NTNU – Trondheim
Norwegian University of
Science and Technology

The Time Varying Elastance Model used as a Boundary Condition in Arterial Network Simulations

Knut Petter Maråk

Mechanical Engineering
Submission date: July 2013
Supervisor: Leif Rune Hellevik, KT
Co-supervisor: Paul Roger Leinan, KT

Norwegian University of Science and Technology
Department of Structural Engineering

1 Assignment

Pressure and flow pulses originating in the heart, propagate and are reflected in the arterial tree. The propagation may be estimated by hyperbolic one-dimensional differential equations accounting for mass and momentum transport. The objective in this thesis will be to investigate and implement adequate numerical methods for such hyperbolic differential equations. In particular, the methods for the boundary conditions will be a focus. Further, the topology of the network must be described, and the geometry and material properties of each blood vessel must be accounted for.

2 Abstract

vascular1Dflow is a program for simulating blood flow in vascular networks. It uses hyperbolic one-dimensional partial differential equations with ordinary differential equations as boundary conditions. The focus of this work has been the development boundary conditions, primarily a time varying elastance model of the left ventricle has been implemented as a boundary condition. The aortic and mitral valves were modeled using a mathematical valve model. The varying elastance model is tested in lumped model and arterial network simulations. An improved solution algorithm for computing pressure and flow at bifurcations is implemented and tested. Also a improved discretization scheme for Windkessel models is presented.

3 Sammendrag

vascular1Dflow er eit program brukt til a simulere blodstrømning i blodårenettverk. Det bruker hyperbolske ein-dimensjonale partielle differensiallikningar med ordinære differensiallikningar som randbetingelsar. Fokus for dette arbeidet har vore utviklinga av randbetingelsar. Hovudsakleg har ein tidsvariant elastans-modell for venstre hjertekammer blitt implementert som ein randbetingelse. Aorta-klaffen og mitral-klaffen blei modellert med ein matematisk hjerteklaffmodell. Hjertemodellen har blitt testa i simuleringar på 0-dimensjonale modellar og i eit blodårenettverk. Ein forbetra algoritme for utrekning av trykk og strømning i forgreiningar har blitt implementert og testa. I tilegg blir eit forbetra diskretiseringsskjema for Windkessel-modellar presentert.

Contents

List of Figures

List of Tables

4 Introduction

The cardiovascular system is responsible for transporting oxygen from the lungs and into all the body's organs as well as providing the organs with neccesary nutrients. Blood is circulated through two separate vascular systems. The pulmonary circulation goes through the lungs where the blood is oxygenated. The systemic circulation transports the oxygenated blood to all the body's organs. The heart is the pump that drives the circulation of blood through the two vascular networks. It consists of four chambers, the right atrium, the right ventricle, the left atrium and the left ventricle. These work in pairs where the first two are responsible for pumping blood into the pulmonary circulation and the second into the systemic circulation. The pumping action of the heart is provided by periodic contractions of muscles in the heart walls. Blood flow between the chambers is controlled by four valves which open and close depending on pressure gradients only allowing blood to flow in one direction.

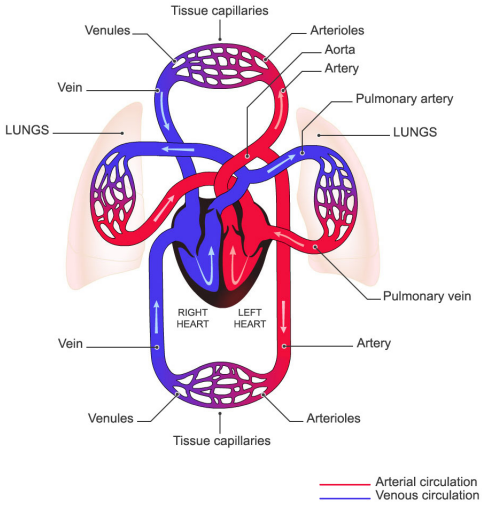


Figure 1: The cardiovascular system

The *vascular1Dflow* software. The work done in this thesis builds on the software *vascular1Dflow*, which was developed by Paul Roger Leinan and Vinzenz Eck. This program simulates bloodflow in vascular networks using a one dimensional model for fluid structure interaction in the blood vessels. Boundary conditions are handled using prescribed flow and pressure or by using lumped models where pressure and flow is related using ordinary differential equations. The program can load network data from .xml-files and also has the ability to plot simulation results in 2D and 3D and save results to the hard drive.

The purpose of this project has been to develop the *vascular1Dflow* software further. The main task has been to implement and test a varying elastance heart model along with a dynamic model for opening and closure of the aortic and mitral valves, the details of these implementations are shown in section 7.1. Furthermore the algorithm for solving the governing equations for bifurcations was evaluated and a new algorithm was suggested. Also the solution method for the two and three element windkessel boundary conditions was improved, specifically the discretization scheme was altered.

