for units) (1/(sec*mmHg))

#IVa. HRV pathway
K_hrv = 0.8    #CNS gain for vagal heart rate control
T_hrv = 1.8    #time parameter for vagal heart rate control (sec)
L_hrv = 0.2    #CNS time delay for vagal heart rate control (sec)
a_hrv = 0.372  #Time constant for efferent vagal firing
tau_hrv =-0.04 #Time parameter for efferent vagal firing (sec)
N0_hrv = 110.0 #Frequency parameter for efferent vagal firing (sec^-1)

#IVb. HRS pathway
K_hrs = 1.0    #CNS gain for sympathetic heart rate control
T_hrs = 10.0   #CNS time parameter for sympathetic heart rate control (sec)
L_hrs = 3.0    #CNS time delay for sympathetic heart rate control (sec)
a_hrs = -0.283 #Time constant for efferent sympathetic heart rate firing
tau_hrs = 0.09 #Time parameter for efferent sympathetic heart rate
               #firing (sec)
N0_hrs = 100.0 #Frequency parameter for efferent sympathetic heart rate
               #firing (sec^-1)

#IVc. CON pathway
K_con = 1.0    #CNS gain for contractility control
T_con = 10.0   #CNS time parameter for contractility control (sec)
L_con = 3.0    #CNS time delay for contractility control (sec)
a_con = 0.0483 #Time constant for efferent contractility firing
b_con = 0.7    #Time constant for efferent sympathetic contractility firing
tau_con = 0.04 #Time parameter for efferent sympathetic contractility
               #firing (sec)
N0_con = 110.0 #Frequency parameter for efferent sympathetic contractility
               #firing (sec^-1)
```

```
#IVd. VASO pathway
K_vaso = 1.0   #CNS gain for vasomotor tone control
T_vaso = 6.0   #CNS time parameter for vasomotor tone control (sec)
L_vaso = 3.0   #CNS time delay for vasomotor tone control  (sec)
a_vaso =-0.411 #Time constant for efferent vasomotor tone firing
tau_vaso =0.04 #Time parameter for efferent vasomotor tone firing (sec)
N0_vaso =110.0 #Frequency parameter for efferent vasomotor tone
               #firing (sec^-1)

#IVe. Heart rate control parameters
h1 = 35.0/60    #Heart rate control offset parameter  (sec^-1)
h2 = 140.0/60   #Heart rate control scaling parameter (sec^-1)
h3 = 40.0/60    #Heart rate control scaling parameter (sec^-1)
h4 = 32.0/60    #Heart rate control scaling parameter (sec^-1)
h5 = 10.0/60    #Heart rate control scaling parameter (sec^-1)
h6 = 20.0/60    #Heart rate control scaling parameter (sec^-1)

#IVf. Contractility control parameters
amin = -1.5     #Contractility control offset
#bmin = 0.7     #Contractility control offset
Ka = 5.0        #Contractility control scaling factor
#Kb = 0.5       #Contractility control scaling factor

#-------------------------------------------------------------------------
#  V.  PARAMETERS OF  PERICARDIUM
#-------------------------------------------------------------------------

K_pcd = 1.0     #Scaling parameter for pericardial P-V curve (mmHg)
phi_pcd = 40.0  #Pericardial P-V curve parameter (ml)
Vpcd0 = 785.0  #Pericardial P-V curve parameter  (ml)
perifl = 15.0   #Pericardial fluid in pericardial cavity (ml)

#-------------------------------------------------------------------------
#  VI.  PARAMETERS OF  CORONARY  CIRCULATION
#-------------------------------------------------------------------------

#Resistance Parameters
Rcorao = 2.6423673077 #Proximal epicardial arteries resistance (s*mmHg/ml)
Rcorea = 2.6423673077 #Distal epicardial arteries resistance (s*mmHg/ml)
Rcorla = 5.0733452308 #Large coronary arteries resistance (s*mmHg/ml)
Rcorsa = 5.0733452308 #Small coronary arteries resistance (s*mmHg/ml)
Rcorcap= 4.2277876923 #Coronary capillaries resistance (s*mmHg/ml)
Rcorsv = 0.4932418974 #Small coronary veins resistance (s*mmHg/ml)
```

```
Rcorlv = 0.4932418974 #Large coronary veins resistance (s*mmHg/ml)
Rcorev = 0.4932418974 #Epicardial veins resistance (s*mmHg/ml)

#Compliance Parameters
Ccorao = 0.13         #Compliance of proximal epicardial arteries (ml/mmHg)
Ccorea = 0.0550702742 #Compliance of distal epicardial arteries (ml/mmHg)
Ccorla = 0.0912942282 #Compliance of large coronary arteries (ml/mmHg)
Ccorsa = 0.1560208066 #Compliance of small coronary arteries (ml/mmHg)
Ccorcap= 1.8          #Compliance of coronary capillaries (ml/mmHg)
Ccorsv = 0.5801545598 #Compliance of small coronary veins (ml/mmHg)
Ccorlv = 0.6837210566 #Compliance of large coronary veins (ml/mmHg)
Ccorev = 0.8322992527 #Compliance of epicardial veins (ml/mmHg)

""",0)


def solve_system_beat(z,t,parameters,nothing):

    exec(parameters)
    neq=len(z)
    dz=zeros(neq)
    """
    #-------------------------------------------------------------------------
    #  VII. VARIABLES  OF  VARYING  ELASTANCE  HEART  MODEL
    #-------------------------------------------------------------------------
    """
    Vla=z[0]     #Volume of left atrium (ml)
    Vlv=z[1]     #Volume of left ventricle (ml)
    Vra=z[2]     #Volume of right atrium (ml)
    Vrv=z[3]     #Volume of right ventricle (ml)
    COutput=z[4] #Cardiac output (ml/sec)
    """
    #-------------------------------------------------------------------------
    #  VIII.  VARIABLES OF  SYSTEMIC  CIRCULATION
    #-------------------------------------------------------------------------
    """
    Paopc=z[5]   #Proximal aorta chamber pressure (mmHg)
    PaodFOL=z[7] #Paod follower (mmHg)
    MAP=z[6]     #Mean proximal aortic pressure (mmHg)

    Qaop=z[15]   #Proximal aorta flow (ml/sec)   #(L/min)
    Qaod=z[16]   #Distal aorta flow (ml/sec)

    Vaop=z[8]    #Proximal aorta volume (ml)
```

```
Vaod=z[9]    #Distal aorta volume (ml)
Vsa=z[11]    #Systemic arteries volume (ml)
Vsap=z[10]   #Systemic arterioles volume (ml)
Vsc=z[12]    #Systemic capillaries volume (ml)
Vsv=z[13]    #Systemic veins volume (ml)
Vvc=z[14]    #Vena cava volume (ml)
"""
#-----------------------------------------------------------------------
#  IX. VARIABLES OF  PULMONARY  CIRCULATION
#-----------------------------------------------------------------------
"""
Vpap=z[17]   #Proximal pulmonary arterial volume (ml)
Vpad=z[18]   #Distal pulmonary arterial volume (ml)
Vpa=z[19]    #Pulmonary arterioles volume (ml)
Vpc=z[20]    #Pulmonary capillaries volume (ml)
Vpv=z[21]    #Pulmonary veins volume (ml)
Qpap=z[22]   #Proximal pulmonary arterial flow (ml/sec)   #(L/min)
Qpad=z[23]   #Distal pulmonary arterial flow (ml/sec)

