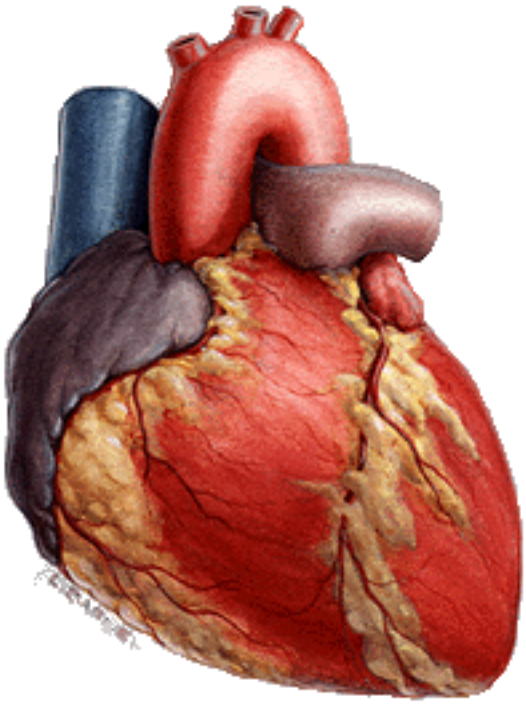
Numerical solution of a human circulation model



Tammo Jan Dijkema

July 22, 2004

Preface

This MSc Thesis describes the results of my final project, which concludes my study at the university of Utrecht. This final project was carried out at Eindhoven University, under the supervision of prof.dr. Wil Schilders.

The project, working title "Donders", originates from dr. Gerrit Jan Noordergraaf, anaesthesiologist at the Elisabeth hospital in Tilburg, responsible for the physiology and the original clinical question, and prof. dr. emeritus Abraham Noordergraaf, working at the University of Pennsylvania, who supplied the model equations.

I would especially like to thank ir. Wil Kortsmit, also a project member, for providing the initial version of the source code, and for teaching me everything I now know about Mathematica.

Also, I thank Gerrit Jan and Abraham Noordergraaf for providing me with this interesting problem, and giving me the opportunity to learn a lot about the human physiology. They even gave me the opportunity to take part of a medical training in "Advanced Cardiac Life Support" (ACLS).

Other people who assisted me with the project are drs. Alex Boer, drs. Pieter Heres, drs. Sandra Allaart–Bruin, dr. Jan ter Maten, dr. Fons van de Ven, and the CPR-Lab team of the Elisabeth hospital.

Lastly, I would like to thank my supervisors from Utrecht: dr. Gerard Sleijpen and prof.dr. Henk van der Vorst.

Contents

Chapter 1

Introduction

The function of the human heart, according to Harvey (1628)[1], is to pump blood through the body. The blood is enriched with oxygen in the lungs, which it later yields to the tissues of various parts of the body. When the heart stops, essential organs stop getting oxygen, most importantly the brain and the heart itself. A way to overcome this is to apply resuscitation. Resuscitation consists of mouth-to-mouth ventilation and thorax compressions, resulting in pressure applied to the heart. Resuscitation is not fully understood, though it has proven to be successful to a certain extent.

The purpose of this project is to create a model of the human circulation which is suitable for use in non-physiological conditions, but shows the same behaviour as a real circulatory system under healthy conditions. The second demand forces a great demand on the modelling. All parts of the model must be based on real physiological insights, instead of just a 'fit' to some desired output. The model will give us a means of validating the standard parameter values for resuscitation.

Resuscitation is generally thought to yield at most 30% of the normal cardiac output. At least 25% is necessary for the brain to survive (for a limited time). This implies that resuscitation must be performed as perfectly as possible. But what this "perfect" is, is not known. How deep should the thorax be compressed? What should the frequencies of the compressions be? And of the ventilations?

Two main ideas exist as to why resuscitation works. The explanation by the *cardiac pump* model is that by pressing on the sternum (the bone in the middle of the chest which connects the ribs), the ventricles of the heart are compressed, forcing all blood to flow out of the them. This corresponds to the origin of resuscitation: before, the chest was cut open, and the heart was

[1] Before this time, it was thought that food was converted (by the liver) to blood which was then consumed by the organs.

compressed by hand[2]. Another explanation of resuscitation is given by the *thoracic pump* model. This model assumes that the chest compressions cause an increase of pressure in the entire thorax. So also parts of the circulation other than the heart are compressed, such as the atria and venae cavae (the veins just before the pulmonary atrium). With our computer simulation, we can test both models.

It is very hard to test all parameters on human beings. A computer model provides us with a way to test all values without having the fear of someone dying needlessly under our hands.

The models are formulated as differential equations, analogous to electrical systems. This has the advantage of being easily depictable, following existing standards. Standard techniques for circuit simulation can be used, with adaptation to some very specific equations.

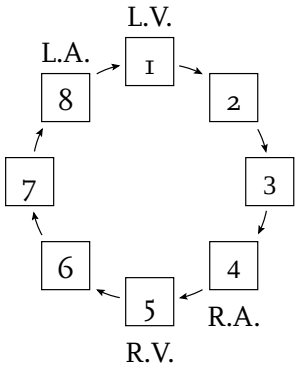
One of the project goals is to do everything from the computer package Mathematica (Wolfram 2003). The structure of this language should allow even non-mathematicians (clinicians, for example) to have an idea of what is going on in the program.

1.1 Basic physiology

The circulatory system can be subdivided into two main parts: the *pulmonary circulation* which enriches blood with oxygen, and the *systemic circulation* which distributes this oxygen amongst the organs. The heart couples these systems. Blood is transported from the heart through *arteries*, and back to the heart through *veins*.

Oxygenated blood leaves the heart through the *aorta*, the main systemic artery. This artery splits into a number of smaller arteries, which branch off into many smaller and smaller vessels, *arterioles*. These eventually empty in small capillaries, where oxygen leaves the blood and enters the tissue of organs through the thin capillary walls. In a similar way, carbon dioxide from the tissues enter the blood in the capillaries. The blood flows on through larger and larger veins. There are two main veins that empty into the heart, the *vena cava inferior* and the *vena cava superior* for blood from the lower and upper half of the body, respectively. The blood now flows into the right half of the heart. It is pumped into the pulmonary arteries, which lead it to the lungs, where the blood is oxygenated. It flows through the pulmonary veins into the left half of the heart from which it is pumped into the aorta again.

[2] This technique was good for getting the circulation going, but unfortunately most patients died of infections



1. Left ventricle
2. Systemic arteries
3. Systemic periphery
4. Systemic veins & right atrium
5. Right ventricle
6. Pulmonary arteries
7. Pulmonary periphery
8. Pulmonary veins & left atrium

Figure 1.1: Schematic representation of the circulation.

The heart itself consists of four chambers. The right side of the heart is for the pulmonary circulation, while the left part serves the systemic circulation. Both sides consist of an *atrium* and a *ventricle*. The atrium acts as a supporting pump, filling the ventricle in a short time. The ventricle is the most powerful pump. It contracts when it receives electrical signals, which happens rhythmically at a frequency of about 70 beats per minute. The atria and ventricles are separated by *A-V valves*, preventing back flow from the ventricles into the atria. These valves are called the *tricuspid* and *mitral* valve for the right and left heart, respectively. On the other side of the ventricles, the ventricles are separated from the arteries by another set of valves, called the *pulmonary* and *aortic* valve, for the right and left part. A schematic representation of the circulation is shown in figure 1.1.

1.2 Notation

Throughout this report, we will work in the *cgs*-system of units (centimeter, gram, second). Furthermore, as a unit of pressure, millimeters of mercury, mmHg, will be used, which is standard medical practice (1mmHg corresponds to approximately 133 Pascal).

A prime will denote a time derivative, so $x'(t) := \frac{\mathrm{d}}{\mathrm{d}t}x(t)$. With a superscript like $x^{[i]}$ we will denote some iterative procedure, while a superscript $f^{(i)}(t)$ will denote the i-th time derivative $\frac{\mathrm{d}^i}{\mathrm{d}t^i}f(t)$.

Many parameter values have been collected. As this represents a lot of, partially unpublished, work by G.J. Noordergraaf and A. Noordergraaf and the project team, we have chosen not to treat them extensively in this thesis.

Chapter 2

Model description

2.1 Introduction

In this chapter, two models will be defined which describe a part of the circulatory system. These models are stated in electrical terms, the correspondence with which is described in section 2.2. Two parts of the models will be described in more detail: ventricles (in section 2.3) and valves (in section 2.4). Finally, the modelling of resuscitation will be regarded.

2.2 Correspondence with electrical circuits

The model that is being developed, a lumped-parameter model, is analogous to electrical circuits. We use the correspondences in table 2.1. In the model, veins are represented as a capacitor, a resistor, and an inductor. These elements have the same behaviour as in electrical terms. There are two non-standard elements, valves (drawn as diodes) and ventricles (drawn as pairs of diamonds).

Note that the correspondence between electrical circuits and the human circulatory system is not absolute. One important observation is that while

Electrical		**Physiological**	
Term	**Symbol**	**Term**	**Symbol**
Voltage	V (V)	Pressure	p (mmHg)
Current	I (A)	Flow	Q (ml/s)
Charge	q (C)	Volume	V (ml)

Table 2.1: Correspondences between electrical and physiological terms.

electrical charge can easily become negative, volume certainly cannot. Luckily, under physiological circumstances, volumes remain positive. In simulations of resuscitation, we aborted the simulation whenever a negative volume occurred.

2.3 Ventricular contraction

Ventricles and atria are pumps. In electrical terms, they can be described as voltage sources. However, due to complicated feedback-mechanisms in the heart, they are not independent sources, and not even just 'current-defined' voltage sources. We will define the behaviour for the element RV (for right ventricle), the behaviour for the left ventricle (LV) is similar. The formula for the element RV is (Palladino and Noordergraaf 2002):

$$p_{RV}(t) = a_{RV}(V_{RV}(t) - b_{RV})^2 + (c_{RV}V_{RV}(t) - d_{RV})F_{RV}(Q_{H9}(t), t). \quad (2.1)$$

The first part of this formula, $a_{RV}(V_{RV}(t)-b_{RV})^2$ can be seen as the passive part of this pump: the generated pressure depends only on the volume, which is part of the model. A larger volume corresponds to better pumping. The second part, the active part, contains an extra term: $F_{RV}(Q_{H9}(t), t)$. This is a function of the blood flow just after the ventricle and of time. The formula for $F_{RV}(Q(t), t)$ is:

$$F_{RV}(Q(t), t) = f_{RV}(t \bmod t_h) - k_1 Q(t) + k_2 Q(t - \kappa t \bmod t_h)^2. \quad (2.2)$$

Here, another function is defined, which is totally independent of the system (it depends only on time). This function, $f_{RV}(t)$, is shown in figure 2.1. It induces the heart rhythm.

The terms with k_1 and k_2 represent a 'flow correction' on the contraction (Palladino and Noordergraaf 2002). The k_2-term represents energy which is stored in the heart muscles. This energy comes available later, when the contraction is over its maximum. Hence the delay in the time $t - \kappa t \bmod t_h$.

2.4 Valves

Cardiac valves are perhaps the most difficult part of the heart to model. Roughly speaking, their behaviour is as follows. When the pressure before the valve is greater than the pressure after, the valve opens, and flow through the valve is possible. However, when the flow tends to become negative, the valve closes because blood is dragging it into its closed position. During this closing phase,

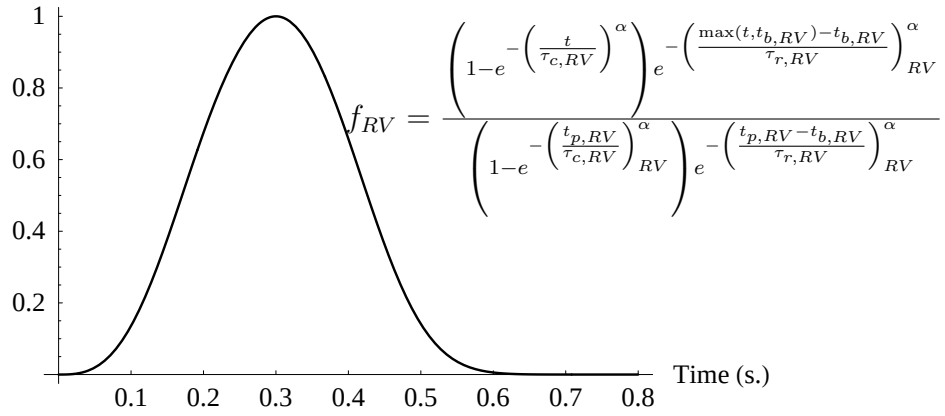


Figure 2.1: The function f_{RV}.

a not so small amount of blood is brought back through the closing valve. This is called *back flow*.

Even in the closed state, valves do not completely prevent flow between the room before and after the valve. However, we will neglect this 'leakage'.

The blood flow through the pulmonary valve, measured over one heart beat, in a real person, is shown in figure 2.2. It should be stressed that obtaining such a graph is rather hard, which is one of the reasons it is nice to have a model like the one we are developing.

A lot of computer simulation is done on the subject of valves, all using three-dimensional models (van de Vosse, de Hart, van Oijen, Bessems, Segal, Wolters, Stijnen, and Baaijens 2003). Often, finite-element methods are used to simulate the behaviour of valves. The experiments show that the behaviour of valves is indeed very complex. In our lumped-parameter models, we therefore had to simplify the behaviour of valves. Various simplifications were considered, which will be described below.