5 Theory

5.1 Mechanical properties of blood vessels

The mechanical properties of the blood vessels are primarily described by the relationship between internal pressure and cross-sectional area. In a differential form this is given by the vessel compliance $C = \frac{dA}{dp}$. Laplaces law provides a relation between the transmural pressure and dilatation of the vessel and is expressed as $p_{int} - p_{ext} = \frac{T}{r}$, where T is the surface tension or tensile force per unit length of the vessel and p_{ext} is a constant external pressure. For a linearly elastic material this tensile force becomes $T = \frac{E\epsilon_1}{1-\nu^2}h$ where E is the elastic modulus, ν^2 is Poisson's ratio and h is wall thickness. Derived from this is a simplified expression for the internal pressure

of the vessel [10]

$$p = p_{ext} + \beta(\sqrt{A} - \sqrt{A_0}) \tag{5.1}$$

where A_0 is a reference cross-sectional area corresponding to the area when $p = p_{ext}$. β is a stiffness parameter characterizing the mechanical properties of the vessel and is given by

$$\beta = \frac{\sqrt{\pi} h_0 E}{(1-\nu^2)A_0} \tag{5.2}$$

Differentiation of this expression gives the compliance:

$$C = \frac{2\sqrt{A}}{\beta} \tag{5.3}$$

5.2 Governing equations

The variables describing blood flow in a compliant vessel are pressure, flow and cross-sectional area (p, q, A). The system is however fully described by two independent system variables. Choosing (p,q) as the independent state variables, the 1D governing equations for flow in compliant vessels are [6, 10]

$$\frac{\partial p}{\partial d} + \frac{1}{C}\frac{\partial q}{\partial z} = 0 \tag{5.4a}$$

$$\frac{\partial q}{\partial d} + \delta\frac{\partial}{\partial z}(\frac{q^2}{A}) + \frac{A}{\rho}\frac{\partial p}{\partial z} = -2\pi(\gamma+2)\frac{\mu}{\rho}\frac{q}{A} \tag{5.4b}$$

which can be written in matrix notation as:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{M}(\boldsymbol{u})\frac{\partial \boldsymbol{u}}{\partial z} = \boldsymbol{b}(\boldsymbol{u}), \boldsymbol{u} = \begin{bmatrix} p \\ q \end{bmatrix} \tag{5.5}$$

5.3 Discretization

In the vascular1Dflow software the governing equations are discretized using the MacCormack numerical scheme. This is an explicit numerical scheme in which the state variables are predicted and then corrected. In the predictor step a forward difference is used to approximate the spatial derivative while in the corrector step a backward difference is used. Giving the following expressions for the predictor and corrector steps respectively [10]:

$$\hat{\boldsymbol{u}}_i = \boldsymbol{u}_i^n - \Delta t\left(\boldsymbol{M}(\boldsymbol{u}_i^n)\frac{\boldsymbol{u}_{i+1}^n - \boldsymbol{u}_i^n}{\Delta z} - \boldsymbol{b}(\boldsymbol{u}_i^n)\right) \tag{5.6a}$$

$$\boldsymbol{u}_i = \frac{1}{2}\left(\boldsymbol{u}_i^n + \hat{\boldsymbol{u}}_i - \Delta t\left(\boldsymbol{M}(\hat{\boldsymbol{u}}_i^n)\frac{\hat{\boldsymbol{u}}_{i+1}^n - \hat{\boldsymbol{u}}_i^n}{\Delta z} - \boldsymbol{b}(\hat{\boldsymbol{u}}_i^n)\right)\right) \tag{5.6b}$$

Being explicit this scheme is only conditionally stable. Its stability depends on the size of the time step which must be below a critical value Δt_{cr} which is the shortest time it takes for information to propagate between two neighbouring nodes in the system this is called the Courant-Friedrichs-Lewy (CFL) condition. The condition can be expressed as [10]:

$$CFL \geq (|u| + c)\frac{\Delta t}{\Delta x} \tag{5.7}$$

where the CFL number is 1. In the vascular1Dflow software the CFL number is given to the program and the appropriate time step length is calculated accordingly.

5.4 Lumped models

There is a long tradition of describing vascular networks using so called lumped models. In such models the vascular network rather than being described by propagating waves is descriped by a single pressure and a relation between pressure and flow.

5.4.1 The two element Windkessel model

The most basic application of the lumped model approach is the two element Windkessel (WK2) model. It was was originally quatified and popularized by Otto Frank [21]. The model consists of a compliance and a resistance coupled in parallel.The variables of the model are pressure (p) and inflow (q), which are related by the expression [6]:

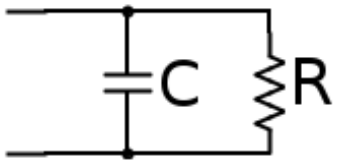


Figure 2: The WK2 represented by it's electric analog.

$$\frac{dp}{dt} + \frac{p - p_0}{RC} = \frac{q}{C} \tag{5.8}$$

where R is the resistance, and C is the compliance. p_0 is the peripheral pressure or reference pressure, if $p = p_0$ the flow is zero. The lumped model when prescribed an inflow Q(t) has the following general solution for the pressure [6]:

$$p(t) = p_{\text{init}} e^{-\frac{t-t_0}{\tau}} + \frac{1}{C} \int_{t_0}^{t} Q(t') e^{\frac{t'-t}{\tau}} \, \mathrm{d}t' \tag{5.9}$$

where Q is the inflow, p_{init} is the initial pressure and $\tau = RC$. Thus if there is no inflow to the system, the pressure will decay exponentially according to:

$$p(t) = p_{\text{init}} e^{-\frac{t-t_0}{\tau}} \tag{5.10}$$

this is used in section 8.4.2 where a pressure decay in a network is simulated and τ is determined by curve fitting the pressure decay function.