"""
//-----------------------------------------------------------------------
//  XII.  VARIABLES OF  CORONARY  CIRCULATION
//-----------------------------------------------------------------------
"""
Vcorao=z[24] #Proximal epicardial arteries volume (ml)
Vcorea=z[25] #Distal epicardial arteries volume (ml)
Vcorla=z[26] #Large coronary arteries volume (ml)
Vcorsa=z[27] #Small coronary arteries volume (ml)
Vcorcap=z[28]#Coronary capillaries volume (ml)
Vcorsv=z[29] #Small coronary veins volume (ml)
Vcorlv=z[30] #Large coronary veins volume (ml)
Vcorev=z[31] #Epicardial veins volume (ml)
"""
//-----------------------------------------------------------------------
//  XIII. EQUATIONS  OF VARYING  ELASTANCE  HEART  MODEL
//-----------------------------------------------------------------------
"""
"""
Heldt (2002) elastance model
"""
if (t+t_delta>=(hr.t_shift+(1/hr.HR))):
    """New beat"""
    hr.NewBeat(t)
```

```
ya,yv=hr.Activation_function(t,PRint)
"""..........................................................."""
"""RA"""
Era = (Emaxra-Eminra)*ya + Eminra #Elastance driver from activation function
Vrar= (1-ya)*(Vrard-Vrars) + Vrars
"""RV"""
Erv = (Emaxrv-Eminrv)*yv + Eminrv
Vrvr= (1-yv)*(Vrvrd-Vrvrs) + Vrvrs
"""LA"""
Ela = (Emaxla-Eminla)*ya + Eminla
Vlar= (1-ya)*(Vlard-Vlars) + Vlars
"""LV"""
Elv = (Emaxlv-Eminlv)*yv + Eminlv
Vlvr= (1-yv)*(Vlvrd-Vlvrs) + Vlvrs
"""
//-----------------------------------------------------------------------
//  XVII.  EQUATIONS OF  PERICARDIUM
//-----------------------------------------------------------------------
"""
Vpcd = (Vrv + Vra + Vlv + Vla + perifl + Vmyo +
        Vcorao+Vcorea+Vcorla+Vcorsa+Vcorcap+Vcorsv+Vcorlv+Vcorev)
Ppcdc = K_pcd*exp((Vpcd-Vpcd0)/phi_pcd) - Px2/(exp(Vpcd/Vx75)-1)+Pplc
#Eq.from Sun et al.
"""
//-----------------------------------------------------------------------
//  XIII. EQUATIONS  OF  VARYING  ELASTANCE  HEART  MODEL  CONTINUE
//-----------------------------------------------------------------------
"""
if hr.baro_on[2]:
    cont=hr.SympatheticContractivity()
else:cont=0

"""Left atrial pressure"""
Plac = (Vla-Vlar)*Ela- Px2/(exp(Vla/Vx8)-1) + Ppcdc        #Eq. B,F
"""Left ventricle pressure, contractility effects systolic elastance """
Plvc = (1-cont)*(Vlv-Vlvr)*Elv- Px2/(exp(Vlv/Vx8)-1) + Ppcdc     #Eq. B,F
"""Right atrial pressure"""
Prac = (Vra-Vrar)*Era- Px2/(exp(Vra/Vx8)-1) + Ppcdc        #Eq. B,F
"""Right ventricle pressure, contractility effects systolic elastance"""
Prvc = (1-cont)*(Vrv-Vrvr)*Erv- Px2/(exp(Vrv/Vx8)-1) + Ppcdc     #Eq. B,F

"""
//-----------------------------------------------------------------------
//  XV. EQUATIONS OF  PULMONARY  CIRCULATION
```

```
//-------------------------------------------------------------------
"""
Ppapc1 =((Vpap/Ctpap -Px2/(exp(Vpap/Vx8)-1) - Rtpap*Qpap + Pplc)
         *Rpuv+ Prvc*Rtpap)/(Rtpap+Rpuv)                          #Eq. B,C,D
Ppapc2 =( Vpap/Ctpap -Px2/(exp(Vpap/Vx8)-1) - Rtpap*Qpap + Pplc) #Eq. B,C,D
if (Prvc>Ppapc1):Ppapc =Ppapc1
else:            Ppapc =Ppapc2
"""
//-------------------------------------------------------------------
//  XIII. EQUATIONS  OF  VARYING  ELASTANCE  HEART  MODEL  CONTINUE
//-------------------------------------------------------------------
"""
if(Plac>Plvc): Qla =(Plac-Plvc)/Rla       #Eq. C; Mitral valve
else:Qla =0

if(Plvc>Paopc):Qlv =(Plvc-Paopc)/Rav      #Eq. C; Aortic valve
else:Qlv =0

if(Prac>Prvc): Qra =(Prac-Prvc)/Rra       #Eq. C; Tricuspid valve
else:Qra =0

if(Prvc>Ppapc):Qrv =(Prvc-Ppapc)/Rpuv     #Eq. C; Pulm valve
else:Qrv =0

SV = COutput/hr.HR                        #Stroke volume
"""
//-------------------------------------------------------------------
//  XIV. EQUATIONS OF  SYSTEMIC  CIRCULATION
//-------------------------------------------------------------------
"""
"""Pressures:"""
Psc  = Vsc/Csc- Px2/(exp(Vsc/Vx8)-1)                #Eq. B
Psv  = -Ksv*log10((Vmax_sv/Vsv)-0.99)               #Lu et al. Eq. (1)

if(Vvc>V0):
    Pvcc= D2+K2*exp(V0 /Vmin_vc)-Px2/(exp(Vvc/Vx8)-1)+K1*(Vvc-V0)+ Pplc
    #Lu et al. Eq. (2)
else:
    Pvcc= D2+K2*exp(Vvc/Vmin_vc)-Px2/(exp(Vvc/Vx8)-1)+ Pplc
    #Lu et al. Eq. (3)

Paodc = (Rcrb*(Rtaod*Qaop- Rtaod*Qaod + Vaod/Caod- Px2/(exp(Vaod/Vx8)-1))+
         Pvcc*Rtaod)/(Rcrb+Rtaod)#Eq. B,C
Paod = Paodc - Pbs
```

```
dPaodFOL = (Paod-PaodFOL)/(0.0005)
"""
//-------------------------------------------------------------------------
//  XVI.  EQUATIONS OF  BARORECEPTOR  MODEL
//-------------------------------------------------------------------------
"""
F_vaso=0.0#0.52
"""
//-------------------------------------------------------------------------
//  XVIII. EQUATIONS OF  CORONARY  CIRCULATION
//-------------------------------------------------------------------------
"""
Pcoreac= Vcorea/Ccorea - Px1/(exp(Vcorea/Vx1)-1)+ Ppcdc #Eq. B
Qcorao = (Paopc-Pcoreac)/Rcorao                        #Eq. C
"""
//-------------------------------------------------------------------------
//  XIV. EQUATIONS OF  SYSTEMIC  CIRCULATION  CONTINUE
//-------------------------------------------------------------------------
"""
"""Nonlinear resistances:"""
Rvc = KR*(Vmax_vc/Vvc)**2 + R0               #Vena cava: Lu et al. Eq.(4)

Rsa = (Rsao + Kr*(exp(4*F_vaso) +(Vsa_max/Vsa)**2))
#Sys. arteries: Lu et al. Eq.(16)
if hr.baro_on[3]:
    Rsa=Rsa*hr.SympatheticPeripheral()