2.4.1 Valves as diodes

The most intuitive electrical analogue of a valve is a diode: it prevents current (or blood flow) to go to the opposite direction. Taking a perfect diode, blood can flow freely in one direction, and is blocked in the other direction. Combining a valve with a nearby resistance R, the flow $Q(t)$ is completely determined from the pressures before ($p_1(t)$) and after ($p_2(t)$) the valve. This blood flow is

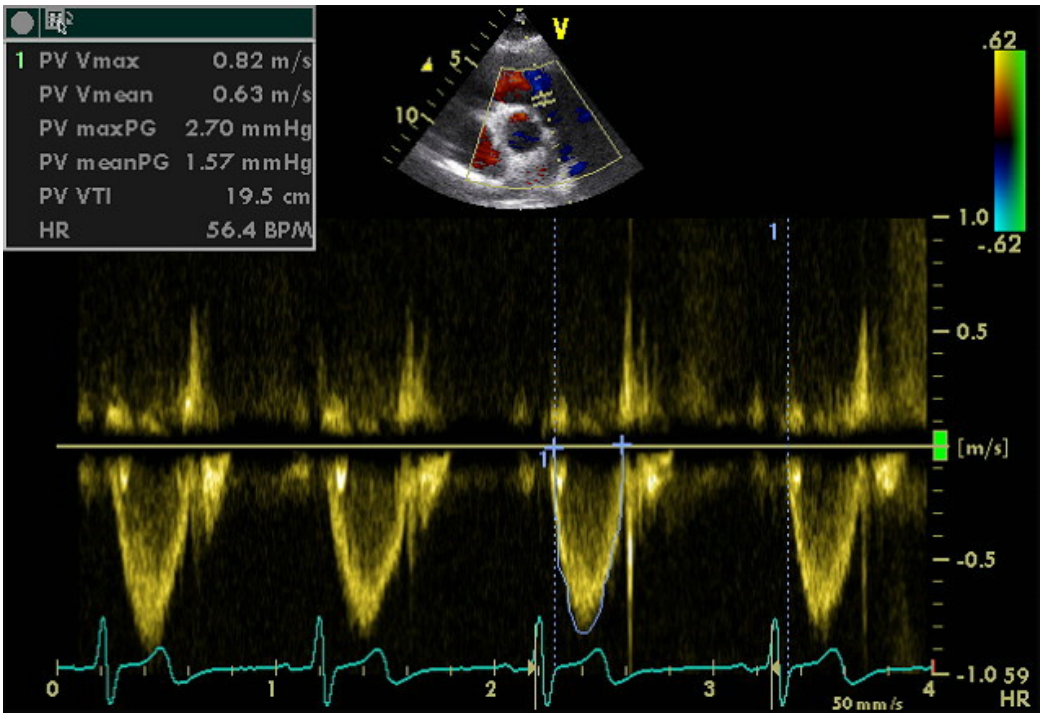


Figure 2.2: Measured data (middle) of the flow through the pulmonary valve over the period of one heart beat. Doppler echo technology was used to obtain the graph. On top, a perpendicular projection of the pulmonary valve is shown. For reference, an electrocardiogram of the patient is given in the bottom of the picture, showing the electrical signals from the heart and therefore the phase of a heart beat. The flow is measured in m/s, where in this case 1 m/s corresponds to roughly 300 ml/s.

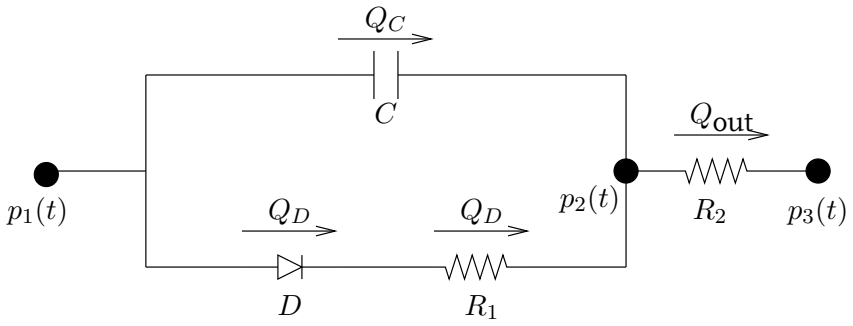


Figure 2.3: Electrical valve model

computed as

$$RQ(t) = \begin{cases} p_1(t) - p_2(t) & \text{if } p_1(t) > p_2(t) \\ 0 & \text{else.} \end{cases} \tag{2.3}$$

In electrical circuit analysis, another model for diodes is common, using a nonlinear resistance. The flow through such a diode is modelled here by the equation

$$Q(t) = e^{\alpha(p_1(t) - p_2(t))}. \tag{2.4}$$

If the voltage drop over the diode is negative, the flow will be nearly zero, while at a positive voltage drop the flow will be extremely large (approaching infinity, specially when thinking numerically).

Even when taking a more general model of a diode, where Q is a more general function of p_1 and p_2, it is only possible to simulate leakage.

2.4.2 Electrical valve model

To introduce back flow, a more general model of a valve was made, using not only a diode, like in section 2.4.1, but also a resistor and a capacitor. This model is shown in figure 2.3. Choosing parameters for the capacitor and the resistor wisely, it is possible to obtain a graph similar to the one in figure 2.2.

The diode that is drawn in this scheme represents a 'perfect' diode, allowing for no back flow. However, the total model will. The equations that describe this model are

$$\begin{aligned} Q_{\text{out}}(t) &= Q_C(t) + Q_D(t), && (2.5) \\ Q_D R_1 &= \max(p_1(t) - p_2(t), 0), && (2.6) \\ Q_C(t) &= C(p_1'(t) - p_2'(t), && (2.7) \\ R_2 Q_2(t) &= p_2(t) - p_3(t). && (2.8) \end{aligned}$$

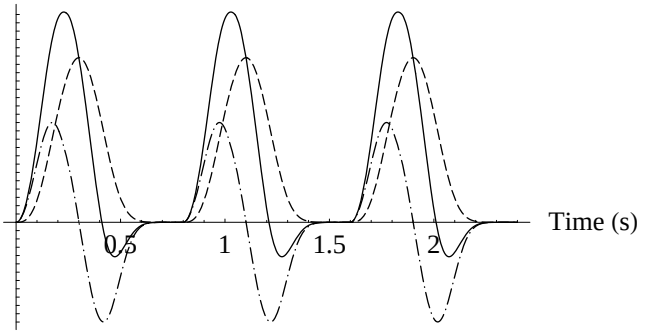


Figure 2.4: Results of an experiment with the electrical valve model. Solid is the total flow, dashed the flow through the diode, and dash-dotted the flow through the capacitor.

A simple test was done using this model, putting $p_1(t) = f_{LV}(t)$ (reasoning the voltage drop over the valve will assume this shape) and grounding $p_3(t) = 0$. The results of this experiment are shown in figure 2.4. Parameter values for this experiment were $C = 0.1$, $R_1 = 1$, $R_2 = 0.01$. The shape of the solid curve is the main result of this experiment, it resembles 2.2, with back flow. The absolute values of the flow and back flow depend on parameter values.

A physiological interpretation of the 'electrical valve' model could be the following. The capacitor C represents a small part of the aorta, which will contain all the blood that flows back into the ventricle. This part is filled (through a special hypothetical hole besides the aortic valve) during the beginning of the ejection phase (see the dash-dotted line in figure 2.4), and starts emptying somewhere halfway this filling. This emptying causes some flow back into the ventricle, but there is still a net flow into the aorta. However, the pressure drop over the aortic valve becomes negative, and for a short period of time, the flow is caused only by the emptying of the reservoir in the aorta. This phase lasts until the reservoir is empty.

Note that this is only an attempt to explain the electrical valve model in physiological terms. There is no real reservoir inside the aorta dedicated to providing back flow. Another interpretation could be treating all of the model as one electrical element, just as resistors, transistors and capacitors. The voltage-current characteristics of this new element are then given by the model.

The reason that this model was rejected is twofold. Firstly, the model does not truly represent a physiological phenomenon. More importantly, the effect of the capacitor C is present during all of the ejection phase, not just during the period of back flow (the dash-dotted line in figure 2.4). This means

that the ventricular contraction alone does not account fully for the ejection of blood: the separation between ventricular contraction and valve behaviour is gone. The solution for back flow, which takes place only a small amount of time, requires adjustments to the whole model. Of course, an *ad-hoc* solution of changing some parameter values could fix this, but this would not be in accordance with the goals of the project, which include having a physiological interpretation for everything.

2.4.3 Prescribing blood flow through the valves

Yet another way to put valves into the model is completely prescribing the blood flow in the negative direction. As soon as the blood tends to flow backward, we stop solving the differential equations, and treat the flow through the valve as a known function. This known function is only used for a very short period (the period that there actually is back flow lasts only about 5% of the time of one heart beat).

This model has one major disadvantage: instead of calculating the flow of blood, we simply prescribe it. The obtained results are of course what we want, but it was not obtained through a model.

2.4.4 Mechanical valve model

It is clear from the previous section that modeling a valve in an electrical one-dimensional model is very hard. Therefore, we include one mechanical part in the model, just for the valves. We model the valves as shown in figure 2.5, which represents a longitudinal section of the pulmonary valve, separating the right ventricle and the pulmonary arteries.

The flow trough the valves is modelled as a plug flow. A Poiseuille flow is not used here because the diameter of the artery is too small for that. This means that the velocity of blood, $v(t)$ is constant in the space between the valves. This can be related to the variables already present in the model by adding the area of the space between the valves:

$$v(t) = \frac{Q(t)}{\pi r(t)^2/4}. \tag{2.9}$$

The diameter $r(t)$ between the valves depends on the angle of the valves $\alpha(t)$ as

$$r(t) = r(0)(1 - \sin(\alpha(t))), \tag{2.10}$$

where $r(0)$ is the diameter of the space when the valves would not have been there.

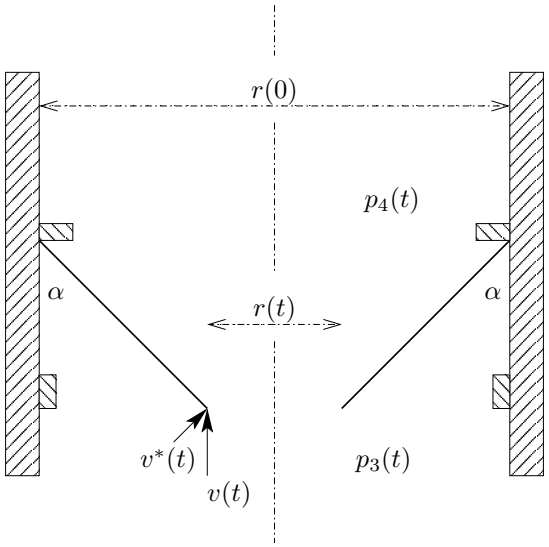


Figure 2.5: Mechanical valve model of the pulmonary arterial valve.

The velocity of the end of the valves is now set to be equal to the perpendicular part of the velocity of the blood, $v^*(t)$ (see figure 2.5). This part is computed as

$$v^*(t) = v(t)\sin(\alpha(t)). \tag{2.11}$$

Since $v^*(t)$ is the tangent velocity of a part that moves with radial velocity $\alpha'(t)$, these two variables are related as

$$v^*(t) = \ell\alpha'(t), \tag{2.12}$$

where ℓ is the length of a valve. Clearly, $\ell = r(0)/2$.

As indicated in the figure, $\alpha(t)$ is bounded between $90°$ (valve completely closed) and $5°$ (valve open). The lower bound on $\alpha(t)$ is clearly helpful, since if the valves would have been allowed to open fully, $\alpha(t) = 0°$, they would never close again (eq. 2.12). However, it is also a physiological phenomenon that valves do not open completely, or else they would stick to the walls of the arteries.

We have adopted this "mechanical" valve model as the most realistic one, and have used this for doing the simulations.

2.5 Resuscitation

When the hearts stops pumping, for whatever reason, blood stops flowing through the body. This stops the distribution of oxygen to the organs. This

is fatal to the brain first: when it's oxygen supply is below 25% of natural, it suffers irreparable damage. To prevent this damage, CPR (cardiopulmonary resuscitation) is carried out.

The basic part of CPR consists of two parts: oxygen is blown into the victims lungs, and a minimal circulation is achieved by rhythmically compressing the chest. There are standard guidelines for the procedure: 15 chest compressions (at a speed of 100 compressions per minute) should be alternated with two ventilations. We will focus on the chest compressions, and study the effect of varying the force and speed on the cardiac output.

The idea of chest compressions is that the intrathoracic pressure $p_e(t)$. This is the pressure external to the heart, so it will have an effect on the heart. In our simulations, a dysfunctional heart is created by setting the terms $F_{LV}(t)$ and $F_{RV}(t)$ to zero. The active part of the ventricular contractions is then replaced by a pressure source $p_e(t)$, which is a periodic function (see figure 2.6) with period t_h,

$$p_e(t \bmod t_h) = \begin{cases} \frac{p_{\text{emax}}}{2}(1 + \sin(6\pi t/t_h - \pi/2) & \text{if } t < \frac{2}{3}t_h, \\ 0 & \text{else.} \end{cases} \tag{2.13}$$

The maximum compression pressure p_{emax} and the duration of the compression t_h will be varied. The effect on the *stroke volume* will be regarded, which is defined as the total amount of blood which leaves the heart at one heart beat. A normal stroke volume is around 60–80 ml. During exercise this can grow up to a factor 4 bigger. The *cardiac output* is defined as the blood that leaves the heart in a minute, so it can be computed by multiplying the stroke volume with the heart rate. In resuscitation, the cardiac output is a more useful concept than the stroke volume, since there is no heart beat left.

The ventricles are not the only parts of the circulation being compressed by chest compressions. For example, the pulmonary veins also lie within the thorax. It is only natural to assume that they are also compressed by CPR. Therefore, we will also add the pressure source to them.

2.6 Model A

Model A contains a representation of blocks 1, 2 and 3 of figure 1.1. As such, the left atrium, the left ventricle, and the systemic arteries are present. The circulation is not closed in this model. The atrial pressure $p_v(t)$ is fixed (a constant pressure / voltage source), and the pressure after the systemic arteries is set to ground (zero). One could visualise this as a large vessel of blood, which is flowing into the ventricles, and flows away freely after the systemic circulation.

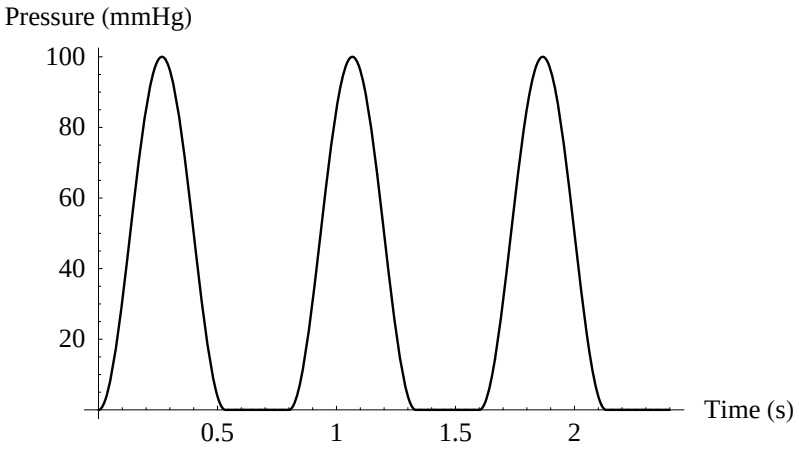


Figure 2.6: The intrathoracic pressure causes by resuscitation, when we assume the maximum pressure reached is 100mmHg.