5.4.2 The three element Windkessel model

More complicated lumped models can be build by adding more elements either in series or parallel. Perhaps the most commonly used model though is the three element windkessel (WK3) model in which a second resistance is coupled in series to the two-element model, as shown in figure 3. As for the names of the two resistances the naming conventions can be different. The parallel resistance is often called the peripheral resistance since it represents the resistive effect of the microcirculation which is peripheral to the arterial network. The other resistance is often called the characteristic impedance (Z_c) of the model since it is somewhat analogous to the characteristic impedance of the aorta. To avoid confusion with the actual vessel characteristic impedance it is called R_f in this thesis. The governing equation of the model is given by:

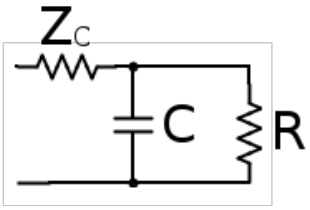


Figure 3: The WK3 represented by it's electric analog.

$$\frac{dp}{dt} + \frac{p - p_0}{\tau} = \frac{dq}{dt} R_f + \frac{q}{\tau}(R + R_f) \tag{5.11}$$

5.4.3 Impedance of Windkessel models

The input impedance of a windkessel model is frequency dependent and is generally a complex number. It is defined as:

$$Z_{in} = \frac{P}{Q} \tag{5.12}$$

where P and Q are the fourier transforms of the pressure and flow signals. The complex nature of the input impedance means that pressure and flow are out of phase. The expression for the WK2 can thus be found by taking the fourier transform of equation 5.8 which after reorganizing gives:

$$Z_{WK2} = \frac{P}{Q} = \frac{1}{i\omega C + \frac{1}{R}} \tag{5.13}$$

The amplitude ratio of a oscillating pressure and flow signal at angular frequency omega is therefore:

$$\frac{a_p}{a_q} = |Z_{WK2}| = \frac{1}{\sqrt{\frac{1}{R^2} + \omega^2 C^2}} \tag{5.14}$$

and the phase shift is:

$$\phi = \arctan(-\omega RC) \tag{5.15}$$

This means that if the flow is given by $q(t) = a_q \sin(\omega t)$ the resulting pressure is:

$$p(t) = a_q \sqrt{\frac{1}{R^2} + \omega^2 C^2} \sin(\omega t + \phi) \tag{5.16}$$

5.5 Boundary conditions

In order to make meaningful and useful simulations of vascular networks some form of boundary conditions must be prescribed at the open ends of the vascular tree. Essentially these boundary condititions must describe some form of relation between pressure and flow at the boundaries. This relation can be described by time varying functions, by differential equations or by combinations of the two. However to actually be used as boundary conditions the equations must be rewritten in terms of forward and backward travelling wave components. This is achieved in vascular1Dflow by using characteristic variables.

5.5.1 Characteristic variables

In order to handle boundary conditions correctly it is necessary to distinguish between forward and backward propagating pressure and flow waves. This can be done conveniently by expressing pressure and flow in terms of the Riemann-invariants or characteristic variables of the system ω_1 and ω_2, which represent pressure and flow waves travelling in positive and negative z-direction respectively. They are generally linear combinations of pressure and flow. To derive the relation between state variables the system matrix $\boldsymbol{M}$ is first decomposed and expressed in terms of its diagonal eigenvalue matrix, as well as its right and left eigenvector matrices so that

$$\boldsymbol{M} = \boldsymbol{R}\boldsymbol{\Lambda}\boldsymbol{L} \tag{5.17}$$

where:

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{5.18a}$$

$$\boldsymbol{R} = \begin{bmatrix} 1 & -1 \\ \frac{1}{Z_1} & -\frac{1}{Z_2} \end{bmatrix} \tag{5.18b}$$

$$\boldsymbol{L} = \frac{1}{Z_1 + Z_2} \begin{bmatrix} Z_1 & Z_1 Z_2 \\ Z_2 & -Z_1 Z_2 \end{bmatrix} \tag{5.18c}$$

and $\boldsymbol{R} = \boldsymbol{L}^{-1}$. Due to the scalability of eigenvectors, there are many different ways of expressing the right and left eigenvector matrices. The expressions given above though are the formulation currently implemented in vascular1Dflow. Z_1 and Z_2 are the characteristic impedances of forward and backward travelling waves respectively and the eigenvalues λ_1 and λ_2 are the wavespeeds. Using these definitions the equation system can be expressed as

$$\boldsymbol{L}\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{\Lambda}\boldsymbol{L}\frac{\partial \boldsymbol{u}}{\partial z} = \boldsymbol{L}\boldsymbol{b} \tag{5.19}$$