"""Transmural flow:"""
dVaop =(Paopc - (Vaop/Caop) + Px2/(exp(Vaop/Vx8)-1))/Rtaop#Eq. A,B,C,D
"""Pressures:"""
dPaopc= (Qlv-dVaop-Qaop-
         Qcorao)*(1/Ccorao+Px2*exp(Vcorao/Vx1)/(exp(Vcorao/Vx1)-1)**2)

Psap  = Vsap/Csap - Px2/(exp(Vsap/Vx8)-1)                #Eq. B

Psa_a = Kc*log10( (Vsa-Vsa0)/D0 + 1)                    #Lu et al. Eq. (13)
Psa_p = Kp1*exp(tau_p*(Vsa-Vsa0)) + Kp2*(Vsa-Vsa0)**2    #Lu et al. Eq. (14)
Psa   = F_vaso*Psa_a + (1-F_vaso)*Psa_p                  #Lu et al. Eq. (15)

if(t>= hr.t_shift_dt+t_delta):
    hr.SumMap(Psa,t_delta)


dMAP  = (Psa-MAP)/COtau
```

```
"""Flows:"""
Qcrb = (Paodc-Pvcc)/Rcrb                                    #Eq. C
Qsap = (Psap-Psa)/Rsap                                      #Eq. C
Qsa  = (Psa-Psc)/Rsa                                        #Eq. C
Qsc  = (Psc-Psv)/Rsc                                        #Eq. C
Qsv  = (Psv-Pvcc)/Rsv                                       #Eq. C
Qvc  = (Pvcc-Prac)/Rvc                                      #Eq. C

"""Transmural flows:"""
dVaod = Qaop- Qaod - Qcrb                                   #Eq. A,D
dVsap = Qaod- Qsap                                          #Eq. A,D
dVsa  = Qsap- Qsa                                           #Eq. A,D
dVsc  = Qsa - Qsc                                           #Eq. A,D
dVsv  = Qsc - Qsv                                           #Eq. A,D
dVvc  = Qsv + Qcrb - Qvc                                    #Eq. A,D

"""Differential equations:"""
dQaop = (Paopc - Qaop*Raop - Paodc)/ Laop                   #Eq. C,E
dQaod = (Paodc - Qaod*Raod - Psap) / Laod                   #Eq. C,E

"""
//-------------------------------------------------------------------------
//  XV. EQUATIONS OF  PULMONARY  CIRCULATION  CONTINUE
//-------------------------------------------------------------------------
"""
Ppadc = Vpad/Ctpad-Px2/(exp(Vpad/Vx8)-1) +(Qpap-Qpad)*Rtpad+ Pplc#Eq. B,C
Ppac  = Vpa /Cpa - Px2/(exp(Vpa/Vx8)-1)+ Pplc                  #Eq. B
Ppcc  = Vpc /Cpc - Px2/(exp(Vpc/Vx8)-1)+ Pplc                  #Eq. B
Ppvc  = Vpv /Cpv - Px2/(exp(Vpv/Vx8)-1)+ Pplc                  #Eq. B

Qps = (Ppac-Ppvc)/Rps                                       #Eq. C
Qpa = (Ppac-Ppcc)/Rpa                                       #Eq. C
Qpc = (Ppcc-Ppvc)/Rpc                                       #Eq. C
Qpv = (Ppvc-Plac)/Rpv                                       #Eq. C

dVpap = Qrv - Qpap                                          #Eq. A,D
dVpad = Qpap- Qpad                                          #Eq. A,D
dVpa  = Qpad- Qps- Qpa                                      #Eq. A,D
dVpc  = Qpa - Qpc                                           #Eq. A,D
dVpv  = Qpc + Qps- Qpv

dQpap= (Ppapc - Qpap*Rpap- Ppadc)/Lpa                       #Eq. C,E
dQpad= (Ppadc - Qpad*Rpad- Ppac )/Lpad                      #Eq. C,E
"""
```

```
//-----------------------------------------------------------------------
//  XVIII. EQUATIONS OF  CORONARY  CIRCULATION CONTINUE
//-----------------------------------------------------------------------
"""
Pcorisfc= abs((Plvc-Ppcdc)/2)
Pcorlac = Vcorla/Ccorla - Px1/(exp(Vcorla/Vx1)-1) + Pcorisfc  #Eq. B
Pcorsac = Vcorsa/Ccorsa - Px1/(exp(Vcorsa/Vx1)-1) + Pcorisfc  #Eq. B
Pcorcapc= Vcorcap/Ccorcap-Px1/(exp(Vcorcap/Vx1)-1)+ Pcorisfc  #Eq. B
Pcorsvc = Vcorsv/Ccorsv - Px1/(exp(Vcorsv/Vx1)-1) + Ppcdc     #Eq. B
Pcorlvc = Vcorlv/Ccorlv - Px1/(exp(Vcorlv/Vx1)-1) + Ppcdc     #Eq. B
Pcorevc = Vcorev/Ccorev - Px2/(exp(Vcorev/Vx8)-1) + Ppcdc     #Eq. B

Qcorea = (Pcoreac-Pcorlac)/Rcorea                             #Eq. C
Qcorla = (Pcorlac-Pcorsac)/Rcorla                             #Eq. C
Qcorsa = (Pcorsac-Pcorcapc)/Rcorsa                            #Eq. C
Qcorcap= (Pcorcapc-Pcorsvc)/Rcorcap                           #Eq. C
Qcorsv = (Pcorsvc-Pcorlvc)/Rcorsv                             #Eq. C
Qcorlv = (Pcorlvc-Pcorevc)/Rcorlv                             #Eq. C
Qcorev = (Pcorevc-Prac)/Rcorev                                #Eq. C

dVcorao = Qlv-dVaop-Qaop-Qcorao                               #Eq. A,D
dVcorea = Qcorao - Qcorea                                     #Eq. A,D
dVcorla = Qcorea - Qcorla                                     #Eq. A,D
dVcorsa = Qcorla - Qcorsa                                     #Eq. A,D
dVcorcap= Qcorsa - Qcorcap                                    #Eq. A,D
dVcorsv = Qcorcap- Qcorsv                                     #Eq. A,D
dVcorlv = Qcorsv - Qcorlv                                     #Eq. A,D
dVcorev = Qcorlv - Qcorev                                     #Eq. A,D

"""
//-----------------------------------------------------------------------
//  XIII. EQUATIONS  OF  VARYING  ELASTANCE  HEART  MODEL  CONTINUE
//-----------------------------------------------------------------------
"""

"""Left atrium, and its contraction:"""
dVla = (Qpv-Qla)                              #Eq. A,D
"""Left ventricle, and its contraction, blood loss function here  """
dVlv = (Qla - Qlv)                            #Eq. A,D
"""Right atrium, and its contraction:"""
dVra = (Qvc - Qra + Qcorev)                   #Eq. A,D
"""Right ventricle, and its contraction:"""
dVrv = (Qra - Qrv)                            #Eq. A,D
dCOutput = (Qlv-COutput)/COtau                #Cardiac output
```