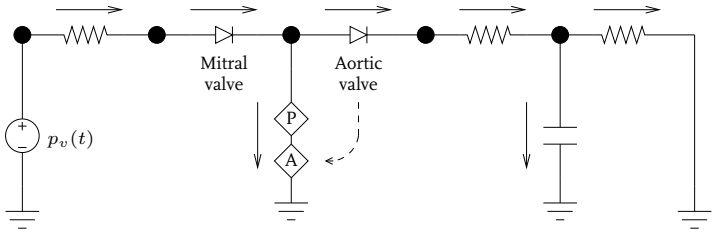


Figure 2.7: Schematical representation of model A.

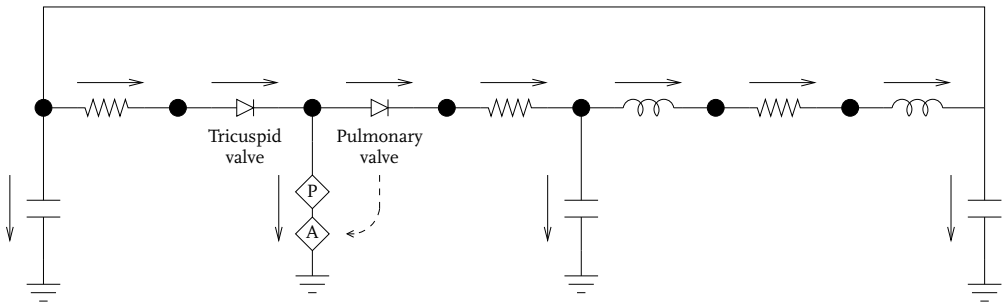


Figure 2.8: Schematical representation of model B2.

The equations that govern model A are

$$\begin{aligned}
p_6(t) - p_e(t) &= a_{LV}\left(V_{LV}(t) - b_{LV}\right)^2 + \\
&\quad \left(c_{LV}V_{LV}(t) - d_{LV}\right) F_{LV}(Q_{H15}(t), t), && (2.14)\\
R_S Q_{H1}(t) &= p_p(t), && (2.15)\\
Q_{H12}(t) &= Q_{H15}(t) + Q_{V9}(t), && (2.16)\\
Q_{V10}(t) &= Q_{H15}(t) - Q_{H1}(t), && (2.17)\\
Q_{V10}(t) &= C_{10}p_p'(t), && (2.18)\\
V_{LV}'(t) &= Q_{H10}(t), && (2.19)\\
R_{LV}Q_{H12}(t) &= (p_v(t) - p_6(t))r_{H12}(t)^2/r_{H12}(0)^2 && (2.20)\\
v_{H12}(t) &= -Q_{H12}(t)/r_{H12}(t), && (2.21)\\
r_{H12}(t) &= r_{H12}(0)(1 - \sin(\alpha_{H12}(t))), && (2.22)\\
\alpha_{H12}'(t) &= 2v_{H12}(t)\sin(\alpha_{H12}(t))/r_{H12}(0), && (2.23)\\
Z_{OS}Q_{H15}(t) &= (p_6(t) - p_p(t))r_{H15}(t)^2/r_{H15}(0)^2 && (2.24)\\
v_{H15}(t) &= -Q_{H15}(t)/r_{H15}(t), && (2.25)\\
r_{H15}(t) &= r_{H15}(0)(1 - \sin(\alpha_{H15}(t))), && (2.26)\\
\alpha_{H15}'(t) &= 2v_{H15}(t)\sin(\alpha_{H15}(t))/r_{H15}(0). && (2.27)
\end{aligned}$$

Care is to be taken to keep $\alpha_{H12}(t)$ and α_{H15} between the bounds of 5° and 90°: the differential equation for $\alpha_{H12}(t)$ and $\alpha_{H15}(t)$ are replaced by $\alpha_{H12}'(t) = 0$ and $\alpha_{H15}'(t) = 0$ upon hitting such a bound.

2.7 Model B2

Model B2 is a representation of the pulmonary circulation, to be specific of blocks 4–7 of figure 1.1. The model rather resembles model A, with different

parameter values. In addition, two capacitors have been added, representing the storage capacity of the right atrium and the systemic veins.

The main difference between model B2 and A is that the system has been closed. This is achieved in a rather artificial way: the pulmonary periphery is connected to the systemic (instead of the pulmonary) veins, thus bypassing the left atrium and ventricle completely. The electrical circuit representation of model B2 is given in figure 2.8.

The equations governing the behavior of the right ventricle are similar to those for the left ventricle (2.1), but with different parameter values. Particularly, the right ventricle is much weaker than the left, which is logical considering the pressures in the pulmonary circulation to be a lot lower.

The equations of model B2 are

$$
\begin{aligned}
p_2(t) - p_e(t) &= a_{RV}\left(V_{RV}(t) - b_{RV}\right)^2 + \\
&\quad \left(c_{RV}V_{RV}(t) - d_{RV}\right) F_{RV}(Q_{H9}(t), t), && (2.28)\\
C_6 p_4'(t) &= Q_{V6}(t), && (2.29)\\
L_{60}Q_{H10}'(t) &= p_4(t) - p_4'(t) && (2.30)\\
L_{30}Q_{H10}'(t) &= p_{300}^*(t) - p_1(t) && (2.31)\\
p_4^*(t) - p_{300}(t) &= R_P Q_{H10}(t) && (2.32)\\
Q_{H6}(t) &= Q_{H9}(t) + Q_{V5}(t) && (2.33)\\
Q_{H9}(t) &= Q_{V6}(t) + Q_{H10}(t) && (2.34)\\
Q_{H10}(t) &= Q_{V3}(t) + Q_{V4}(t) + Q_{H6}(t) && (2.35)\\
C_3 p_1'(t) &= Q_{V3}(t) && (2.36)\\
C_4 p_1'(t) &= Q_{V4}(t) && (2.37)\\
Q_{V5}(t) &= V_{RV}'(t) && (2.38)\\
R_{RV}Q_{H6}(t) &= (p_1(t) - p_2(t)) r_{H6}(t)^2 / r_{H6}(0)^2 && (2.39)\\
v_{H6}(t) &= -Q_{H6}(t)/r_{H6}(t), && (2.40)\\
r_{H6}(t) &= r_{H6}(0)(1 - \sin(\alpha_{H6}(t))), && (2.41)\\
\alpha_{H6}'(t) &= 2v_{H6}(t)\sin(\alpha_{H6}(t))/r_{H6}(0), && (2.42)\\
Z_{OP}Q_{H9}(t) &= (p_2(t) - p_4(t)) r_{H9}(t)^2 / r_{H9}(0)^2 && (2.43)\\
v_{H9}(t) &= -Q_{H9}(t)/r_{H9}(t), && (2.44)\\
r_{H9}(t) &= r_{H9}(0)(1 - \sin(\alpha_{H9}(t))), && (2.45)\\
\alpha_{H9}'(t) &= 2v_{H9}(t)\sin(\alpha_{H9}(t))/r_{H9}(0). && (2.46)
\end{aligned}
$$

Again, care is taken to keep $\alpha_{H6}(t)$ and $\alpha_{H9}(t)$ between their bounds, replacing equations 2.42 by $\alpha_{H6}'(t) = 0$ and $\alpha_{H9}'(t) = 0$, respectively.

Chapter 3

Numerical methods for differential-algebraic equations

3.1 Introduction

In this chapter, we will introduce the numerical methods that are used to solve a differential-algebraic equation (DAE). First, in section 3.2, backward difference formulas will be introduced for ordinary differential equations. In section 3.5, the concept of differential-algebraic equations will be introduced and the concept of index will be treated. This chapter will be concluded with a section about DASSL, a numerical package which uses backward difference formulas to solve index-1 DAE's. This theory will be applied to the circulation-specific systems in chapter 5.

3.2 Backward Difference Formulas

An ordinary differential equation (ODE) is given as

$$x'(t) = f(x(t), t) \tag{3.1}$$

where $x(t)$ may of course be a vector. The solution of this equation depends on an initial value, $x(t = 0)$.

Assuming an analytical solution of (3.1) is hard or impossible to find—which is often the case—an approximation is sought for $x(t)$. This can be done by an approximation to the derivative, for example by

$$x'(t_n) \approx \frac{x_n - x_{n-1}}{h}, \tag{3.2}$$

where x_n and x_{n-1} are approximations on times $t_n = n \cdot h$ and $t_{n-1} = (n-1) \cdot h$. Here, $h > 0$ is called the *step size*. Combining these formulas with 3.1 leads to the following equations for the successive approximations to the x_n:

$$x_n = x_{n-1} + hf(x_n, t_n). \tag{3.3}$$

This method is known as Backward Euler, or Implicit Euler. It is implicit, because x_n is not known explicitly in terms of values in the past. Some kind of numerical method, generally Newton-Raphson (see 3.3), has to be used to solve x_n from (3.3).

Euler Backward is part of a larger family of methods, known as *Backward Differentiation Formulas*, or BDFs. A general k-th order backward difference formula uses the following approximation formula:

$$x'(t_n) \approx \frac{1}{h} \sum_{i=0}^{k} \alpha_i x_{n-i}, \tag{3.4}$$

where α_i, $i = 0, 1, \ldots k$ are the coefficients of the BDF method. This approximation can be viewed as taking a k-th order polynomial, fitted through the points $x_{n-k}, \ldots x_n$ and evaluating its derivative at t_n. For the first order method, the coefficients are $\alpha_0 = -1$, $\alpha_1 = 1$.

It is clear that in order for the BDF-formula to be good, the solution should be smooth, so that the fitting polynomial is in fact a good approximation to the solution.

3.2.1 Consistency

Most numerical methods are based on replacing the derivative $x'(t)$ by some approximation formula, for example 3.4. The local truncation error (LTE) is the error that is made by using this approximation, if all the previous solutions had been correct. More precisely, assume that that $x_k = x(t_k)$, for all $k < n$, i.e. no error has been made in previous calculations. Then the local truncation error is

$$lte = \frac{x(t_n) - x_n}{h}. \tag{3.5}$$

The local truncation error of the implicit Euler method (3.3) can be computed easily. It is based on the Taylor expansion

$$x(t_{n-1}) = x(t_n) - hx'(t_n) + \frac{1}{2}h^2 x''(\xi_1), \qquad t_{n-1} \leq \xi \leq t_n. \tag{3.6}$$

For the approximated solution, we have $x_n = x_{n-1} + hf(t_n)$, or

$$x_{n-1} = x_n - hf(t_n) \tag{3.7}$$

If we assume that all the solutions up to n are correct, it holds that $x(t_{n-1}) = x_{n-1}$ and $x'(t_{n-1}) = f(t_{n-1})$. Subtracting (3.7) from (3.6) yields

$$x(t_n) - x_n = \frac{1}{2}h^2 x''(\xi). \tag{3.8}$$

We see that the local truncation error of the implicit Euler method is of order h.

The coefficients of the k-th order backward difference methods are chosen such that the local truncation error is of order h^k. If $h \to 0$, so does the local truncation error (if we take the order $k > 0$). The method is called *consistent*.

However, we are much more interested in the global error $e(t)$, which is defined by

$$e(t) = \|x(t_n) - x_n\|, \tag{3.9}$$

where no assumption about previous solutions has been made. An estimate of the global error can be obtained from an estimate of the LTE, whenever the method is *stable*. This concept is explained in the next section.

3.2.2 Stability

The stability (see for instance Lambert (1991) for much more detailed theory) of numerical methods is often defined for onedimensional the linear test problem

$$x'(t) = \lambda t, \tag{3.10}$$
$$x(0) = x_0, \tag{3.11}$$

where $\lambda \in \mathbf{C}$. The general solution of 3.10 is

$$x(t) = x_0 e^{\lambda t}. \tag{3.12}$$

A numerical method is called *stable* for a certain value of $h\lambda$ (where h is the step size) if the error remains bounded for $t \to \infty$.

The first order BDF is applied to this differential equation yields

$$x_{n+1} = x_n + h\lambda x_{n+1}. \tag{3.13}$$

This solution remains bounded if and only if

$$\frac{1}{|1 - h\lambda|} \leq 1. \tag{3.14}$$

So, if $|1 - h\lambda| > 1$, the implicit Euler method is stable. The *stability region* is the set of all $h\lambda$ such that the solution of 3.10 remains bounded. Clearly, the

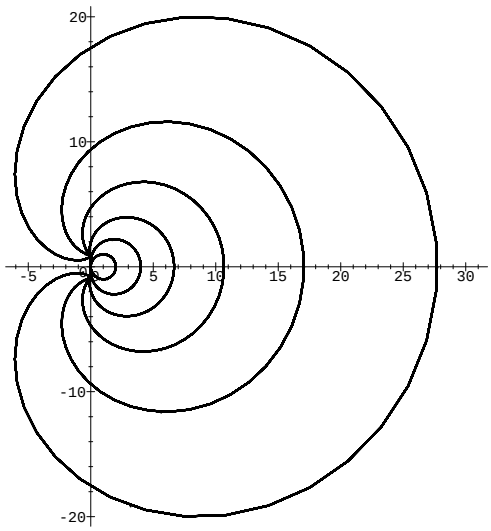


Figure 3.1: The stability region of BDF-methods up to order 6. The stability region is the region outside the closed contours. The contour for region 1 is the smallest circle, and the contours are growing with the order.

stability region for Euler backward is the area outside the circle with radius 1 around 1, in the complex plane.