The characteristic variables are then introduced by

$$\frac{\partial \boldsymbol{\omega}}{\partial \boldsymbol{u}} = \boldsymbol{L} \tag{5.20}$$

And the equation system can be expressed in terms of the characteristic variables:

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + \boldsymbol{\Lambda}\frac{\partial \boldsymbol{\omega}}{\partial z} = \boldsymbol{L}\boldsymbol{b} \tag{5.21}$$

Integration of this expression from a reference state $\boldsymbol{u_0}$ to the current state $\boldsymbol{u}$ gives the state variable increments in terms of the characteristic variables

$$\Delta \boldsymbol{u} = \boldsymbol{R}(\boldsymbol{u}^n)\Delta \boldsymbol{\omega} \tag{5.22}$$

Expressions for the pressure and flow increments thus become

$$\Delta p = R_{11}\omega_1 + R_{12}\omega_2 \tag{5.23a}$$

$$\Delta q = R_{21}\omega_1 + R_{22}\omega_2 \tag{5.23b}$$

Boundary conditions are then handled in the program by rewriting the given pressure-flow relationship at the boundary in incremental form. Then substitute pressure and flow increments according to equations 5.23a and then solve the equation for ω_1 or ω_2 depending on the position of the boundary condition. If the boundary condition is located at the left boundary, with positive z direction going from left to right, then the variable going into the system is ω_1. The outgoing variable at the boundary can then be extrapolated from the previous timestep by computing the value of ω_2 at a distance of $\lambda_2 \Delta t$ from the boundary giving [6]

$$\omega_2 = \omega_2(t_n, \lambda_2 \Delta t) \tag{5.24}$$

5.5.2 Outgoing characteristic variable

The characteristic variable leaving the vessel at timestep n is generally unknown but can be estimated by using the characteristic from the previous timestep located at a distance into the vessel corresponding to the length that the outgoing wave travels in one timestep. The outgoing characteristic variables at the two boundaries at timestep $n+1$ are thus:

$$\omega_2 = \omega_2(z1 - \lambda_2 \Delta t, t^n) \text{ at } A1 \tag{5.25a}$$

$$\omega_1 = \omega_1(z2 - \lambda_1 \Delta t, t^n) \text{ at } A2 \tag{5.25b}$$

The values of ω are computed from the pressure and flow increments $\Delta \boldsymbol{u}$, as:

$$\omega_2 = \boldsymbol{l}_2 \Delta \boldsymbol{u} \tag{5.26a}$$

$$\omega_1 = \boldsymbol{l}_1 \Delta \boldsymbol{u} \tag{5.26b}$$

The point where the value of the outgoing characteristic is extrapolated from is generally located between the boundary node and the next node (node 0 and 1 on the A_1 boundary). The solution arrays therefore have to be interpolated to find values for P and Q at this point. Further some kind of difference method must be used to find $\Delta \boldsymbol{u}$.

The currently implemented method is given by the expression:

$$\Delta \boldsymbol{u}^{n-1} = \boldsymbol{u}(z_1 - \lambda_2 \Delta t, t^{n-1}) - \boldsymbol{u}(z_1, t^n) \tag{5.27}$$

5.5.3 Discretization of boundary conditions

Although the governing equations for lumped models such as the windkessel models are linear, when used as boundary conditions they form non-linear systems because of the nonlinearity of the equations describing the compliant vessels. This is the result of the vessels characteristic impedance being dependent on the vessel pressure. For example in the case of a simple terminal resistance where flow and pressure are related as $p = Rq$, when expressed in terms of forward and backward propagating pressures this becomes:

$$p_b + p_f = R(\frac{p_f}{Z_1} - \frac{p_b}{Z_2}) \tag{5.28}$$

which is nonlinear because of the values of the characteristic impedances Z_1 and Z_2 being dependent on the pressure. In vascular1Dflow the discretization is handled by introducing the introducing the incremental relation:

$$\Delta \boldsymbol{u} = \boldsymbol{R} \Delta \boldsymbol{\omega} \tag{5.29}$$

this also represents a linearization of the system since the value of $\boldsymbol{R}$ from timestep n is used as an approximation. The solution for a resistance boundary condition located on the distal end of a vessel thus becomes:

$$\omega_2 = \frac{-(1 - RR_{21})}{1 - RR_{22}} \tag{5.30}$$

The peripheral pressure and the initial pressure In vascular1Dflow it is possible to define a few different pressure which depending on how they are used can be though of as initial pressure, peripheral pressure or reference pressure. The initial pressure is a pressure that is prescribed at the beginning of simulation for all the vessels. Depending on how the boundary conditions are defined and implemented this pressure can also function as the peripheral pressure.

In the existing implementation of vascular1Dflow all the terminal boundary conditions act as pure reflectors. The reason for this is that the discretisations of the Windkessel BCs are purely incremental. This is also true for the resistance BC and the prescribed terminal reflection. The initial pressure then becomes identical to a peripheral pressure or reference pressure. The pressure at which there is no flow.