```
    dz[0] =dVla
    dz[1] =dVlv
    dz[2] =dVra
    dz[3] =dVrv
    dz[4] =dCOutput
    dz[5] =dPaopc
    dz[6] =dMAP
    dz[7] =dPaodFOL
    dz[8] =dVaop
    dz[9] =dVaod
    dz[10]=dVsap
    dz[11]=dVsa
    dz[12]=dVsc
    dz[13]=dVsv
    dz[14]=dVvc
    dz[15]=dQaop
    dz[16]=dQaod
    dz[17]=dVpap
    dz[18]=dVpad
    dz[19]=dVpa
    dz[20]=dVpc
    dz[21]=dVpv
    dz[22]=dQpap
    dz[23]=dQpad
    dz[24]=dVcorao
    dz[25]=dVcorea
    dz[26]=dVcorla
    dz[27]=dVcorsa
    dz[28]=dVcorcap
    dz[29]=dVcorsv
    dz[30]=dVcorlv
    dz[31]=dVcorev

    return dz


def solve_system(z,t,parameters,nothing):

    exec(parameters)
    """
    //-------------------------------------------------------------------------
    //  VII. VARIABLES  OF  VARYING  ELASTANCE  HEART  MODEL
    //-------------------------------------------------------------------------
```

```
"""
neq=len(z)
dz=zeros(neq)
Vla=z[0]     #Volume of left atrium (ml)
Vlv=z[1]     #Volume of left ventricle (ml)
Vra=z[2]     #Volume of right atrium (ml)
Vrv=z[3]     #Volume of right ventricle (ml)
COutput=z[4] #Cardiac output (ml/sec)
"""
//-------------------------------------------------------------------------
//  VIII.  VARIABLES OF  SYSTEMIC  CIRCULATION
//-------------------------------------------------------------------------
"""
Paopc=z[5]   #Proximal aorta chamber pressure (mmHg)
PaodFOL=z[7] #Paod follower (mmHg)
MAP=z[6]     #Mean proximal aortic pressure (mmHg)
"""Flows:"""
Qaop=z[15]   #Proximal aorta flow (ml/sec)   #(L/min)
Qaod=z[16]   #Distal aorta flow (ml/sec)
"""Volumes:"""
Vaop=z[8]    #Proximal aorta volume (ml)
Vaod=z[9]    #Distal aorta volume (ml)
Vsa=z[11]    #Systemic arteries volume (ml)
Vsap=z[10]   #Systemic arterioles volume (ml)
Vsc=z[12]    #Systemic capillaries volume (ml)
Vsv=z[13]    #Systemic veins volume (ml)
Vvc=z[14]    #Vena cava volume (ml)
"""
//-------------------------------------------------------------------------
//  IX. VARIABLES OF  PULMONARY  CIRCULATION
//-------------------------------------------------------------------------
"""
Vpap=z[17]   #Proximal pulmonary arterial volume (ml)
Vpad=z[18]   #Distal pulmonary arterial volume (ml)
Vpa=z[19]    #Pulmonary arterioles volume (ml)
Vpc=z[20]    #Pulmonary capillaries volume (ml)
Vpv=z[21]    #Pulmonary veins volume (ml)
Qpap=z[22]   #Proximal pulmonary arterial flow (ml/sec)   #(L/min)
Qpad=z[23]   #Distal pulmonary arterial flow (ml/sec)
"""
//-------------------------------------------------------------------------
//  X.  VARIABLES OF  BARORECEPTOR  MODEL
//-------------------------------------------------------------------------
"""
```

```
Nbr=z[24]    #Instantaneous firing frequency of baroreceptor (1/sec)
Nbr_t=z[25]  #Time derivative of baroreceptor firing rate (sec^-2)
N_hrv=z[26]  #Vagal discharge rate at CNS controlling heart rate (1/sec)
N_hrs=z[27]  #Sympathetic discharge rate at CNS controlling heart rate(1/sec)
N_con=z[28]  #Sympathetic discharge rate at CNS for contractility (1/sec)
N_vaso=z[29] #Sympathetic discharge rate at CNS controlling vasomotor
             #tone (1/sec)
"""
//-------------------------------------------------------------------------
//  XII.  VARIABLES OF  CORONARY  CIRCULATION
//-------------------------------------------------------------------------
"""
Vcorao=z[30] #Proximal epicardial arteries volume (ml)
Vcorea=z[31] #Distal epicardial arteries volume (ml)
Vcorla=z[32] #Large coronary arteries volume (ml)
Vcorsa=z[33] #Small coronary arteries volume (ml)
Vcorcap=z[34]#Coronary capillaries volume (ml)
Vcorsv=z[35] #Small coronary veins volume (ml)
Vcorlv=z[36] #Large coronary veins volume (ml)
Vcorev=z[37] #Epicardial veins volume (ml)
"""
//-------------------------------------------------------------------------
//  XIII. EQUATIONS  OF VARYING  ELASTANCE  HEART  MODEL
//-------------------------------------------------------------------------
"""
"""
Heldt (2002) elastance model
"""
if (t+t_delta>=(hr.t_shift+(1/hr.HR))):
    hr.t_shift = t
    hr.shift=True

trela = t-hr.t_shift
trelv = t-hr.t_shift-PRint

if (0.0<=trela and trela<hr.Tsa):
    ya=(1.0 - cos(pi*trela/hr.Tsa))/2.0
elif (hr.Tsa<=trela and trela<1.5*hr.Tsa):
    ya=(1.0 + cos(2.0*pi*(trela-hr.Tsa)/hr.Tsa))/2.0
else:
    ya=0.0

if (0.0<=trelv and trelv<hr.Tsv):
    yv=(1.0 - cos(pi*trelv/hr.Tsv))/2.0
```

```
elif (hr.Tsv<=trelv and trelv<1.5*hr.Tsv):
    yv=(1.0 + cos(2.0*pi*(trelv-hr.Tsv)/hr.Tsv))/2.0
else:
    yv=0.0
"""..............................................................."""
"""RA"""
Era = (Emaxra-Eminra)*ya + Eminra #Elastance driver from activation function
Vrar= (1-ya)*(Vrard-Vrars) + Vrars
"""RV"""
Erv = (Emaxrv-Eminrv)*yv + Eminrv
Vrvr= (1-yv)*(Vrvrd-Vrvrs) + Vrvrs
"""LA"""
Ela = (Emaxla-Eminla)*ya + Eminla
Vlar= (1-ya)*(Vlard-Vlars) + Vlars
"""LV"""
Elv = (Emaxlv-Eminlv)*yv + Eminlv
Vlvr= (1-yv)*(Vlvrd-Vlvrs) + Vlvrs
"""
//-----------------------------------------------------------------------
//  XVII.  EQUATIONS OF  PERICARDIUM
//-----------------------------------------------------------------------
"""
Vpcd = (Vrv + Vra + Vlv + Vla + perifl + Vmyo +
        Vcorao+Vcorea+Vcorla+Vcorsa+Vcorcap+Vcorsv+Vcorlv+Vcorev)
Ppcdc = K_pcd*exp((Vpcd-Vpcd0)/phi_pcd) - Px2/(exp(Vpcd/Vx75)-1)+Pplc
#Eq.from Sun et al.
"""
//-----------------------------------------------------------------------
//  XIII. EQUATIONS  OF  VARYING  ELASTANCE  HEART  MODEL  CONTINUE
//-----------------------------------------------------------------------
"""
"""Left atrial pressure"""
Plac =             (Vla-Vlar)*Ela- Px2/(exp(Vla/Vx8)-1) + Ppcdc #Eq. B,F
"""Left ventricle pressure, contractility effects systolic elastance """
Plvc = hr.afs_con2*(Vlv-Vlvr)*Elv- Px2/(exp(Vlv/Vx8)-1) + Ppcdc #Eq. B,F
"""Right atrial pressure"""
Prac =             (Vra-Vrar)*Era- Px2/(exp(Vra/Vx8)-1) + Ppcdc #Eq. B,F
"""Right ventricle pressure, contractility effects systolic elastance"""
Prvc = hr.afs_con2*(Vrv-Vrvr)*Erv- Px2/(exp(Vrv/Vx8)-1) + Ppcdc #Eq. B,F