A method is called A-*stable* if the stability region includes all of the negative half-plane $\{h\lambda \in \mathbf{C} : \mathrm{Re}(h\lambda) \leq 0\}$. It is clear that the implicit Euler method is A-stable. $A(\alpha)$-stability of a method means that the method is stable for the set $\{h\lambda \in \mathbf{C} : -\pi + \alpha \leq \arg(h\lambda) \leq \pi - \alpha\}$. Note that $A(\pi/2)$-stability is the same as A-stability. BDF-formulas up to order 6 are $A(\alpha)$-stable, for some $\alpha \in (0, \pi/2)$, while orders 1 and 2 are even A-stable.

For backward difference formulas of order up to 6, the stability regions are shown in figure 3.1 (Bruin 2001). For orders greater than 6, backward difference formulas are instable.

3.3 Newton-Raphson

The Newton-Raphson method is a method for finding roots to a vector-valued function $F : \mathbf{R}^n \to \mathbf{R}^n$, with $n > 1$. Supposing that ξ is a zero of F, and that $x^{[0]}$ is an approximation to ξ, $F(\xi)$ can be approximated by its first order Taylor expansion around $x^{[0]} = (x_1^{[0]}, \ldots, x_0^{[0]})$:

$$0 = F(\xi) \approx F(x^{[0]}) + J(x^{[0]})(\xi - x^{[0]}), \tag{3.15}$$

where

$$J(x) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial F_n}{\partial x_1} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix} \tag{3.16}$$

is the Jacobian of $F(x)$. We can now use equation 3.15 to obtain a new estimate $x^{[1]}$ for ξ, solving

$$x^{[1]} = x^{[0]} - (J(x^{[0]}))^{-1} F(x^{[0]}), \tag{3.17}$$

where the inversion of the matrix $J(x^{[0]})$ can be done by some direct ore iterative matrix solver. $x^{[1]}$ is again an approximation to ξ (which we hope is better), and we may obtain another approximation by using the iteration

$$x^{[i+1]} = x^{[i]} - (J(x^{[i]}))^{-1} F(x^{[i]}). \tag{3.18}$$

Provided that the initial guess $x^{[0]}$ is good enough, this method converges to ξ, and does so quadratically (Press, Teukolsky, Vetterling, and Flannery 1992).

3.4 Predictor-corrector methods

To use backward difference methods, it is essential that the Newton-Raphson method converges quickly. To achieve this, it is essential that the initial guess $x_{n+1}^{[0]}$ is accurate. A good estimate for x_{n+1} could be the value x_n. However, if the solution changes rapidly, a large error is made. Another way to get a good initial guess is by using a predictor method.

An example of a predictor method could be the Euler Forward method: for x_{n+1}:

$$x_{n+1}^{[0]} = x_n + hf(x_n, t_n), \tag{3.19}$$

where $f(x(t), t)$ is the function from the ODE we are solving (equation 3.1). This method is often worse than Euler Backward, but suits well for just an initial guess. This predictor is used to create an initial guess for the following implicit solver, for example a BDF-method.

The scheme that is described above is actually the most basic amongst predictor-corrector methods: the predictor is used only once, after which the corrector takes over completely. The consistency of such a method is inherited only from the corrector method.

A more advanced and more stable predictor method consists of interpolating the solution x_n at previous time steps and extrapolating this solution to t_{n+1}. An extra advantage of this method is that the extrapolated polynomial also can be used for estimating the local truncation error at time t_{n+1}.

The local truncation error at point t_n is often of the form $hf^{(k)}(\xi)$, where $t_n \leq \xi \leq t_{n+1}$. If an estimate for $f^{(k)}(t_{n+1})$ is known, an h can be chosen to make the local truncation error as small as desired, or to make the step size as big as possible to gain efficiency without losing information. Methods that use this are called variable step size methods. Due to the variable step size, the coefficients for BDF method change, and the analysis becomes a lot more tedious.

A way to handle changing step sizes and orders is to keep track of the *Nordsieck vector* N. This vector contains approximations to various derivatives of the solution x at $t = t_n$:

$$N_{n,i} = x^{(i)}(t_n). \tag{3.20}$$

From this vector, all information about historical values can be stored (and retrieved, using Taylor expansions). When the step size or order changes, the only thing that needs to be done is to update the Nordsieck vector (Hairer and Wanner 1997).

3.5 Differential Algebraic Equations

The systems we had to solve were not of the form (3.1), since also algebraic equations (following from Kirchhoff's law) are involved. This makes the system a differential-algebraic equation (DAE). In the most general form, differential-algebraic equations are given as

$$f(x(t), x'(t), t) = 0. \tag{3.21}$$

Also here, an initial value must be supplied. However, the choice of initial values is much harder than in the case of ODEs. The initial values must be *consistent* with the DAE, meaning that all algebraic equations in the equation must be satisfied. Therefore, finding an initial value can be a problem.

Frequently, differential-algebraic equation can be written in the following form:

$$x'(t) \quad = \quad F(x(t), y(t), t) \tag{3.22}$$

$$0 = G(x(t), y(t), t). \tag{3.23}$$

This is called the *semi-explicit* form. The vector of unknowns is split into two parts, $x(t)$ and $y(t)$. The variables in $x(t)$ are called the *differential variables*, and the ones in $y(t)$ are called *algebraic variables*. When specifying initial conditions, it should be enough to supply only initial conditions for the differential variables, since the algebraic variables should follow from them (by the Implicit Function Theorem).

If it is possible to invert $G(x(t), y(t), t)$ with respect to $y(t)$, we can write the system as an ordinary differential equation:

$$x'(t) = F(x(t), G^{-1}(x(t)), t). \tag{3.24}$$

Calculating initial values for this case is feasible: given some initial values for $x(0)$, the values for $y(0)$ can be readily calculated from $y(0) = G^{-1}(x(0), 0)$.

Even if this rewriting to an ODE is possible, it is sometimes not desirable. In rewriting the system, sparsity patterns will probably be lost (which is a problem in large systems, but not in the systems we are solving), as will the direct connection between the system and the equations.

3.5.1 Differential index

Consider the general DAE $f(x(t), x'(t)) = 0$. The differential index m of this system is defined as the smallest number of differentiations such that the system of equations

$$\begin{aligned} f(t, x(t), x'(t)) &= 0, \\ \frac{\mathrm{d}}{\mathrm{d}t} f(t, x(t), x'(t)) &= 0, \\ &\vdots \\ \frac{\mathrm{d}^m}{\mathrm{d}t^m} f(t, x(t), x'(t)) &= 0 \end{aligned} \tag{3.25}$$

uniquely determines the explicit ODE $x'(t) = g(x(t), t)$. From this definition, it is clear that the differential index of an ODE is zero. Note that this method of converting a DAE to an ODE is not a good method for solving it, it is merely a theoretical concept.

The index can be easily determined for the semi-explicit DAE

$$x'(t) = F(x(t), y(t), t) \tag{3.26}$$

$$0 = G(x(t), y(t), t). \tag{3.27}$$

Differentiating the algebraic equation with respect to t gives

$$0 = G_x x'(t) + G_y y'(t) + G_t. \tag{3.28}$$

If we substitute 3.26 for $x'(t)$, this formula yields

$$x'(t) = F\left(x(t), y(t), t\right), \tag{3.29}$$

$$y'(t) = -\left(G_y\right)^{-1}\left(G_x F(x_1(t), x_2(t), t) + G_t\right). \tag{3.30}$$

This is an ordinary differential equation. Since one time derivative was used, we see that the differential index of a semi-explicit system is 1, provided that $G_y := \frac{\partial G}{\partial y}$ is invertible (Tischendorf 1996). If the DAE stems from a linear electrical circuit, this condition can be shown (Brenan, Campbell, and Petzold 1989) to be equivalent to two conditions on the topology of the circuit: the circuit should contain no closed loops of only voltage sources and capacitors, and no cutsets[1] of only inductances and current sources.

3.5.2 A linear example

Consider the following form of (3.21) (see (Brenan, Campbell, and Petzold 1989)):

$$Ay'(t) + By(t) = f(t) \tag{3.31}$$

$$y(0) = y_0, \tag{3.32}$$

where $y(t) \in \mathbf{R}^n$, and A and B are time-independent matrices of size $n \times n$. If A is invertible, we can write $y'(t) = -A^{-1}By(t)$, and are in fact dealing with an ODE. We will assume A to be singular, so that the differential index of this system will be ≥ 1 ($y'(t)$ cannot be solved from the system without differentiation).

If it does not hold that $\lambda A + B$ is singular for *all* $\lambda \in \mathbf{C}$, then the system is *solvable*, i.e. it possesses a unique solution for every consistent initial value y_0, and we can apply the following transformation:

$$PAQy'(t) + PBQy = Pf(t), \tag{3.33}$$

and choose P and Q such that the following equalities hold:

$$PAQ = \begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix}, \qquad PBQ = \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix}, \tag{3.34}$$

[1] A cutset is a set of branches from the circuit which causes the system to become disconnected if removed and does not if any branch from the set is left in place.

where I is an identity matrix, and N is a matrix of nilpotency k (the nilpotency of a matrix is defined as the integer i such that $N^i = 0$ and $N^{i-1} \neq 0$). If $N = 0$, we define $k = 1$.

In the special case that A is nonsingular, we can take $PAQ = I$, $PBQ = C$ and define $k = 0$. The nilpotency degree k is the same as the differential index of the DAE.

If we apply the coordinate changes P and Q to system 3.21, using 3.34 we see that it takes the form

$$\begin{aligned} y_1'(t) + Cy_1(t) &= f_1(t) && (3.35)\\ Ny_2'(t) + y_2(t) &= f_2(t), && (3.36) \end{aligned}$$

where we have split the vectors $y(t)$ and $f(t)$ into two parts. The initial value of this system is given by $(y_1(0), y_2(0))^T = Q^{-1}y_0$. Equation 3.35 is just an ODE, so it gives a solution for any function $f_1(t)$ and initial value for y_1. The second equation, (3.36), can be written as

$$(ND + I)y_2(t) = f_2(t), \tag{3.37}$$

where D is the differentiation operator. This is not an ODE. However, we can obtain a solution for it by differentiation:

$$\begin{aligned} y_2(t) &= f_2(t) - Ny_2'(t)\\ &= f_2(t) - Nf_2'(t) + N^2 f_2''(t)\\ &\vdots\\ &= f_2(t) - Nf_2'(t) + \cdots + (-1)^{k-1}N^{k-1}f_2^{(k-1)}(t) + (-1)^k N^k f_2^{(k)}(t). \end{aligned}$$

But since $N^k = 0$, the solution of (3.36) can be written as

$$y_2(t) = \sum_{i=0}^{k-1} (-1)^i N^i f_2^{(i)}(t). \tag{3.38}$$

From this last solution, we can make some useful observations. Firstly, the initial value y_0 must be consistent, so it must also be a solution of 3.38 with $t = 0$. Second, the solution of a DAE can involve higher derivatives of the forcing function $f(t)$. The solution process for $y_2(t)$ does not force $y_2(t)$ to be differentiable, so generally it need not be.

3.6 The numerical method inside Mathematica

Since one of the project terms was to work in *Mathematica*, we were somewhat limited in choosing the numerical method. Specifically, for the solution

of DAE's, Mathematica includes a special link to the package IDA, part of SUNDIALS (Hindmarsh, Brown, Grant, Lee, Serban, Shumaker, and Woodward 2003). Mathematica interfaces this program in a very nice way, requiring no special programming knowledge of the user.

IDA is in fact a re-implementation of the algorithm DASPK, which is in turn an extension to DASSL (Brenan, Campbell, and Petzold 1989). The main difference between DASPK and DASSL is that the first can use a Krylov subspace method to solve the matrix-vector equation in the Newton algorithm, while the latter uses a direct method (another difference is that DASSL was written in FORTRAN, and DASPK in C). Since the systems we are using are not very large (smaller than 100×100), we did not use an iterative method.

DASSL is a quite advanced algorithm, and we will not go into all details. However, some highlights are certainly worth mentioning (Petzold 1983). In the algorithm, a variable step size, variable order backward difference formula is used at each step, where the order $k \leq 5$ is derived from the local truncation error.

The local truncation error is estimated by taking an interpolating polynomial through the last $k+1$ computed points, and evaluating this polynomial at t_n. Also, the step size is variable, using techniques mentioned in section 3.4.

The backward difference formula is substituted in the DAE, yielding

$$F(t_n, y_n, \frac{1}{h_n} \sum_{i=1}^{k} \alpha_{n,i} y_{n-i}) = 0. \tag{3.39}$$

Note that the coefficients $\alpha_{n,i}$ of the BDF are not constant anymore, because the step size may vary. The formula above is solved by a Newton-Raphson-like scheme, using as an approximation to the Jacobian

$$J = \frac{\partial F}{\partial y} + \frac{\alpha_0}{h_n} \frac{\partial F}{\partial y'}. \tag{3.40}$$

Since the computation of the Jacobian is the most expensive (in operations) part of the total computation, the Jacobian is reused several time steps. This will have the effect that the Newton-procedure may converge slower (linearly) or not at all. However, if it converges too slowly, the Jacobian is recomputed.

Chapter 4

Other relevant numerical methods

4.1 Introduction

This chapter will be about three other numerical phenomena that are applicable to the systems of Chapter 2: delay differential equations, waveform relaxation and vector extrapolation. We will touch these subjects only very briefly, since we will not need them so much in the treatment of the models from Chapter 2. However, they could be very useful for more advanced future models.

4.2 Delay differential equations

Delay (or retarded) differential equations (DDEs) are an extension to ordinary differential equations because they allow the dependence on the solution at a time in the past. They are written in the form

$$y'(t) = f\left(t, y(t), y(t-\tau(t))\right), \quad \text{for } t \geq t_0, \tag{4.1}$$

$$y(t) = \varphi(t), \quad \text{for } t < t_0. \tag{4.2}$$

where $\tau(t)$ is a given function of time, with $\tau(t) \leq t$ for all t. (We will restrict ourselves to only one delay function.) Various aspects distinguish delay differential equations (DDEs) from ordinary differential equations. The most obvious one is that in order to compute the solution of a DDE, apart from the initial conditions, the solution $y(t)$ must be known at a certain period in the history, $[-\min_{t\geq 0}\tau(t), 0]$.