By discretizing the Windkessel BCs in a way so that the pressure and flow values from the previous time-step also are included and thus also retaining the peripheral pressure, the initial pressure gets a different meaning. If an initial pressure different from the peripheral pressure of the boundary condition is prescribed, the windkessel will at the beginning of the simulation generate a wave traveling into the vessel in order to satisfy the relation between pressure difference and flow given by $p - p_0 = Rq$. The details of the discretization of the WK2 is given in section 7.3.

This kind of discretization can not be done succesfully in the resistance BC, because here because since there is no compliance in this model the equilibrium between pressure and flow would be achieved immediately causing a step shaped wave that would cause numerical problems.

The interpretation of the different pressures thus depends on what kind of boundary conditions are used and how they are implemented.

Nonlinearity and predictor-corrector scheme Where the values of the components of $\boldsymbol{R}$ from timestep n are used when computing the pressure and flow increments $\Delta \boldsymbol{u}$ from timestep n to timestep n+1. This approximation is a possible source of inaccuracy however it could perhaps be reduced by using a predictor corrector procedure when computing the boundary conditions.

$$\hat{\boldsymbol{u}} = \boldsymbol{u}^n + \boldsymbol{R}(\boldsymbol{u}^n) \Delta \boldsymbol{\omega} \tag{5.31a}$$

$$\boldsymbol{u}^{n+1} = \frac{1}{2}(\hat{\boldsymbol{u}} + \boldsymbol{u}^n + \boldsymbol{R}(\hat{\boldsymbol{u}}) \Delta \hat{\boldsymbol{\omega}}) \tag{5.31b}$$

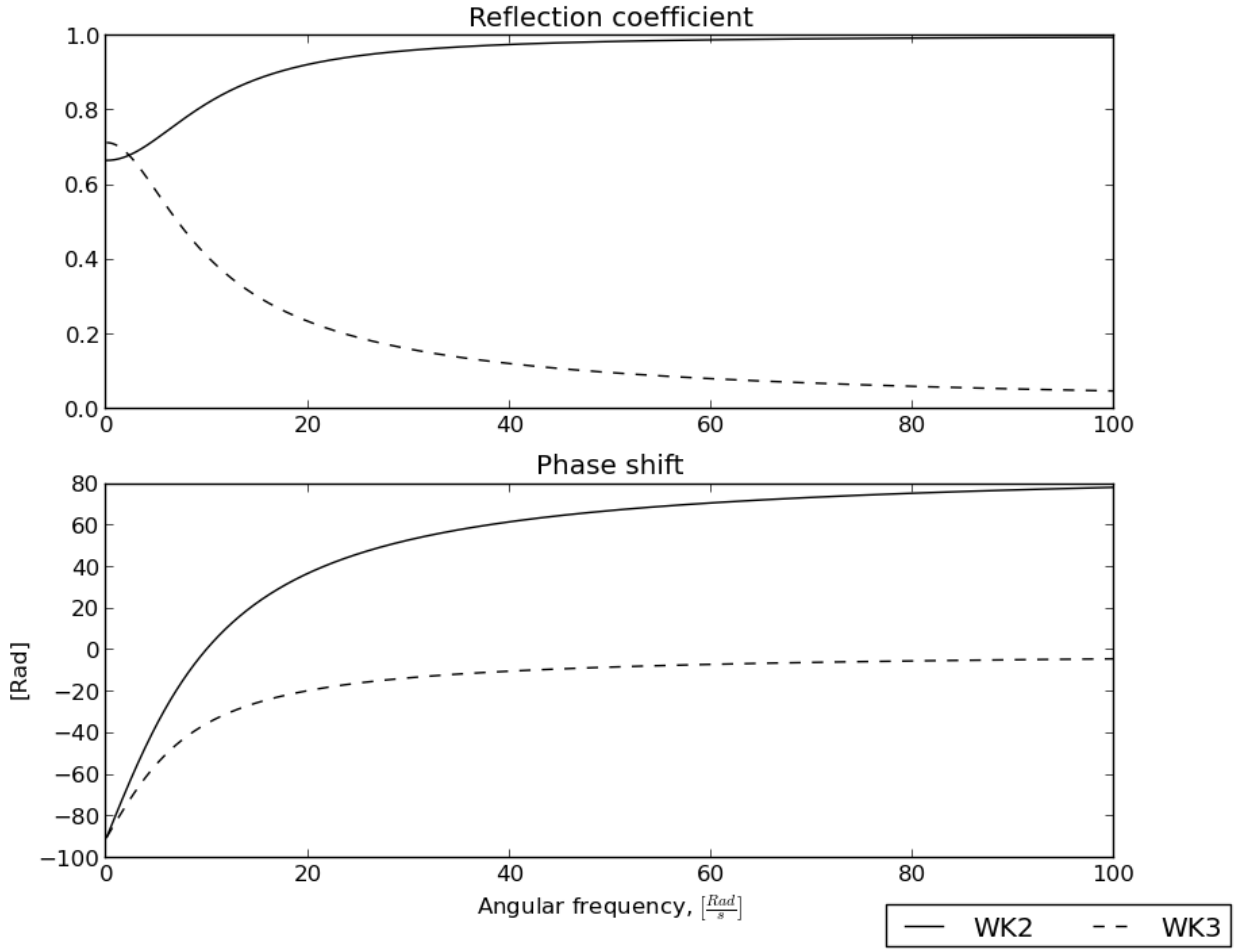


Figure 4: The reflection coefficient and phase shift for two- and three element windkessel models as a function of frequency

where $\Delta\boldsymbol{\omega}$ and $\Delta\hat{\boldsymbol{\omega}}$ are computed using values using $\boldsymbol{R}$ and $\hat{\boldsymbol{\omega}}$ respectively. It is however uncertain whether this is necessary due to the relatively small timesteps. In the current implementation a predictor-corrector scheme is not used for the boundary-conditions.