"""
//-----------------------------------------------------------------------
//  XV. EQUATIONS OF  PULMONARY  CIRCULATION
//-----------------------------------------------------------------------
```

```
"""
Ppapc1 =((Vpap/Ctpap -Px2/(exp(Vpap/Vx8)-1) - Rtpap*Qpap + Pplc)
         *Rpuv+ Prvc*Rtpap)/(Rtpap+Rpuv)                        #Eq. B,C,D

Ppapc2 =( Vpap/Ctpap -Px2/(exp(Vpap/Vx8)-1) - Rtpap*Qpap + Pplc) #Eq. B,C,D
if (Prvc>Ppapc1):Ppapc =Ppapc1
else:            Ppapc =Ppapc2
"""
//-------------------------------------------------------------------
//  XIII. EQUATIONS  OF  VARYING  ELASTANCE  HEART  MODEL  CONTINUE
//-------------------------------------------------------------------
"""
if(Plac>Plvc): Qla =(Plac-Plvc)/Rla       #Eq. C; Mitral valve
else:Qla =0

if(Plvc>Paopc):Qlv =(Plvc-Paopc)/Rav      #Eq. C; Aortic valve
else:Qlv =0

if(Prac>Prvc): Qra =(Prac-Prvc)/Rra       #Eq. C; Tricuspid valve
else:Qra =0

if(Prvc>Ppapc):Qrv =(Prvc-Ppapc)/Rpuv     #Eq. C; Pulm valve
else:Qrv =0

SV = COutput/hr.HR                        #Stroke volume
"""
//-------------------------------------------------------------------
//  XIV. EQUATIONS OF  SYSTEMIC  CIRCULATION
//-------------------------------------------------------------------
"""
"""Pressures:"""
Psc  = Vsc/Csc- Px2/(exp(Vsc/Vx8)-1)                #Eq. B
Psv  = -Ksv*log10((Vmax_sv/Vsv)-0.99)               #Lu et al. Eq. (1)

if(Vvc>V0):
    Pvcc= D2+K2*exp(V0 /Vmin_vc)-Px2/(exp(Vvc/Vx8)-1)+K1*(Vvc-V0)+ Pplc
    #Lu et al. Eq. (2)
else:
    Pvcc= D2+K2*exp(Vvc/Vmin_vc)-Px2/(exp(Vvc/Vx8)-1)+ Pplc
    #Lu et al. Eq. (3)

Paodc = (Rcrb*(Rtaod*Qaop- Rtaod*Qaod + Vaod/Caod- Px2/(exp(Vaod/Vx8)-1))+
         Pvcc*Rtaod)/(Rcrb+Rtaod)#Eq. B,C
Paod = Paodc - Pbs
```

```
dPaodFOL = (Paod-PaodFOL)/(0.0005)
"""
//-------------------------------------------------------------------------
//  XVI.  EQUATIONS OF  BARORECEPTOR  MODEL
//-------------------------------------------------------------------------
"""
"""
The governing equation for Nbr is adopted from
Spickler et al.(Kezdi and Geller, 1967).

The general form of the equation determining efferent
responses to the baroreceptor:

T_x*(N_x:t) + N_x = K_x*Nbr(t-L_x), Lu et al. Eq. in CNS box of Fig 2
and
F_x = a_x +b_x/(exp(tau_x*(N_x-No_x))+1.0),Lu et al. Eq. (7)
where the subscript "x" denotes the pathway name.
"""
if(t>= hr.t_shift_Nbr+t_delta):
    hr.Nbr.append(Nbr)
    hr.t_shift_Nbr+=t_delta


dNbr = Nbr_t
#Spickler et al. p. 34, also Lu et al. Eq. (6):
dNbr_t=1.0/(a2*a) *(-(a2+a)*Nbr_t -Nbr +K*Paod +a1*K*dPaodFOL)

if hr.baro_on[0]:
    """// Discharge frequency controlling heart rate (vagal pathway)"""
    if (t_min+t>L_hrv):
        dN_hrv =(-N_hrv + (K_hrv*hr.Nbr[-int(L_hrv/t_delta)]))/T_hrv
    else:dN_hrv = 0
    b_hrv = 1 - a_hrv
    F_hrv = a_hrv + (b_hrv / (exp(tau_hrv*(N_hrv-N0_hrv)) + 1.0))
else:
    F_hrv=0
    dN_hrv=0

if hr.baro_on[1]:
    """Discharge frequency controlling heart rate (sympathetic)"""
    if (t_min+t>L_hrs):
        dN_hrs =(-N_hrs + (K_hrs*hr.Nbr[-int(L_hrs/t_delta)]))/T_hrs
    else: dN_hrs =0
    b_hrs = 1 - a_hrs
```

```
        F_hrs = a_hrs + (b_hrs / (exp(tau_hrs*(N_hrs-N0_hrs)) + 1.0))
    else:
        F_hrs=0
        dN_hrs =0

    if hr.baro_on[2]:
        """Discharge frequency controlling contractility of heart """
        if (t_min+t>L_con):
            dN_con =(-N_con + (K_con*hr.Nbr[-int(L_con/t_delta)]))/T_con
        else:dN_con =0
        F_con = a_con + (b_con / (exp(tau_con*(N_con-N0_con)) + 1.0))
    else:
        F_con=0
        dN_con =0

    if hr.baro_on[3]:
        """Discharge frequency controlling vasomotor tone"""
        if (t_min+t>L_vaso):
            dN_vaso =(-N_vaso + (K_vaso*hr.Nbr[-int(L_vaso/t_delta)]))/T_vaso
        else:dN_vaso =0
        b_vaso = 1- a_vaso
        F_vaso = a_vaso + (b_vaso / (exp(tau_vaso*(N_vaso-N0_vaso)) + 1.0))
    else:
        F_vaso=0
        dN_vaso =0

    """Heart rate control"""
    if hr.baro_on[0] and hr.baro_on[1]:
        HRcont = (h1 + (h2*F_hrs)-(h3*F_hrs**2)-(h4*F_hrv)+
                  (h5*F_hrv**2)-(h6*F_hrv*F_hrs)) #Lu et al. Eq. (8)
    else:
        HRcont =hr.HR

    """Contractility control"""
    if hr.baro_on[2]:
        afs_con = amin + (Ka*F_con)
    else:
        afs_con=1
    #bfs_con = bmin + (Kb*F_con)
    """
    //-------------------------------------------------------------------------
    //  XVIII. EQUATIONS OF  CORONARY  CIRCULATION
    //-------------------------------------------------------------------------
    """
```