Another important difference lies in the propagation of discontinuities. In most cases, there is at least one discontinuity, since generally the prescribed historical solution is not the same *in all derivatives* at $t = 0$. Since there is now

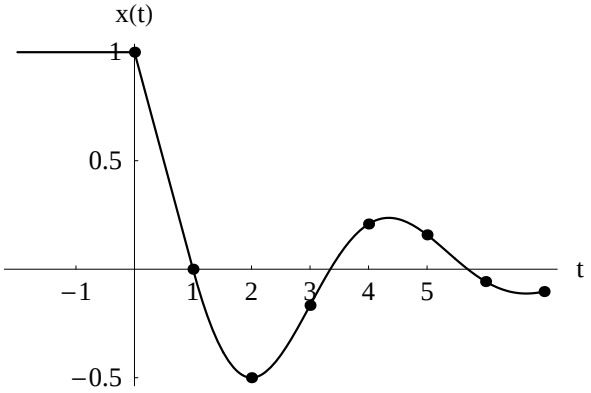


Figure 4.1: Solution of the delay differential equation 4.3. It can be seen that the solution smoothens as t grows. The discontinuity at $t = 0$ is in $x'(t)$, the discontinuity at $t = 1$ is in $x''(t)$, and so forth.

a discontinuity (in some derivative of y) at $t = 0$, there also is one at the point t_1 with $t_1 - \tau(t_1) = 0$. And since there is a discontinuity at t_1, there will also be one at the time t_2 where $t_2 - \tau(t_2) = t_1$.

As an example of this, consider the very simple delay differential equation:

$$\begin{aligned} x'(t) &= -x(t-1) && \text{for } t \geq 0 \\ x(t) &= 1 && \text{for } -1 \leq t < 0. \end{aligned} \tag{4.3}$$

For $0 \leq t < 1$, the solution is clearly given by $x(t) = 1 - t$, which yields a discontinuity in the first derivative at $t = 0$. Advancing the solution, we see that the solution for $1 \leq t < 2$, the solution is $x(t) = t - t^2/2 - 1/2$. Now, the discontinuity at $t = 1$ lies in the second derivative (See figure 4.1.)

Returning to the general case, if we suppose that $\tau(t)$ is a smooth function, this propagation of discontinuities will continue forever. However, at each step, the discontinuity will be present in a higher derivative. If the function has a so-called "fading memory" (i.e, $\tau(t) \to \infty$ as $t \to \infty$), the solution becomes smoother as $t \to \infty$.

4.2.1 Standard approach

The example in (4.3) has been solved using a standard approach for DDEs (used by Hairer and Wanner (1997)). For $0 \leq t < 1$, the function $x(t-1)$ can be treated as a given function, since the values of x at $t < 0$ are known. The solution on $0 \leq t < 1$ can be computed using any method suitable for ODEs. But now that the solution is known at $0 \leq t < 1$ is known, we can use this to solve the equation at $1 \leq t < 2$. This method can be repeated ad infinitum.

We have to pay special attention here to the observation that the solution $x(t)$ is computed in each time interval at a set of points t_n. In the next interval, the solution $x(t^*)$ can be needed, where $t_n < t^* < t_{n+1}$ for a certain n. We should use interpolation to get the approximated value of $x(t^*)$. Care may be taken that the error introduced by the interpolation is smaller than the error of the numerical method. To guarantee this, we may impose a maximum step size.

Returning to a more general case (4.1), this method can be used when $\tau(t)$ is bounded from below at some constant c. The periods during which an ODE-solver can be used are then of length c. A more advanced algorithm would be to compute for each time interval $[t_i, t_i + \Delta t]$, to find an optimal value for Δt such that $\tau(t) \leq t_i$ for all $t_i < t < t_i + \Delta t$. If $t - \tau(t)$ is strictly increasing, this would be a simple root finding problem, solving $\tau(t_i + \Delta t) - t_i = 0$.

The approach which is described above is implemented for Mathematica by Allan Hayes (Hayes 2004). His implementation only works for $\tau(t)$ constant, but it can work with multiple delays $\tau_1(t) = c_1$, $\tau_2(t) = c_2$, ...

4.2.2 Method of steps

Another possible approach to solve DDEs using solution methods for ordinary differential equations is the method of steps (Bellen and Zennaro 2003). Let's consider 4.1, with the assumption that $\tau(t) > \tau_{\min}$ for all t. Also, assume that $\alpha(t) := t - \tau(t)$ is strictly increasing. As in the approach above, we solve interval by interval. The first interval, $[t_0, t_1]$, is found by solving $\alpha(t_1) = t_0$. On this interval an ODE solver can be used to solve

$$\begin{aligned} y'(t) &= f(t, y(t), \varphi(\alpha(t))) && (4.4)\\ y(t_0) &= \varphi(t_0). && (4.5) \end{aligned}$$

In the second interval, $[t_1, t_2]$, where $\alpha(t_2) = t_1$, we make the substitution $y_1(t) = y(\alpha(t))$ and $y_2(t) = y(t)$. We can now use an ODE solver to solve

$$\begin{aligned} y_1'(t) &= \alpha'(t) f(\alpha(t), y_1(t), \varphi(\alpha(\alpha(t)))), && (4.6)\\ y_2'(t) &= f(t, y_2(t), y_1(t)), && (4.7)\\ y_1(t_1) &= \varphi(t_0), && (4.8)\\ y_2(t_1) &= y(t_1). && (4.9) \end{aligned}$$

Note that no interpolation of previous results is necessary here, because essentially the solution on the previous interval $[t_0, t_1]$ is recomputed (however solving it the first time was not useless, since the initial value $y(t_1)$ is required).

This method can be repeated for the following steps, yielding the following system of ODEs for the interval $[t_i, t_{i+1}]$:

$$y'_j(t) = \alpha'_{i-j}(t) f(\alpha_{k-j}(t), y_j(t), y_{j-1}(t)), \quad j = 1, \ldots, i, \tag{4.10}$$

$$y_j(t_{i-1}) = y(t_{i-1}). \tag{4.11}$$

This method does not suffer much from discontinuities, since it is restarted at every discontinuity point. However, it has one big disadvantage: for solving the i-th time step, the system of ODEs has length i.

4.2.3 Tracking discontinuities

As we have seen in section 4.2 , discontinuities tend to propagate in time, because of the delay terms. These discontinuities can be bad, because the local truncation error for many ODE solvers is of the form $c_k h^k y^{(k)}(\xi)$, where ξ is some point between two time steps, say t and $t+h$. If the $k-1$-st derivative has a discontinuity in $[t, t+h]$, a large local truncation error could occur.

However, if we follow the method of steps, or the standard approach for DDEs, the points of discontinuity will be exactly the points where the ODE solver will be restarted. Therefore, the discontinuity will not influence any local truncation error. The error around the discontinuities will behave similar to those around the starting point.

There is another reason that can cause discontinuities to occur: the discontinuities in the function $\tau(t)$ itself. Referring to equation 4.1, $x'(t)$ can only be continuous if the left derivative of $x(t)$ at point t_i is equal to the right derivative at $t - \tau(t_i)$, where $\tau(t)$ has a discontinuity at point t_i. This will happen only in very special cases. To be safe, it is a good idea to restart a solving routine at the discontinuity points of $\tau(t)$.

4.2.4 Delay DAEs

In the treatment of delay differential algebraic equations (DDAEs) (Baker, Paul, and Tian 2000), we do not have the guarantee that the solution smoothens out. A general form of a semi-explicit DDAE with one delay function is

$$y'(t) = F(t, x(t), x(\tau(t))) \tag{4.12}$$

$$0 = G(t, x(t), x(\tau(t))). \tag{4.13}$$

The presence of $\tau(t)$ in the algebraic part of the system, $G(t, x(t), x(\tau(t))$, causes discontinuities to jump to other times. The smoothening that we saw before does not occur here. This problem is very specific to delay differential algebraic equations. The only thing we can do is choosing a method which can handle this discontinuities appropriately, for example by restarting.

4.3 Waveform relaxation

Waveform relaxation (Zubik-Kowal and Vandewalle 1999) is an iterative method for solving ordinary differential equations. It differs from 'normal' iterative methods in that it iterates with function in a function space instead of with discrete unknowns. It can be suited for solving delay equations.

Consider the general form of a DDE (4.1). The general waveform relaxation method for this equation is an iteration scheme of the following form:

$$y'^{[k+1]}(t) = G\left(t, y^{[k+1]}(t), y^{[k]}(t), y^{[k+1]}(t-\tau(t)), y^{[k]}(t-\tau(t))\right) \quad (4.14)$$

$$y^{[k+1]}(0) = \varphi(t), \quad \text{for } t \leq 0, \quad (4.15)$$

where k is the iteration number, starting with $k = 0$. The function G is called the *splitting function*, it splits $f(t, y(t), y(t-\tau(t))$ into part that depends on the current solution, and a part that does not. The most basic splitting function is

$$G(\ldots) = f\left(t, y^{[k+1]}(t), y^{[k]}(t-\tau(t))\right), \quad (4.16)$$

that is, fill in a previous solution for the delay function, and further treat it as an ODE. If this is too difficult, other methods could be used.

A special property of waveform relaxation methods is that an ODE solver is not always necessary. For example, consider solving the standard ODE 3.1. As an initial solution, we take the constant function $x^{[0]}(t) = x_0$. A waveform relaxation for solving is now

$$x^{[i+1]}(t) = x_0 + \int_0^t f(x^{[i]}(\sigma), \sigma)\, \mathrm{d}\sigma. \quad (4.17)$$

If $x^{[i]}$ is the solution of the differential equation, the integral will yield the same function again, so $x^{[i+1]} = x^{[i]}$ upon convergence. This is why this is called fixed-point iteration. This method is known as the method of Picard-Lindel. This method only needs a numerical procedure to compute the integral in (4.17). Its main disadvantage is that the convergence may be very slow.

4.4 Vector extrapolation

Consider the solution of a general problem (not necessarily a differential equation), where iterates $x^{[k]}$ are obtained by a numerical procedure $x^{[k+1]} = G(x^{[k]})$. It is often the case that the error decreases linearly with the iteration number. If $x^* \in \mathbf{R}$ denotes the true solution of the problem, and $x^{[k]}$ the k-th approximation, it then holds that

$$x^{[k+1]} - x^* = \lambda(x^{[k]} - x^*), \quad (4.18)$$

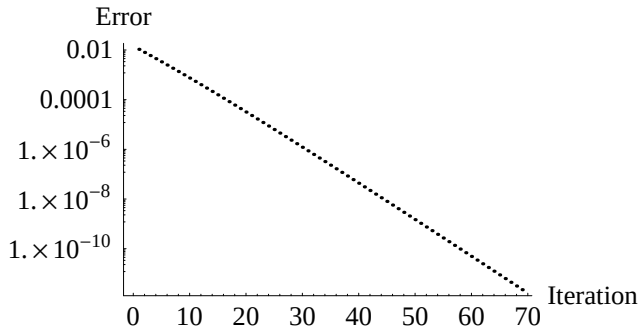


Figure 4.2: The sequence $\|x^{[i]} - x^*\|$, where C is a 2×2 matrix with real eigenvalues $\mu_{1,2} = 0.3$.

for $k \geq 0$. This equation of course also holds for $k + 2$:

$$x^{[k+2]} - x^* = \lambda(x^{[k+1]} - x^*). \tag{4.19}$$

If we now subtract 4.18 from 4.19, we can find the value for λ, without knowing x^*:

$$x^{[k+2]} - x^{[k+1]} = \lambda(x^{[k+1]} - x^{[k]}). \tag{4.20}$$

Substituting this value of λ into either 4.18 or 4.19 yields the value of x^*. We see that just by using three iterates and the knowledge that the error decreases linearly, we can find the solution. In practical cases, only the leading order term of the error behaves linearly, and it is only this term that we eliminate by this procedure (keeping higher-order terms). So the value that is obtained with this extrapolation procedure will not be the true value of x^*, and further iterations are necessary. If the normal iteration procedure is followed, the method is called the *Aitken* method. However, often the new iterates also show linear convergence, and another extrapolation can be carried out after three iterates. This method is called the *Steffensen* method (Stoer and Bulirsch 1978).

We now consider the case where $x^* \in \mathbf{R}^n$ is a vector. In this case, the error will decrease linearly as

$$x^{[k+1]} - x^* = C(x^{[k]} - x^*), \tag{4.21}$$

where C is an $n \times n$ matrix. Examples of convergence plots are shown in figure 4.2 and 4.3. Note that the presence of complex eigenvalues can cause the graph to have a specific shape (there are two of these shapes in figure 4.3, representing two complex eigenvalues).

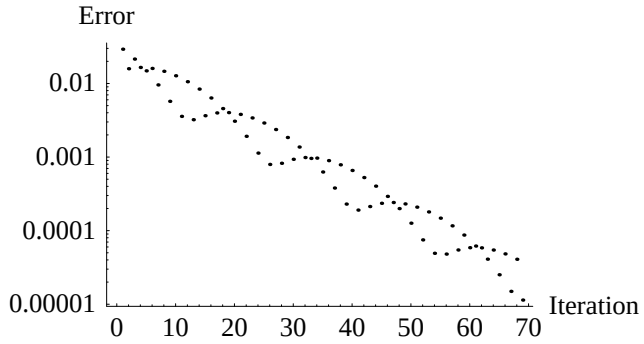


Figure 4.3: The sequence $\|x^{[i]} - x^*\|$, where C is a 2×2 matrix with complex eigenvalues $\mu_{1,2} = 0.9(1 \pm i)$.