Reflection in boundary conditions. When the windkessel is used as a boundary condition for a vessel of characteristic impedance Z_c the reflection coefficient for a wave entering the windkessel becomes:

$$\Gamma_{WK2} = \frac{P_b}{P_f} = \frac{\frac{1}{Zc} - \frac{1}{Z_{WK2}}}{\frac{1}{Zc} + \frac{1}{Z_{WK2}}} = \frac{\frac{1}{Zc} - i\omega C - \frac{1}{R}}{\frac{1}{Zc} + i\omega C + \frac{1}{R}} \tag{5.32}$$

which is also complex and frequency dependent. Similarly the expression for the reflection coefficient of the three element windkessel model is:

$$\Gamma_{WK3} = \frac{Z_c(1 + i\omega R_c C) - R_c - R_f - i\omega R_c R_f C}{Z_c(1 + i\omega R_c C) + R_c + R_f + i\omega R_c R_f C} \tag{5.33}$$

A special and useful case of the WK3 is if the characteristic impedance of the vessel is matched to that of the windkessel, (ie $R_f = Z_c$. In this case $F \to 0$ when $\omega \to \infty$, meaning that higher frequencies are absorbed. The reflection coefficients with corresponding phase shifts for the two models are shown in figure 4 Thus for any periodic signal that can be expressed as a fourier series, an analytical solution can be obtained. This is utilized in section 7.3.3 where the numerical solution of the windkessel model is verified.

5.6 Network bifurcations

At bifurcations in the network some kind of physical model must be used to determine the distribution of pressure and flow in the vessels involved. The governing principles are conservation of mass an conservation of kinetic energy.

5.6.1 The governing equations

The pressure through the bifurcation is described by Bernoulli's principle so that:

$$p + \frac{1}{2}\rho v^2 = \text{constant} \tag{5.34}$$

along a streamline. By assuming a flat velocity profile so that $v = \frac{q}{A}$ one gets the governing equations [10]:

$$q_1 = q_2 + q_3 \tag{5.35a}$$
$$p_1 + \frac{\rho}{2}\frac{q_1^2}{A_1^2} = p_2 + \frac{\rho}{2}\frac{q_2^2}{A_2^2} \tag{5.35b}$$
$$p_1 + \frac{\rho}{2}\frac{q_1^2}{A_1^2} = p_3 + \frac{\rho}{2}\frac{q_3^2}{A_3^2} \tag{5.35c}$$
$$\tag{5.35d}$$

Linear model. For flows approaching zero the nonlinear terms in equation 5.35 vanish and the equations become:

$$q_1 = q_2 + q_3 \tag{5.36a}$$
$$p_1 = p_2 \tag{5.36b}$$
$$p_1 = p_3 \tag{5.36c}$$

where the subscript 1 denotes the mother vessel and subscripts 2 and 3 the daughter vessels. This simple relation is accurate at low flow rates.

5.6.2 Reflection and transmission coefficients

Using this linear model allows us to think of the boundary condition as a sudden change in impedance which causes waves to be reflected and transmitted according to reflection and transmission coeficcients. These are obtained by expressing pressure and flow as a superposition of forward and backward travelling waves so that:

$$p = p_f + p_b \tag{5.37a}$$
$$q = q_f + q_b \tag{5.37b}$$

and using the vessels characteristic impedances to relate pressure and flow:

$$Z_c = \frac{p_f}{q_f} = -\frac{p_b}{q_b} \tag{5.38}$$

Inserting this into the equations in 5.36 gives the system:

$$\frac{1}{Z_1}(p_{f,1} - p_{b,1}) = \frac{1}{Z_2}(p_{f,2} - p_{b,2}) + \frac{1}{Z_3}(p_{f,3} - p_{b,3}) \tag{5.39a}$$
$$p_{f,1} + p_{b,1} = p_{f,2} + p_{b,2} \tag{5.39b}$$
$$p_{f,1} + p_{b,1} = p_{f,3} + p_{b,3} \tag{5.39c}$$

which can then be solved for the reflection coefficient $\Gamma = \frac{p_b}{p_f}$. A wave travelling from the mother vessel (vessel 1) into the bifurcation is therefore reflected according to the reflection coefficient:

$$\Gamma = \frac{\frac{1}{Z_1} - (\frac{1}{Z_2} + \frac{1}{Z_3})}{\frac{1}{Z_1} + \frac{1}{Z_2} + \frac{1}{Z_3}} \tag{5.40}$$

and accordingly transmitted, into the two daughter vessels with an amplitude described by the transmission coefficient $T = 1 + \Gamma$.