```
Pcoreac= Vcorea/Ccorea - Px1/(exp(Vcorea/Vx1)-1)+ Ppcdc #Eq. B
Qcorao = (Paopc-Pcoreac)/Rcorao                        #Eq. C
"""
//-------------------------------------------------------------------------
//  XIV. EQUATIONS OF  SYSTEMIC  CIRCULATION  CONTINUE
//-------------------------------------------------------------------------
"""
"""Nonlinear resistances:"""
Rvc = KR*(Vmax_vc/Vvc)**2 + R0                 #Vena cava: Lu et al. Eq.(4)
Rsa = Rsao + Kr*(exp(4*F_vaso) +(Vsa_max/Vsa)**2)
#Sys. arteries: Lu et al. Eq.(16)

"""Transmural flow:"""
dVaop =(Paopc - (Vaop/Caop) + Px2/(exp(Vaop/Vx8)-1))/Rtaop#Eq. A,B,C,D
"""Pressures:"""
dPaopc= (Qlv-dVaop-Qaop-
         Qcorao)*(1/Ccorao+Px2*exp(Vcorao/Vx1)/(exp(Vcorao/Vx1)-1)**2)

Psap  = Vsap/Csap - Px2/(exp(Vsap/Vx8)-1)                #Eq. B

Psa_a = Kc*log10( (Vsa-Vsa0)/D0 + 1)                   #Lu et al. Eq. (13)
Psa_p = Kp1*exp(tau_p*(Vsa-Vsa0)) + Kp2*(Vsa-Vsa0)**2   #Lu et al. Eq. (14)
Psa   = F_vaso*Psa_a + (1-F_vaso)*Psa_p                 #Lu et al. Eq. (15)
dMAP  = (Psa-MAP)/COtau;
"""Flows:"""
Qcrb = (Paodc-Pvcc)/Rcrb                                 #Eq. C
Qsap = (Psap-Psa)/Rsap                                   #Eq. C
Qsa  = (Psa-Psc)/Rsa                                     #Eq. C
#Qsa  = (Psa-Psap)/Rsa                                    #Eq. C
#Qsap = (Psap-Psc)/Rsap                                   #Eq. C

Qsc  = (Psc-Psv)/Rsc                                     #Eq. C
Qsv  = (Psv-Pvcc)/Rsv                                    #Eq. C
Qvc  = (Pvcc-Prac)/Rvc                                   #Eq. C

"""Transmural flows:"""
dVaod = Qaop- Qaod - Qcrb                                #Eq. A,D
dVsap = Qaod- Qsap                                       #Eq. A,D
dVsa  = Qsap- Qsa                                        #Eq. A,D
dVsc  = Qsa - Qsc                                        #Eq. A,D
#dVsa = Qaod- Qsa                                         #Eq. A,D
#dVsap  = Qsa- Qsap                                       #Eq. A,D
#dVsc  = Qsap - Qsc                                       #Eq. A,D
dVsv  = Qsc - Qsv                                        #Eq. A,D
```

```
dVvc  = Qsv + Qcrb - Qvc                                    #Eq. A,D

"""Differential equations:"""
dQaop = (Paopc - Qaop*Raop - Paodc)/ Laop                   #Eq. C,E
dQaod = (Paodc - Qaod*Raod - Psap) / Laod                   #Eq. C,E
#dQaod = (Paodc - Qaod*Raod - Psa) / Laod                    #Eq. C,E

"""
//-------------------------------------------------------------------------
//  XV. EQUATIONS OF  PULMONARY  CIRCULATION  CONTINUE
//-------------------------------------------------------------------------
"""
Ppadc = Vpad/Ctpad-Px2/(exp(Vpad/Vx8)-1) +(Qpap-Qpad)*Rtpad+ Pplc#Eq. B,C
Ppac  = Vpa /Cpa - Px2/(exp(Vpa/Vx8)-1)+ Pplc                  #Eq. B
Ppcc  = Vpc /Cpc - Px2/(exp(Vpc/Vx8)-1)+ Pplc                  #Eq. B
Ppvc  = Vpv /Cpv - Px2/(exp(Vpv/Vx8)-1)+ Pplc                  #Eq. B

Qps = (Ppac-Ppvc)/Rps                                       #Eq. C
Qpa = (Ppac-Ppcc)/Rpa                                       #Eq. C
Qpc = (Ppcc-Ppvc)/Rpc                                       #Eq. C
Qpv = (Ppvc-Plac)/Rpv                                       #Eq. C

dVpap = Qrv - Qpap                                          #Eq. A,D
dVpad = Qpap- Qpad                                          #Eq. A,D
dVpa  = Qpad- Qps- Qpa                                      #Eq. A,D
dVpc  = Qpa - Qpc                                           #Eq. A,D
dVpv  = Qpc + Qps- Qpv

dQpap= (Ppapc - Qpap*Rpap- Ppadc)/Lpa                       #Eq. C,E
dQpad= (Ppadc - Qpad*Rpad- Ppac )/Lpad                      #Eq. C,E
"""
//-------------------------------------------------------------------------
//  XVIII. EQUATIONS OF  CORONARY  CIRCULATION CONTINUE
//-------------------------------------------------------------------------
"""
Pcorisfc= abs((Plvc-Ppcdc)/2)
Pcorlac = Vcorla/Ccorla - Px1/(exp(Vcorla/Vx1)-1) + Pcorisfc  #Eq. B
Pcorsac = Vcorsa/Ccorsa - Px1/(exp(Vcorsa/Vx1)-1) + Pcorisfc  #Eq. B
Pcorcapc= Vcorcap/Ccorcap-Px1/(exp(Vcorcap/Vx1)-1)+ Pcorisfc  #Eq. B
Pcorsvc = Vcorsv/Ccorsv - Px1/(exp(Vcorsv/Vx1)-1) + Ppcdc     #Eq. B
Pcorlvc = Vcorlv/Ccorlv - Px1/(exp(Vcorlv/Vx1)-1) + Ppcdc     #Eq. B
Pcorevc = Vcorev/Ccorev - Px2/(exp(Vcorev/Vx8)-1) + Ppcdc     #Eq. B

Qcorea = (Pcoreac-Pcorlac)/Rcorea                           #Eq. C
```

```
Qcorla = (Pcorlac-Pcorsac)/Rcorla                          #Eq. C
Qcorsa = (Pcorsac-Pcorcapc)/Rcorsa                         #Eq. C
Qcorcap= (Pcorcapc-Pcorsvc)/Rcorcap                        #Eq. C
Qcorsv = (Pcorsvc-Pcorlvc)/Rcorsv                          #Eq. C
Qcorlv = (Pcorlvc-Pcorevc)/Rcorlv                          #Eq. C
Qcorev = (Pcorevc-Prac)/Rcorev                             #Eq. C

dVcorao = Qlv-dVaop-Qaop-Qcorao                            #Eq. A,D
dVcorea = Qcorao - Qcorea                                  #Eq. A,D
dVcorla = Qcorea - Qcorla                                  #Eq. A,D
dVcorsa = Qcorla - Qcorsa                                  #Eq. A,D
dVcorcap= Qcorsa - Qcorcap                                 #Eq. A,D
dVcorsv = Qcorcap- Qcorsv                                  #Eq. A,D
dVcorlv = Qcorsv - Qcorlv                                  #Eq. A,D
dVcorev = Qcorlv - Qcorev                                  #Eq. A,D

"""
//-------------------------------------------------------------------------
//  XIII. EQUATIONS  OF  VARYING  ELASTANCE  HEART  MODEL  CONTINUE
//-------------------------------------------------------------------------
"""
if (hr.shift):
    hr.HR = HRcont
    hr.Tsv = hr.TsvK*sqrt(1.0/hr.HR)
    hr.Tsa = hr.TsaK*sqrt(1.0/hr.HR)
    hr.shift = False
    hr.afs_con2 = afs_con