Computing the matrix C takes a lot more than three iterates (if $n > 3$). Equation 4.21 does provide n equations (since the x's are vectors), so it could be expected that n iterates will provide enough equations for determining the n^2 elements of C. They indeed do, as follows. Define the vectors of first and second order differences

$$u^{[k]} = x^{[k+1]} - x^{[k]}, \qquad k = 0, 1, \ldots, n-1, \tag{4.22}$$

$$v^{[k]} = x^{[k+2]} - 2x^{[k+1]} + x^{[k]}, \qquad k = 0, 1, \ldots, n-1, \tag{4.23}$$

and the $n \times k$ matrices

$$U^{[k]} = \begin{pmatrix} | & | & & | \\ u^{[0]} & u^{[1]} & \ldots & u^{[k-1]} \\ | & | & & | \end{pmatrix}, \; V^{[k]} = \begin{pmatrix} | & | & & | \\ v^{[0]} & v^{[1]} & \ldots & v^{[k-1]} \\ | & | & & | \end{pmatrix}. \tag{4.24}$$

From 4.21 we see that for all k

$$v^{[k]} = \left(x^{[k+2]} - x^{[k+1]}\right) - \left(x^{[k+1]} - x^{[k]}\right) = (C - I_n)u^{[k]}. \tag{4.25}$$

Combining this for all k yields

$$V^{[n]} = (C - I_n)U^{[n]}. \tag{4.26}$$

This yields, if V is nonsingular,

$$(I_n - C)^{-1} = -U^{[n]}(V^{[n]})^{-1}. \tag{4.27}$$

Using basic algebra, the solution x^* can now be obtained through

$$x^* = x^{[0]} - U^{[n]}(V^{[n]})^{-1}u^{[0]}, \tag{4.28}$$

the solution of which can be found by solving

$$V^{[n]}y = u^{[0]}, \tag{4.29}$$

followed by computing

$$x^* = x^{[0]} + U^{[n]}y. \tag{4.30}$$

This method is called *full rank extrapolation* (FRE). If n is big, the computation of the extrapolation can be too high to be useful. However, if we perform only k iterations, we already have the $n \times k$ matrices $U^{[k]}$ and $V^{[k]}$. Using these matrices can be enough to solve equation 4.29 in a least squares sense. We hope that the obtained solution is a good approximation to x^*. The minimum number of iterations necessary is 3, just like in the one-dimensional case. This method is called *reduced rank extrapolation* (RRE).

A common procedure (referred to as *cycling*) for performing RRE is as follows:

1. Take an initial guess $x^{[0]}$, set $i \leftarrow 0$.
2. Compute $x^{[i+1]} = G(x^{[i]})$.
3. Compute $x^{[i+2]} = G(x^{[i+1]})$.
4. Extrapolate: compute x^* from $x^{[i]}, x^{[i+1]}, x^{[i+2]}$.
5. Set $x^{[i+3]} = x^*$, $i \leftarrow i + 3$, goto step 2.

In the case that the fixed point operator $G(x)$ is linear, i.e.

$$G(x) = Ax + b, \tag{4.31}$$

a k-step reduced rank extrapolation method has an extra advantage. Since, from 4.30,

$$x^* = x^{[0]} + U^{[k]}y, \tag{4.32}$$

we can write

$$G(x^*) = G(x^{[0]}) + G(U^{[k]}y) = x^{[1]} + \sum_{i=0}^{k-1} G(u^{[i]}y_i) = x^{[1]} + \sum_{i=1}^{k} u^{[i]}y_i. \tag{4.33}$$

This means that one evaluation of G (the one in step 2 of the procedure above, for $i \geq 3$) is not necessary: it can be computed directly from previous solutions.

In the case that the original problem is linear, these methods are closely related to Krylov subspace methods (see van der Vorst (2003) for an introduction). The columns of the matrix $U^{[k]}$ contains the errors which span the

k-dimensional Krylov subspace. In fact, the reduced rank extrapolation can be shown to be equivalent to the GMRES(k) method (Sidi 1991). If we take the operator G to be as in 4.31, we are looking for the stationary solution x^* such that

$$G(x^*) = x^*, \tag{4.34}$$

or

$$\begin{aligned} Ax^* + b &= x^* && (4.35)\\ (I - A)x^* &= b. && (4.36) \end{aligned}$$

To show the equivalence of RRE and GMRES, it is enough to show that $x^{[i]} \in \mathcal{K}_i(I - A, r^{[0]})$, and that $r^{[i+1]} \perp \mathcal{K}_i(I - A, r^{[0]})$, where $r^{[i]} := b - (I - A)x^{[i]}$. The Krylov subspace used for the vector $x^{[i]}$ is

$$\begin{aligned} \mathcal{K}_i(I - A, u_0) = \text{span}\{u_0, (I - A)u_0, \ldots, (I - A)^{i-1}u_0\} = && (4.37)\\ = \text{span}\{u_0, Au_0, \ldots, A^{i-1}u_0\} = \{u_0, u_1, \ldots u_{k-1}\}, && (4.38) \end{aligned}$$

where we have used that

$$r^{[0]} = b - (I - A)x^{[0]} = x^{[1]} - x^{[0]} = u^{[0]}. \tag{4.39}$$

It has been shown by Andersson (1997) that $r^{[i+1]}$ is indeed perpendicular to $\mathcal{K}_i(I - A, u_0)$, by which the equivalence of RRE and GMRES follows.

Chapter 5

Numerical treatment of the circulation model

5.1 Introduction

In this chapter we will explain how the techniques from the previous two chapters have been applied to solve the specific problem from chapter 2. Even though we used the standard solver IDA, some specific methods had to be applied to handle discontinuities, as well as the delay term from equation (2.2). Also, some Mathematica-related issues will be discussed.

5.2 General remarks

The systems that had to be solved (2.14–2.27 and 2.28–2.46) are largely just differential algebraic equations. They can both be separated into a differential and an algebraic part, which puts them into the semi-explicit form. There is only one delay-term, which appears in the algebraic part.

Since the input functions $f_{LV}(t)$, $f_{RV}(t)$ and $p_e(t)$ are all periodic, it is expected that the solution reaches a steady state, where the solution is periodic too. We will treat the system for one heart beat at a time.

5.3 Tracking discontinuities

In the human circulatory system, valves cause discontinuities. This is not just a mathematical property, but also physiological. When the aortic valve closes, it does so with such a force, that it can even be felt in the pulse. As an example,

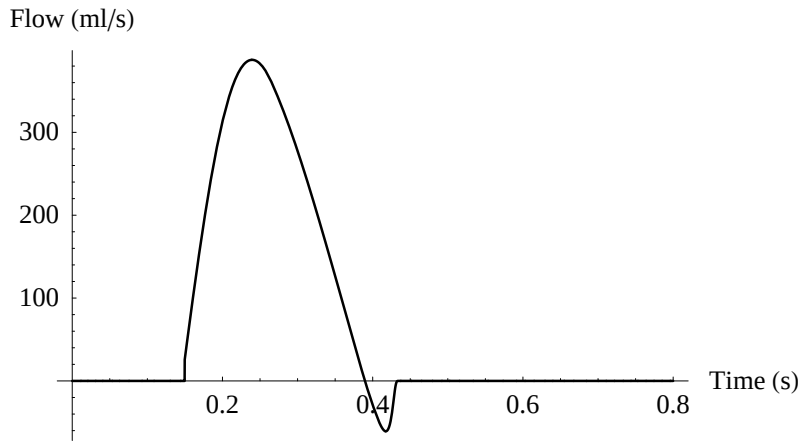


Figure 5.1: Graph of the flow through the aortic valve vs. time.

the flow through the aortic valve $Q_{LV}(t)$ is shown in figure 5.1. A discontinuity is clearly present in the derivative of this function, around $t = 0.43$s.

As mentioned in section 3.2, backward difference formulas (as well as most other methods) have troubles with discontinuities. To avoid these problems, we have chosen to stop solving at whenever a discontinuity occurs, and then restart. As initial values for the new solving method, the values at the end point of the previous run are taken, but more historical values are not used. This causes the numerical methods to start again, using only a 1-step formula here (implicit Euler), which is the best it can do (initially).

5.4 Rewriting to ODE

Let's consider the system of equations for model A. This system can be written in the form

$$x_1'(t) = F(x_1(t), x_2(t), t) \tag{5.1}$$
$$0 = G(x_1(t), x_2(t), t). \tag{5.2}$$

We treat each state of the valves separately. As an example, let us take that state with the systemic A-V valve closed, and the aortic valve open. The values of $x_1(t)$ and $x_2(t)$ are now give by

$$x_1(t) = \{p_p(t), V_{LV}(t)\}, \tag{5.3}$$
$$x_2(t) = \{Q_{LV}(t), Q_{H15}(t), Q_{H1}(t), Q_{V9}(t), p_5(t), p_6(t)\}. \tag{5.4}$$

In this state, $G(x_1(t), x_2(t), t)$ is a linear function, consisting of only applications of Kirchhoff's law. It can even be inverted. We used the Mathematica-command `Solve` for this, which yields its results analytically. So we can write

$$x_2(t) = G^{-1}(x_1(t), t). \tag{5.5}$$

This is very fortunate, since the resulting differential equation (5.3) can now be solved using a standard ODE solver.

The Mathematica code for this little experiment:

```
system
```

```
{QV10[t] == C10 pp'[t], QV9[t] == VLV'[t], RLV QH12[t] == 0,
 ZOS QH15[t] == p6[t] - pp[t], pp[t] == RS QH1[t], QH12[t] == QH15[t] + QV9[t],
 QV10[t] == -QH1[t] + QH15[t], p6[t] - pe[t] == aLV (-bLV + VLV[t])^2 +
   (fLV[t] - k1 QH15[t] + k2 QH15κ[t - κ Mod[t, th]]^2) (-dLV + cLV VLV[t])}
```

```
dsystem = system[[{1, 2}]];
asystem = system[[{3, 4, 5, 6, 7, 8}]];

dvars = {pp[t], VLV[t]};
avars = {p6[t], QH12[t], QH15[t], QH1[t], QV9[t], QV10[t]};

asol = First[Solve[asystem, avars]];

initial = {pp[0] == 50, VLV[0] == 100};

dsol = NDSolve[
    Join[dsystem /. asol /. QH15κ[_] → 0, initial], dvars, {t, 0, th}
  ] // First;

asol2 = asol /. dsol;
```

5.5 Approach for delay term

In the equations for ventricular contractions (2.1 and 2.2), a delay term appears, containing a function value evaluated in the past:

$$F_{RV}(Q_{H9}(t), t) = f_{RV}(t \bmod t_h) - k_1 Q_{H9}(t) + k_2 Q_{H9}(t - \kappa t \bmod t_h)^2. \tag{5.6}$$

The delay function $\tau(t)$ is given by $\tau(t) = \kappa t \bmod t_h$. Note that $\tau(0) = 0$ and $\tau(t)$ is strictly increasing, so that we do not need to specify historical function values.

Problems can be expected since the delay function contains discontinuities. However, it turns out that the solutions $Q_{H9}(t)$ and $Q_{H15}(t)$ (the terms to which the delay applies) are identically zero in a large part of each heart

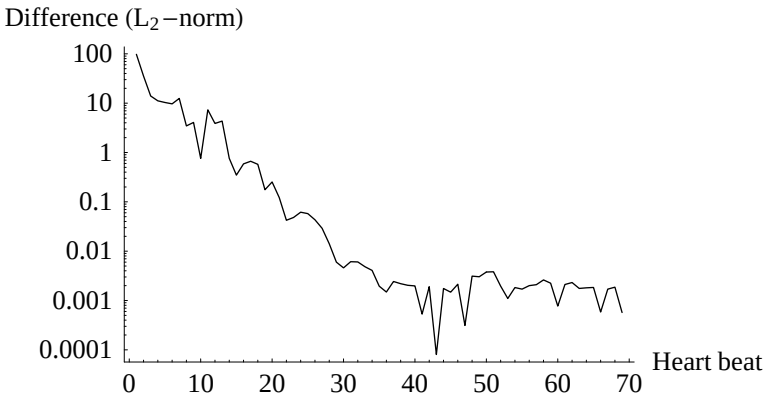


Figure 5.2: Difference between successive computations of $Q_{H9}(t)$ using model B2, in L_2-norm (logarithmic plot).

beat. Specifically, $Q_{H9}(t - \kappa t \bmod t_h) = Q_{H15}(t - \kappa t \bmod t_h) = 0$ around the discontinuities of $\tau(t)$. In this section, we will write $Q_{H9}(t)$ from now on, but everything mentioned here will also be applicable to $Q_{H15}(t)$.

5.5.1 Standard approach

The standard approach for delay differential equations cannot be used to solve this delay equation. The problem is that $\tau(t) = \kappa t \bmod t_h$ is not bounded from below. To overcome this problem, we make use of the fact that the solution of the model is, or should become, periodic. This means that functions will not differ much between heart beats. Therefore, we can safely replace $\kappa t \bmod t_h$ by $\tau(t) = \kappa t \bmod t_h + t_h$, i.e. we take the value at the previous heart beat.

Now we can take a lower bound for $\tau(t)$, namely t_h. As an initial function we take

$$Q_{H9}(t) = 0 \qquad \text{for } t < 0. \tag{5.7}$$

We do make an error in taking the value of the flow at the previous heart beat. A measure in this error is how close the flow function $Q(t)$ differs between two heart beats. A measure for this difference is the L_2 norm. In figure 5.2, the distance between $Q_{H9}(t - (i+1)t_h)$ and $Q_{H9}(t - it_h)$ is shown, the difference at the i-th heart beat.

In this example, the accuracy goal was set to 6 digits. This means that the step sizes are taken small enough to assure that the solution will be correct up to 6 digits. That this works can also be seen from figure 5.2. The function $Q_{H9}(t)$ takes values around 100ml/s. The difference between two heart beats

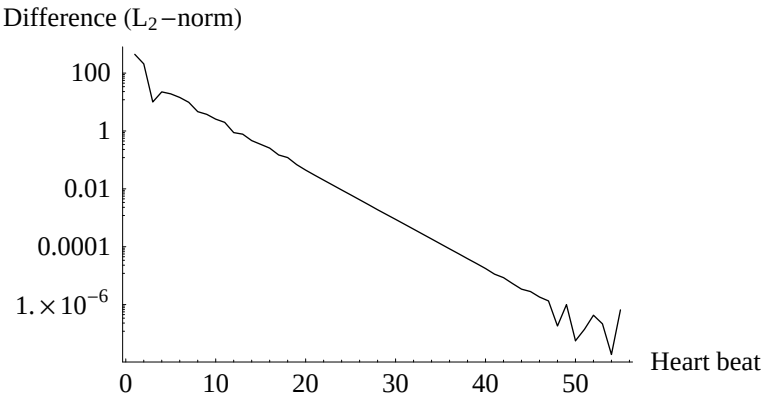


Figure 5.3: Difference between successive computations of $Q_{H9}(t)$ using model B2x (pathological), in L_2-norm (logarithmic plot). No delay term is present.

after convergence has occurred can be seen to be around 0.001. This implies that indeed around 6 digits are correct.