5.7 The reservoir-wave approach

An experimental approach to understanding wave-propagation in vascular network is the reservoir-wave approach. This tries to unite Windkessel, or reservoir, models of vascular networks, with wave propagation models. The idea is that pressure and flow throughout a vascular network can be separated into a reservoir component, which is equal for the whole network and a wave component also called excess pressure and flow. For a pressentation of the concept see Tyberg et al. [19]. If the network has a total volume compliance of C_{tot} the reservoir pressure increases according to:

$$\frac{dP_{res}}{dt} = \frac{Q_{in} - Q_{out}}{C_{tot}} \tag{5.41}$$

Where P_{res} is the reservoir pressure of the arterial network. The outflow is determined by:

$$Q_{out} = \frac{P_{res} - P_0}{R_{res}} \tag{5.42}$$

Which is equivalent to a two element windkessel model, which has the analytical solution given in equation 5.9.

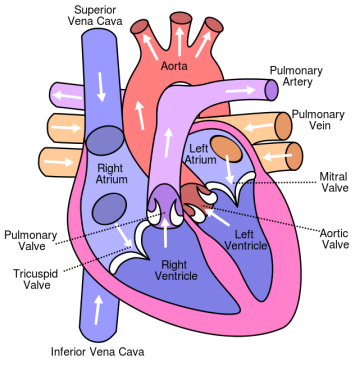
In an experimental setting pressure and flow can be measured at the aorta, but Q_{out}, C_tot, P_0 and R_{res} are generally unknown and must be determined from the governing equations and the measured data. After having determined P_{res}, the reservoir of excess pressure, the wave pressure is determined as $P_{\text{wave}} = P - P_{res}$. It is shown by Tyberg et al. [19], that the resulting wave pressure when computed at the aortic root is proportional to the inflow. It can therefore be seen as the driving force of the inflow or as a pressure gradient between the aortic root and the reservoir which is imagined to be located some distance from the aorta. Further the wave pressure can be subject to wavesplitting, it is shown by Tyberg et al. that the backward component of this wavepressure is almost non-existing at the aortic root.

In section 8.4.4, the reservoir-wave approach is applied to the numerically simulated arterial network along with a varying elastance model. In this case the determination of the reservoir pressure is very straightforward, since both inflow and outflow are known. And the compliance can be computed from vessel data.

6 The left ventricle and varying elastance

6.1 Physiological background - the ventricle

During each heart cycle the left ventricle receives oxygenated blood from the pulmonary circulation via the left atrium and ejects it into the systemic circulation through the ascending aorta. The blood flow is driven by the periodic contraction of muscle fibers in the heart wall. The flow between the left atrium and the left ventricle is controlled by the mitral valve and blood flow between the left atrium and ascending aorta by the aortic valve, these valves only allow blood to flow in one direction.



6.1.1 Phases of the ventricular cardiac cycle

The pumping cycle of the ventricle can be divided into four distinct phases. First are the two phases of diastole which starts with the closing of the aortic valve. First there is an isovolumic relaxation phase. In this phase both mitral and aortic valves are closed and there is a relaxation of the heart muscles giving a decrease in pressure inside the ventricle. When the pressure has decreased sufficiently that there is a positive pressure difference between the left atrium and the left ventricle, the mitral valve opens and the next phase starts. This next phase is a passive filling phase. The ventricle is now filled with blood forced by the blood pressure in the pulmonary veins.

Next are the two phases of systole. As the muscles in the heart wall start to contract there is an increase in pressure. The mitral valve closes and systole starts. There is an isovolumic contraction phase. In this phase pressure inside the ventricle increases, the aortic valve opens and there is an ejection phase in which blood flows out of the ventricle and into the ascending aorta. At the end of systole the aortic valve closes and the heart cycle is complete.

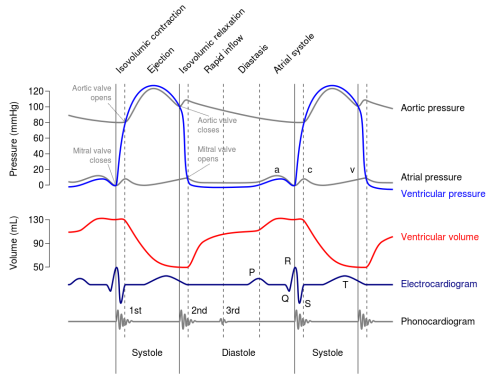


Figure 6: The four phases of the cardiac cycle are shown in Wiggers diagram

6.1.2 Mechanisms of heart contraction

The contractions are controlled by electric signals which trigger a series of chemical reactions activating the sliding mechanism of actin and myosin that cause the muscle fibers to contract. This in combination with the elasticity of the heart tissue makes the heart able to function as a pump. When the heart muscle is relaxed the ventricle acts as an elastic chamber that can be filled with blood from the atrium. It can then contract in order to pump blood into the aorta.