"""Left atrium, and its contraction:"""
dVla = (Qpv-Qla)                                #Eq. A,D
"""Left ventricle, and its contraction, blood loss function here"""
dVlv = (Qla - Qlv)                              #Eq. A,D
"""Right atrium, and its contraction:"""
dVra = (Qvc - Qra + Qcorev)                     #Eq. A,D
"""Right ventricle, and its contraction:"""
dVrv = (Qra - Qrv)                              #Eq. A,D
dCOutput = (Qlv-COutput)/COtau                  #Cardiac output

dz[0] =dVla
dz[1] =dVlv
dz[2] =dVra
dz[3] =dVrv
dz[4] =dCOutput
```

```
    dz[5] =dPaopc
    dz[6] =dMAP
    dz[7] =dPaodFOL
    dz[8] =dVaop
    dz[9] =dVaod
    dz[10]=dVsap
    dz[11]=dVsa
    dz[12]=dVsc
    dz[13]=dVsv
    dz[14]=dVvc
    dz[15]=dQaop
    dz[16]=dQaod
    dz[17]=dVpap
    dz[18]=dVpad
    dz[19]=dVpa
    dz[20]=dVpc
    dz[21]=dVpv
    dz[22]=dQpap
    dz[23]=dQpad
    dz[24]=dNbr
    dz[25]=dNbr_t
    dz[26]=dN_hrv
    dz[27]=dN_hrs
    dz[28]=dN_con
    dz[29]=dN_vaso
    dz[30]=dVcorao
    dz[31]=dVcorea
    dz[32]=dVcorla
    dz[33]=dVcorsa
    dz[34]=dVcorcap
    dz[35]=dVcorsv
    dz[36]=dVcorlv
    dz[37]=dVcorev

    return dz



if __name__=='__main__':
    c0=time.clock()
    e0=time.time()
    try:
        outfile = sys.argv[1]
    except:
```

```
        usage = 'Usage: python %s <outfile>' % sys.argv[0]
        print usage; sys.exit(1)

    t_min=0; t_max=1; t_delta=0.01
    N_t=t_max/t_delta
    t=linspace(t_min,t_max,N_t)

    dirname=outfile.split('/')[0]

    """Initial Conditions"""
    Vra_0    =  78.49717445
    Vrv_0    = 167.58374732
    Vla_0    =  85.98610858
    Vlv_0    = 125.30275819
    MAP_0    =  90.51555238
    Qaop_0   =  11.5057925
    Qaod_0   =  23.55735293
    COutput_0= 108.45746221
    Vaop_0   =  31.10372993
    Vaod_0   = 138.2299
    Vsa_0    = 519.72188207
    Vsap_0   = 129.39153837
    Vsc_0    = 256.79837023
    Vsv_0    =2961.8176
    Vvc_0    = 232.53848523
    PaodFOL_0=  87.78931885
    Vpap_0   =  33.08150781
    Vpad_0   =  60.00629041
    Vpa_0    =  58.84204803
    Vpc_0    = 107.52620632
    Vpv_0    = 293.43006456
    Qpap_0   =  20.3730138333
    Qpad_0   =  56.97029945
    Paopc_0  =  87.76812894
    Nbr_0    =  86.70389591
    Nbr_t_0  = -30.55212276
    N_hrv_0  =  76.47010405
    N_hrs_0  =  94.97070286
    N_con_0  =  94.97070286
    N_vaso_0 =  94.92069156
    Vcorao_0 =   2.73896754
    Vcorea_0 =   4.40458414
    Vcorla_0 =   4.98811541
    Vcorsa_0 =   4.21922358
```

```
    Vcorcap_0=   8.55952212
    Vcorsv_0 =   7.82911551
    Vcorlv_0 =   8.21022342
    Vcorev_0 =   8.77002707


    beat_system=True

    if(beat_system):
        extra=InitialSimulation('%s/RELC_NEW2.PTX'%dirname)
        hr=Baroreflex(extra['ModParam'])

        z0=[Vla_0,Vlv_0,Vra_0,Vrv_0,COutput_0,Paopc_0,MAP_0,PaodFOL_0,Vaop_0,
            Vaod_0,Vsap_0,Vsa_0,Vsc_0,Vsv_0,Vvc_0,Qaop_0,Qaod_0,Vpap_0,
            Vpad_0,Vpa_0,Vpc_0,Vpv_0,Qpap_0,Qpad_0,Vcorao_0,
            Vcorea_0,Vcorla_0,Vcorsa_0,Vcorcap_0,Vcorsv_0,Vcorlv_0,Vcorev_0]

        z0=asarray(z0)
        z,info_dict=odeint(solve_system_beat,z0,t,args=parameters,
                           full_output=1 ,printmessg=1 )


    else:
        hr=Heartrate()
        z0=[Vla_0,Vlv_0,Vra_0,Vrv_0,COutput_0,Paopc_0,MAP_0,PaodFOL_0,Vaop_0,
            Vaod_0,Vsap_0,Vsa_0,Vsc_0,Vsv_0,Vvc_0,Qaop_0,Qaod_0,Vpap_0,Vpad_0,
            Vpa_0,Vpc_0,Vpv_0,Qpap_0,Qpad_0,Nbr_0,Nbr_t_0,N_hrv_0,N_hrs_0,
            N_con_0,N_vaso_0,Vcorao_0,Vcorea_0,Vcorla_0,Vcorsa_0,Vcorcap_0,
            Vcorsv_0,Vcorlv_0,Vcorev_0]

        z0=asarray(z0)
        z,info_dict=odeint(solve_system,z0,t,args=parameters,
                           full_output=1 ,printmessg=1 )

    print_to_outfile(t,z,outfile)

    print time.clock()-c0
    print time.time()-e0
"""
// REFERENCE EQUATIONS:

//  Eq. A)  Flow (mL/unit time) = change in volume / change in time
// Basis: Definition of flow
//  Eq. B)  Compliance = Change in volume / Change in pressure
```

```
//       Basis: Fluid analog of capacitance
//  Eq. C)  Pressure drop = Resistance * Flow
// Basis: Fluid analog of Ohm's Law
//  Eq. D)  (Sum of flows entering junction = sum of flows leaving junction)
//       Basis: Kirchhoff Junction rule
//  Eq. E)  Pressure drop = (change in Flow/change in time)*Inertance
// Basis: Fluid analog of inductance
//  Eq. F)  Elastance = Compliance^(-1)
// Basis: Definition of elastance
//  Eq. G)  Resistance = 8*(length of tube)*viscosity/(PI * radius^4)
//              Basis: Poiseuille's Law
//  Eq. H)  Reynold's number=(mean velocity of flow)*density*radius/viscosity
//              Basis: Reynold's equation
//  Eq. I)  Flow = velocity of fluid through tube * X-section of tube
//  Eq. J)  Conductance = Resistance^(-1)
//              Basis: Definition of conductance
//-------------------------------------------------------------------------
"""


"""
Notes: In the beatwise baroreflex case, changing F_vaso still
changes the solution.
F_vaso should be in the interval [0,1].
Setting F_vaso<0.4 decrease MAP. F_vaso>=0.5 increase MAP
We set F_vaso=0 to model passive arterial pressure, without sympathetic
stimuli
"""

"""
REVISION HISTORY:
        Original Author : Maxwell Neal Date: 14/dec/06
        Revised by: BEJ Date:18may11 : update comment format
        Modified and translated to Python by Siri Kallhovd: 2012

COPYRIGHT AND REQUEST FOR ACKNOWLEDGMENT OF USE:
Copyright (C) 1999-2011 University of Washington. From the National
Simulation Resource,
Director J. B. Bassingthwaighte, Department of Bioengineering,
University of Washington, Seattle WA 98195-5061.
Academic use is unrestricted. Software may be copied so long as this
copyright notice is included.