For reference, a similar plot is given in figure 5.3, which was obtained in pathological circumstances, where $F_{RV}(t) = 0$, and there is no delay term whatsoever. This plot shows linear convergence (to working precision), which is also present in figure 5.2, but in the latter graph the delay plays a role too.

5.5.2 Waveform relaxation

The solution above makes heavy use of the periodicity of the solution. However, it is anticipated that future versions of the model will not be periodic. Therefore, we have also tested a waveform relaxation method. The splitting function here is simply

$$G = f\left(t, x^{[k+1]}(t), Q_{H9}^{[k]}(t - \tau(t))\right), \tag{5.8}$$

which means that only the delay term will be replaced by an estimate from a previous iterate.

Note that this waveform relaxation very closely resembles the method we have taken above. If we solve the system for one heart beat, $0 \leq t \leq t_h$, the solutions are exactly the same. However, the waveform relaxation approach could also be used to solve for 20 heart beats with an initial guess of 0 for the function with the delay, $Q_{H9\kappa}$, and then substitute the obtained $Q_{H9}(t)$ into the system to obtain a new estimate, for the entire period $(0, 20t_h)$. The

```
system = {
   Qc[t] + Q1[t] == Q2[t],
   Q1[t] R1 == If[p2[t] > p3[t], p2[t] - p3[t], 0],
   Qc[t] == C1 (p2'[t] - p3'[t]),
   R2 Q2[t] == p3[t],
   p2[t] == V[t]
  };
initial = {
   p3[0] == 0,
   p2[0] == 0
  };
vars = {
   Qc[t], Q1[t], Q2[t], p2[t], p3[t]
  };
R1 = 1;
R2 = .01;
C1 = .1;
V[t_] = fLV[Mod[t, th]];

NDSolve[Join[system, initial], vars, {t, 0, 5}, AccuracyGoal → 3]
```

Figure 5.4: Code for the computation of the electrical valve model.

main advantage here is that the stabilisation of the periodic system does not influence the convergence of the delay approximation procedure.

In pseudocode, the iteration scheme used was:

1. Set $Q_{H9\kappa}(t) = 0$.
2. Compute the solution of 2.28–2.46 for $0 \leq t \leq 20t_h$.
3. Set $Q_{H9\kappa}(t)$ to the computed solution of $Q_{H9}(t)$.
4. If convergence: stop.
5. Go to step 2

5.6 Mathematica issues

As mentioned before, Mathematica contains a link to the numerical package SUNDIALS, specifically to IDA, its DAE-solver. A problem can be solved rather simple. As an example, the code used to calculate the 'electrical valve model' (section 2.4.2) is given in figure 5.4.

From this example, it should be clear that a very intuitive interface is provided to the numerical package; only one line of real code is used to invoke

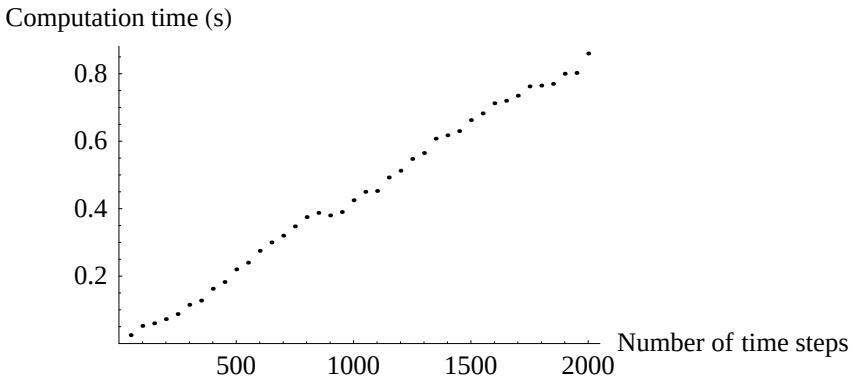


Figure 5.5: Time experiment for the electrical valve model.

it. Inside Mathematica, all equations and numbers are represented as objects (this makes it possible to define the system of equations even before the parameter values). Calculations inside the program are done analytically: if desired, a computation can be performed accurately up to 100 decimal places. This is in contrast to IDA, which is written in C. Here, numbers are represented by `double`'s, containing only 15 digits. The conversion between these two data structures is performed by Mathematica upon the call of `NDSolve`. Also, the system of equations is converted into a form that is understandable by IDA. That this takes very little time is shown in figure 5.5, showing the computation time vs. the number of time steps taken by IDA. This curve is almost linear (as it should be), and it intersects the origin near 0 sec, meaning that the time Mathematica takes for processing the equation is negligible.

Chapter 6

Results

6.1 Introduction

This chapter will show some basic results that were obtained using models A and B2. Various experiments are shown to determine an optimal resuscitation function.

6.2 Model A

The pressure curves of the left ventricle, atrium and aorta are shown in figure 6.1. The curves resemble those that are actually measured in patients. The blood pressure can be seen to be between 120 and 80, which are the reference values for standard medical practice (aiming at the 30-year old, 70 kg. John Doe). Flows through the mitral valve and the aortic valve are shown in figure 6.2

It is interesting to compare these graphs to those obtained by resuscitation. For a maximum resuscitation pressure $p_{\text{emax}} = 50\text{mmHg}$, the pressures and flows are shown in figure 6.3 and 6.4. From these figures, it can be seen that the chest compressions perform very well at replacing the ventricular contractions. The shape of the graphs is almost the same, and even the values correspond reasonably, indicating that 50mmHg is a good resuscitation pressure.

6.3 Model B2

The same experiments as in model A were also performed on the right heart half, model B2. The physiological outcomes are shown in figures 6.5 and 6.6,

while the pathological results (again with $p_{\text{emax}} = 50\text{mmHg}$) are shown in figures 6.7 and 6.8.

It is remarkable that with the same resuscitation pressure, the flows (and thus the cardiac output) for model A are below normal, which is to be expected, and above normal for model B2. This shows that the right ventricle is much more susceptible to resuscitation than the left ventricle. There is a physiological reason for this: the walls of the right ventricle are thinner than those of the left ventricle.

As mentioned before, it is not clear on which parts of the heart the resuscitation pressure is exerted. If this is just the ventricle, we are using the cardiac pump model, otherwise we are in the thoracic pump model. We certainly hope to compress the right and left ventricle, but from a anatomical point of view, it can be argued that we are mainly compressing on the right atrium and the outflow tracts (Noordergraaf, van Tilborg, Schonen, Ottesen, and Noordergraaf 2004). Several experiments have been done to see what happens in these circumstances. These experiments were carried out with the simplest valve model, the diode model from section 2.4.1. We are interested mainly in the flows through the tricuspid valve and the pulmonary valve. These flows (obtained with $p_{\text{emax}} = 25\text{mmHg}$) are printed for three settings:

- Compression works on just the right ventricle (figure 6.9).
- Compression works on just the right atrium (figure 6.10).
- Compression works on both the right ventricle and the right atrium (figure 6.11).

From these experiments, it can be seen that the compression on the ventricle has a good effect. The cardiac output is sometimes even higher than under physiological circumstances. In contrast, compressing the atrium does not have a large effect on the cardiac output. Large flows into and out of the atrium occur, but rather than flowing to the ventricle, the blood tends to flow right back into the venae cavae. This may be due to a lack of pressure applied to the latter. Only a little blood flows into the ventricle. When compressing both the atrium and the ventricle, the results seem to be a superposition of the previous two: the cardiac output is only slightly higher than in the case where only the ventricle was compressed.

The result of varying the maximum resuscitation pressure p_{emax} is shown in figure 6.12. It is almost linear. We even ran the simulation with higher p_{emax}, but the results from these (which lay on the same line) were not valid since a negative volume occurred in the right atrium. Fixing this could cause the line in figure 6.12 to flatten and approach a certain maximum.

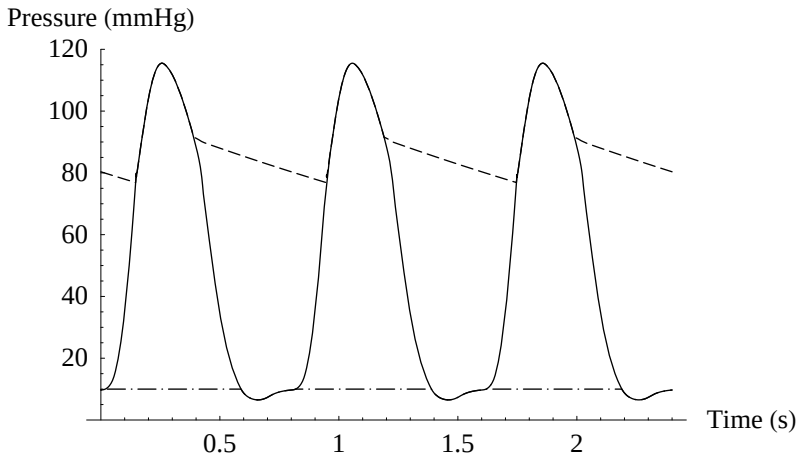


Figure 6.1: Pressure under physiological circumstances in model A. Solid: ventricular pressure $p_6(t)$, dashed: aortic pressure $p_p(t)$, dash-dotted: atrial pressure $p_v(t)$.

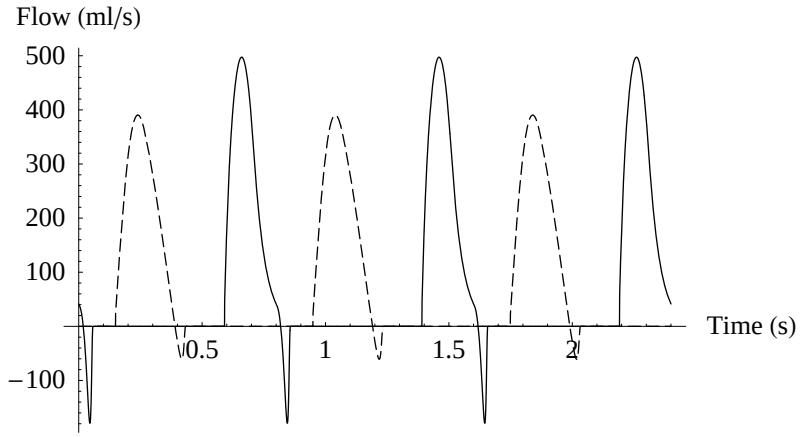


Figure 6.2: Flow through the mitral valve (dashed) and the aortic valve (solid) in model A under physiological circumstances.

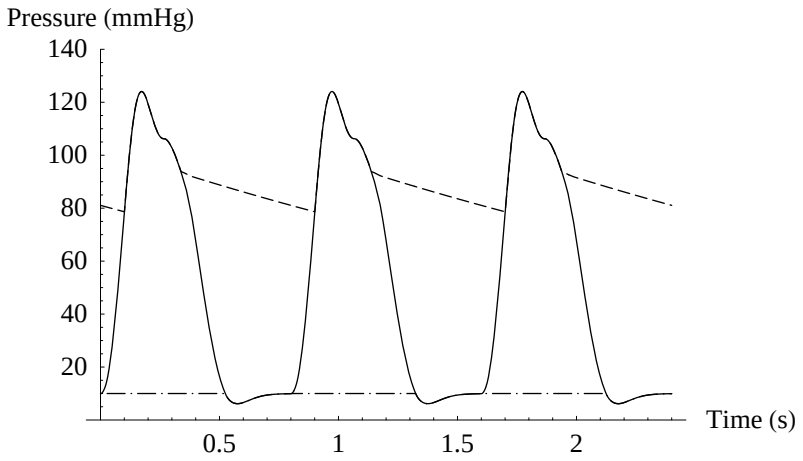


Figure 6.3: Pressure under pathological circumstances in model A. Solid: ventricular pressure $p_6(t)$, dashed: aortic pressure $p_p(t)$, dash-dotted: atrial pressure $p_v(t)$.

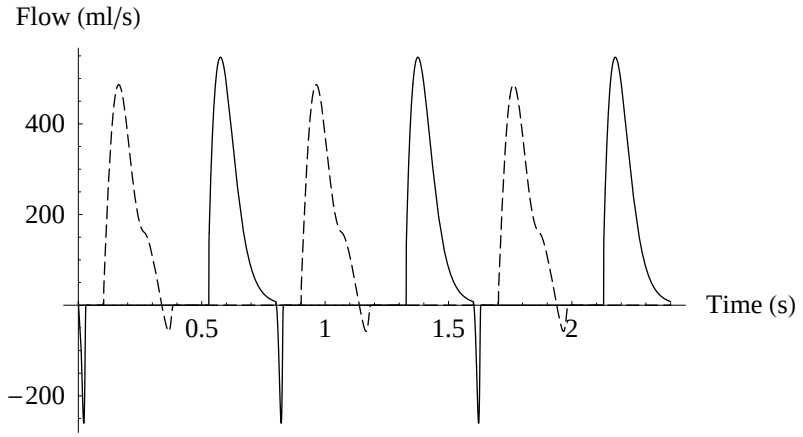


Figure 6.4: Flow through the mitral valve (dashed) and the aortic valve (solid) in model A under pathological circumstances.

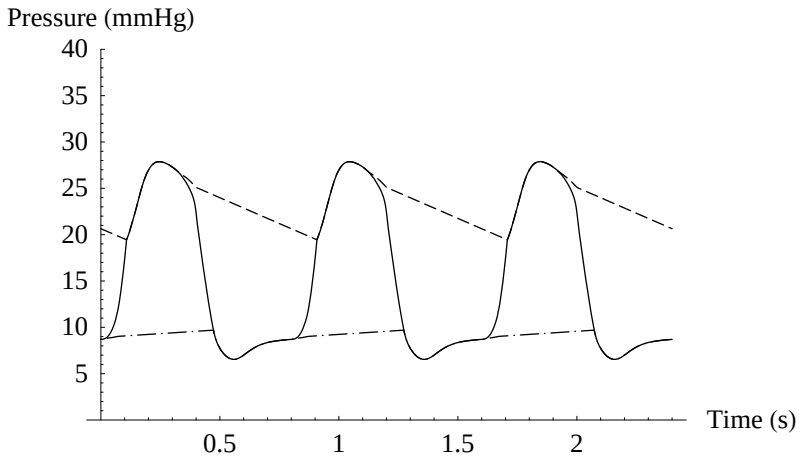


Figure 6.5: Pressure under physiological circumstances in model B2. Solid: ventricular pressure $p_2(t)$, dashed: pulmonary arterial pressure $p_3(t)$, dash-dotted: atrial pressure $p_1(t)$.

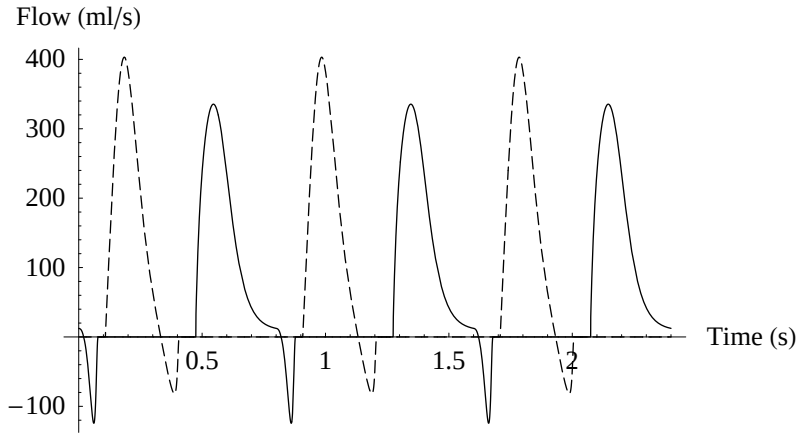


Figure 6.6: Flow through the tricuspid valve (dashed) and the pulmonary valve (solid) in model B2 under physiological circumstances.

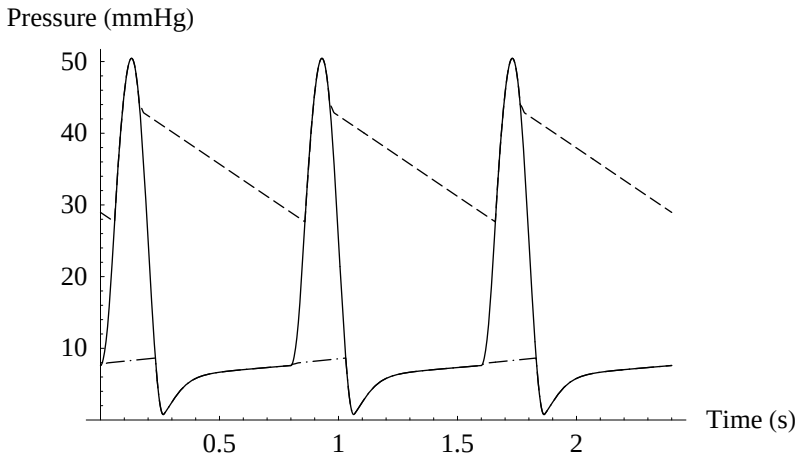


Figure 6.7: Pressure under pathological circumstances in model B2. Solid: ventricular pressure $p_2(t)$, dashed: aortic pressure $p_3(t)$, dash-dotted: atrial pressure $p_1(t)$.

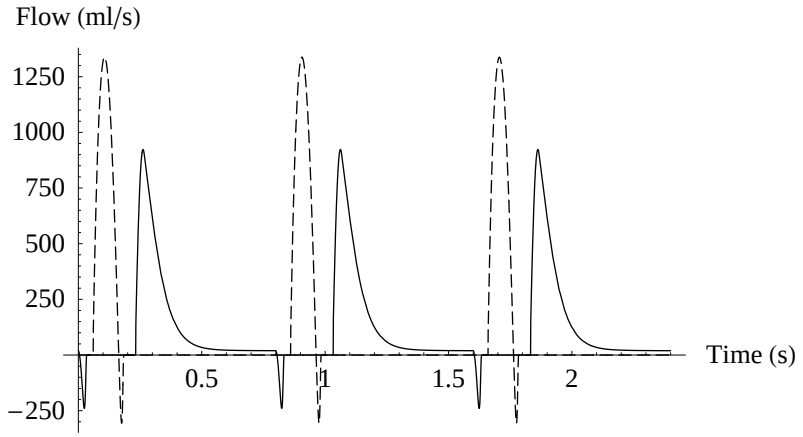


Figure 6.8: Flow through the tricuspid valve (dashed) and the pulmonary valve (solid) in model B2 under pathological circumstances.

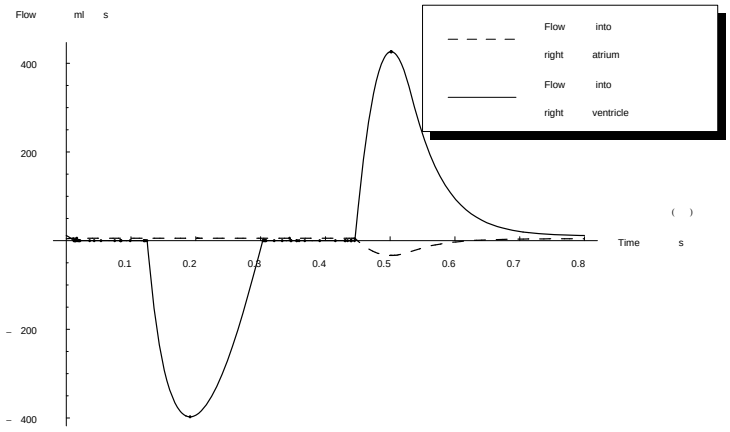


Figure 6.9: Flow into the right ventricle (solid) and atrium (dashed) in model B2 under pathological circumstances, the pressure is applied *only to the ventricle*.

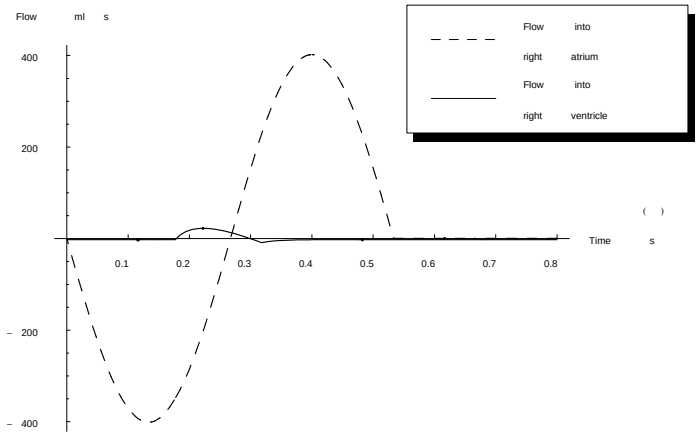


Figure 6.10: Flow into the right ventricle (solid) and atrium (dashed) in model B2 under pathological circumstances, the pressure is applied *only to the atrium*.

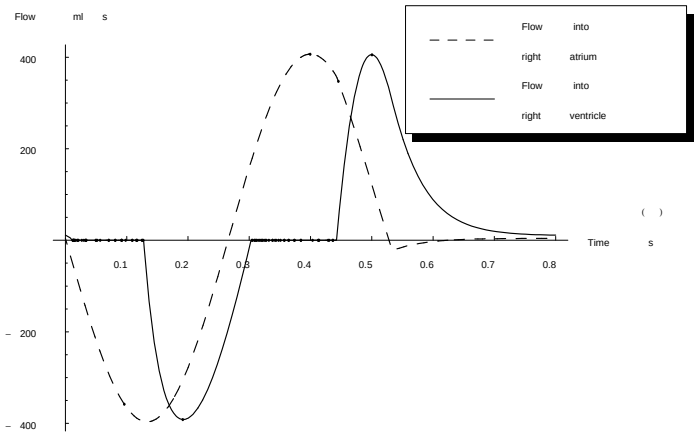


Figure 6.11: Flow into the right ventricle (solid) and atrium (dashed) in model B2 under pathological circumstances, the pressure is applied *to both the ventricle and the atrium*.

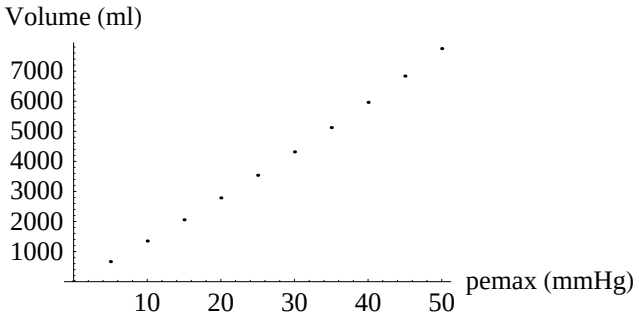


Figure 6.12: The effect of the maximum resuscitation pressure on the cardiac output in model B2, while exerting the pressure only on the ventricle.

Chapter 7

Conclusions

The models A and B2 are capable of producing output that is comparable to what is physiologically expected. Because they are based on physiological insights, individual parts of the model (such as the valves) can be treated individually.

From the results of models A and B2, resuscitation seems to work. Even if the pressure is applied also to the right atrium, there is still a (perhaps too) good cardiac output.

The standard method for delay differential equations works nicely for working with the delay equation, even combined with the differential algebraic system solver. Mathematica is a very nice environment to work in, also numerically. It is very clear what is being calculated; the equations are not obfuscated by matrix-vector notation.

7.1 Further work

Physiologically, some work needs to be done to transform the model into a more accurate description of the circulatory system. A first point in this direction is of course to integrate the models A and B2, to form a closed loop including both the left and the right heart. Also, parts need to be included representing for example the coronary and cerebral circulation (the circulation which supplies oxygen to the heart and brains). The atria, which are now passive, should become active pumps.

Another modelling improvement could be to apply the mechanical valve model also to the opening of the valves. While the current model, which opens the valves instantaneously, suits well for the current experiments, a fully functional valve model could help in simulation of certain defects in the heart.

An important issue which should be fixed is the occurrence of negative

volumes. They do not appear under physiological (neither in the real world nor in the models), but the simulations for pathological circumstances, with a high value for p_{emax} do show them. This causes that there is, according to the models, no upper bound on p_{emax}.

7.2 Numerical improvements

Figure 5.3 shows linear convergence, like the example figure 4.2. This linear convergence could be used to speed up the convergence. An extrapolation method like in section 4.4 could be used. Figure 5.2 shows behaviour like figure 4.3, which could mean that the delay equation causes the system to have a complex eigenvalue. To use extrapolation here, a somewhat smarter method has to be used (requiring more than 3 iterates to compute the extrapolate).

Bibliography

Andersson, C. (1997). Solving Linear Equations on Parallel Distributed Memory Architectures by Extrapolation. Technical Report TRITA-NA-E9746, Royal Institute of Technology.

Baker, C. T., C. A. Paul, and H. Tian (2000). Differential Algebraic Equations with After-Effect. Technical Report 367, Manchester Centre for Computational Mathematics.

Bellen, A. and M. Zennaro (2003). *Numerical Methods for Delay Differential Equations*. Oxford University Press.

Brenan, K., S. Campbell, and L. Petzold (1989). *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. New York: North-Holland.

Bruin, S. M. (2001). Modified Extended BDF applied to circuit equations. Master's thesis, Vrije Universiteit Amsterdam.

Donders, F. C. (1856). *Physiologie des Menschen*. Leipzig: Hirzel.

Gear, C. (1981). Simultaeous numerical solution of differential-algebraic equations. *IEEE Trans. Circuit Theory CT-18*, 89–95.

Guyton, A. C. and J. E. Hall (1986). *Textbook of Medical Physiology*. W.B. Saunder Company.

Hairer, E. and G. Wanner (1997). *Solving Ordinary Differential Equations* (2nd ed.), Volume 2. Springer.

Harvey, W. (1628). *Exercitatio Anatomica de Motu Cordis et Sanguinis in Animalibus*, Chapter 14. Frankfurt.

Hayes, A. (2004). NDelayDSolve.

Hindmarsh, A. C., P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward (2003, September). SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. Technical Report UCRL-JP-200037, Lawrence Livermore National Laboratory.

Lambert, J. (1991). *Numerical methods for ordinary differential equations*. Wiley & Sons.

Moser, M., J. W. Huang, G. S. Schwarz, T. Kenner, and A. Noordergraaf (1998). Impedance defined flow. Generalisation of William Harvey's concept of the circulation—370 years later. *International Journal of Cardiovascular Medicine and Science 1*(3/4), 205–211.

Mukkamala, R. (2000). *A forward model-based analysis of cardiovascular system identification methods*. Ph. D. thesis, MIT.

Noordergraaf, G., G. van Tilborg, J. Schonen, J. Ottesen, and A. Noordergraaf (2004). Thoracic CT-scans and cardiovascular models: the effect of external force in CPR. *The International Journal of Cardiovascular Medicine and Science* (4).

Palladino, J. L. and A. Noordergraaf (2002). A paradigm for quantifying ventricular contraction. *Cellular & Molecular Biology Letters 7*(2), 331–335.

Petzold, L. R. (1983). A description of DASSL: a differential/algebraic system solver. In R. Stepleman et al. (Eds.), *Scientific Computing*, Amsterdam, pp. 65–68. Sandia National Laboratories: North-Holland.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992). *Numerical Recipes in C* (Second ed.). Cambridge University Press.

Sidi, A. (1991). Efficient Implementation of Minimal Polynomial and Reduced Rank Extrapolation Methods. *J. Comput. Appl. Math 36*, 305–337.

Stoer, J. and R. Bulirsch (1978). *Einführung in die Numerische Mathematik*, Volume 2. Springer.

Tischendorf, C. (1996). *Solution of index-2 differential algebraic equations and its application in circuit simulation*. Ph. D. thesis, Humboldt University Berlin.

van de Vosse, F., J. de Hart, C. van Oijen, D. Bessems, A. Segal, B. Wolters, J. Stijnen, and F. Baaijens (2003). Finite-element-based computational methods for cardiovascular fluid-structure interaction. *J. Eng. Math.* (47), 335–368.

van der Vorst, H. A. (2003). *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press.

Wolfram, S. (2003). *The Mathematica Book* (Fifth ed.). Wolfram Media/Cambridge University Press.

Zubik-Kowal, B. and S. Vandewalle (1999). Waveform Relaxation for Functional-diifferential Equations. *Siam J. Sci. Comput. 21*(1), 207–226.