This software was developed with support from NIH grant HL073598.
```

```
"""
```

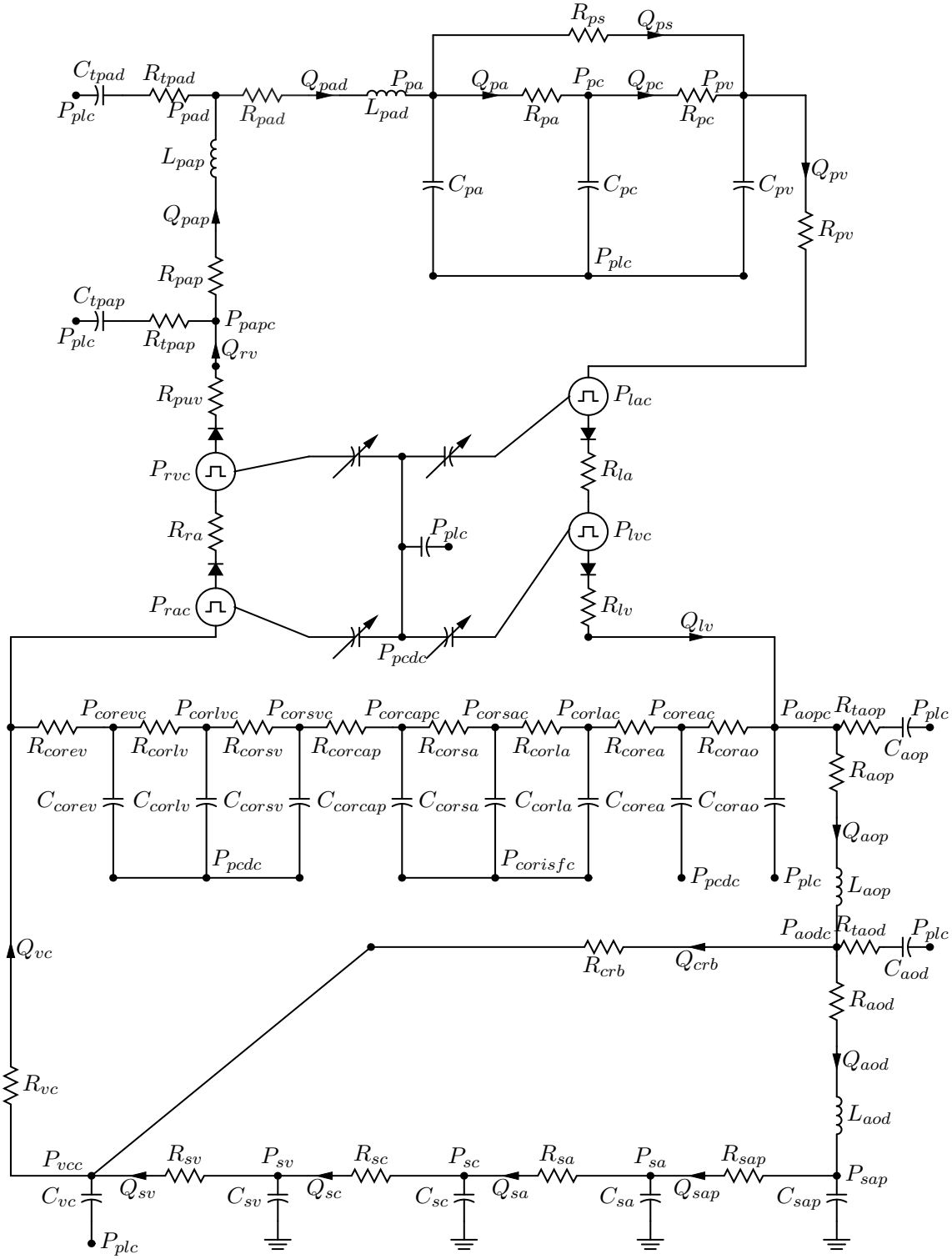


Figure A.1: Scematics of the BwC-model

References

[1] http://physiome.org/jsim/models/webmodel/NSR/Circ_with_Baroreceptors/ (2011-09-16).

[2] DC Chung, SC Niranjan, JW Clark Jr, A. Bidani, WE Johnston, JB Zwischenberger, and DL Traber. A dynamic model of ventricular interaction and pericardial influence. *American Journal of Physiology-Heart and Circulatory Physiology*, 272(6):H2942–H2962, 1997.

[3] M. Elstad, I.H. Nådland, K. Toska, and L. Walløe. Stroke volume decreases during mild dynamic and static exercise in supine humans. *Acta Physiologica*, 195(2):289–300, 2009.

[4] M. Elstad, K. Toska, and L. Walløe. Model simulations of cardiovascular changes at the onset of moderate exercise in humans. *The Journal of physiology*, 543(2):719–728, 2002.

[5] T. Heldt, E.B. Shim, R.D. Kamm, and R.G. Mark. Computational modeling of cardiovascular response to orthostatic stress. *Journal of Applied Physiology*, 92(3):1239–1254, 2002.

[6] F.C. Hoppensteadt and Peskin C.S. *Modeling and Simulation in Medicine and the Life Sciences*. Springer, 2002.

[7] J. Keener and J. Sneyd. *Mathematical Physiology*, volume System Physiology. Springer, second edition, 2009.

[8] K. Lu, JW Clark, FH Ghorbel, DL Ware, and A. Bidani. A human cardiopulmonary system model applied to the analysis of the valsalva maneuver. *American Journal of Physiology-Heart and Circulatory Physiology*, 281(6):H2661, 2001.

[9] JB Olansen, JW Clark, D. Khoury, F. Ghorbel, and A. Bidani. A closed-loop model of the canine cardiovascular system that includes ventricular interaction. *Computers and biomedical research*, 33(4):260–295, 2000.

[10] D.U. Silverthorn. *Human Physiology: an Integrated Approach*. Pearson/Benjamin Cummings, 5th, international edition, 2010.

[11] JW Spickler, P. Kezdi, and E. Geller. Transfer characteristics of the carotid sinus pressure control system. *Baroreceptors and Hypertension. Dayton, OH: Pergamon*, pages 31–40, 1967.

[12] Magnus Tølløfsrud. Modeling muscle flow during exercise. Simula, 12 2008.

[13] K. Toska and M. Eriksen. Peripheral vasoconstriction shortly after onset of moderate exercise in humans. *Journal of Applied Physiology*, 77(3):1519–1525, 1994.

[14] K. Toska, M. Eriksen, and L. Walloe. Short-term control of cardiovascular function: estimation of control parameters in healthy humans. *American Journal of Physiology-Heart and Circulatory Physiology*, 270(2):H651–H660, 1996.

[15] H.D. Young and R.A. Freedman. *Sears and Zemansky's University Physics: with Modern Physics*. Pearson/Addison-Wesley, 11th, international edition, 2004.