6.1.3 Heart pre-load

Higher venous pressure leads to a larger ventricular volume at the end of diastole (EDV). This again leads to muscle fibers in the heart wall being more stretched out. The initial stretch of the muscle fibers is called the heart pre-load. According to the Frank-Starling law of the heart, higher preload results in a larger stroke volume and a greater work done by the heart [9] when all other conditions are held constant.

6.1.4 Heart after-load and arterial reflections

The forces that opposes this pumping of blood is called the afterload on the heart. A higher afterload increases ventricular pressure during systole and reduces outflow. There two main components or sources of the afterload are the characteristic impedance of the aorta and pressure wave reflections from the periphery. The characteristic impedance of the aorta is the only thing that impedes ventricular outflow in the beginning of systole. The effect of reflections is slightly delayed, the main part of the reflection comes during the second half of systole.

The heart afterload is thus characterized by the input impedance measured at the proximal end of the aorta i.e. the pressure-flow ratio; due to the complex nature of the vascular system with its many bifurcations and resulting reflections, this is a very complex relationship. However it has been shown that it is well described by a three element windkessel model [21]

6.1.5 Contractility and load-dependence

The contractility of the heart is a concept related to the ability of the heart itself, independent of pre- and afterload, to contract and generate pressure. In other words how powerful the heart is. A commonly used measure of the contractility is its end-systolic pressure-volume relationship (ESPVR). This is the relation between pressure and fluid volume of the ventricle at the end of systole, marked by the closing of the aortic valve. Various in-vitro experiments where a varying afterload is applied to the same heart shows that this relationship is approximately linear for a large range of afterloads. However for large variations of after-load the relationship becomes non-linear. This shows that the ESPVR is a good measure for the contractility of a heart, but not perfect as it is somewhat dependent on load [11, 13].

ESPVR is closely related, but not identical, to the concept of maximum elastance (E_{max}) introduced in the following section. The ESPVR is usually measured at the closing of the aortic valve whereas the maximum elastance is the measured at the point on pressure-volume loop where the ration of pressure and volume is the highest, in the upper left corner of the p-v loop.

6.1.6 Pressure-volume loops

A common way of presenting data from the ventricular pumping cycle is through the socalled p-v loop, in which ventricular pressure and volume are plotted against each other with volume on the x-axis and pressure on the y-axis. This approach is particularly useful when comparing multiple results where one or more parameter has been changed. Typically preload and afterload are varied to determine the effects on the shape of the loops. When a curve is drawn through points of identical time on several p-v loops this is called an isochrone.

6.2 Varying elastance

The varying elastance concept simplifies the system of elasticity and active contraction of the ventricle by viewing the heart as an elastic chamber where the elastance is allowed vary with time. The classic definition of varying elastance was provided by Suga and Sagawa [16] and is defined as the instantaneous relationship between pressure and fluid volume volume in the ventricle given by:

$$E(t) = \frac{p}{V - V_0} \tag{6.1}$$

where the constant V_0 is a referance volume that can be interpreted as the fluid volume at zero pressure. When measured continously throughout a heart cycle the elastance becomes a function of time. The shape and magnitude of this curve serves as a measure of the contractility of the individual heart. It was initially conceived as a load independent measure of contractility. A multitude of studies have shown that this is not true [1, 2, 8, 20]. Typically for variations in preload cause convex isochrones during systole [8]. Little research has been made however on the effect of changes in afterload on isochrones [18].

Load dependence. An elastance defined entirely as a function of time implies that isochrone lines on the p-v loop are linear for variations in load. Due to load dependence this is generally not the case. Rather than having a concrete physical interpretation the fixed and time varying parameter can be seen as curve fitting parameters to these curved isochrones. The interpretation of V_0 as the zero-pressure volume is therefore invalidated and it can be seen more as a curve fitting parameter.

A few modifications have been proposed to improve on the varying elastance model in how well it can be fitted to isochrones, e.g. six different models are compared in Lankhaar et al. [8]. However few of them are suited to predictive simulations as they often have to many parameters.

Time-varying intercept One such modification is to allow the volume axis intercept to vary with time so that $V_0 = V_0(t)$. In a simulation context this requires the construction of two independent time varying functions which adds to complexity. The model is shown by Lankhaar et al. [8] to be an improvement over the standard varying elastance model, without too much added complexity. The time varying intercept is however difficult to interpret physiologically.

Source resistance. Another simple modification of the definition of the elastance can be made that ensures the convexity of the ESPVR while keeping the elastance function itself unchanged. A modification to the definition of the elastance originally proposed by Shroff and Weber [14] is made by introducing a source resistance, also called a systolic resistance, of the ventricle that is proportional to the pressure. So that it becomes:

$$E(t) = \frac{p}{(V - V_0)(1 - KQ)} \tag{6.2}$$

This approach only introduces one extra parameter, it does however introduce a non-linearity in the equation which must be handled in the numerical scheme. This modification has the benefit that it provides some load-dependence in the model without a significant increase in complexity.

No data was found for the value of the source resistance coefficient for the human heart. The value used by Shroff in the original paper was K = 1500 $\frac{s}{m^3}$ but this value was used for dogs [14]. The same same value was used by Lankhaar et al. [8] where the model was evaluated and compared to other heart models with regards to how well they could be fitted to experimental data from sheep.

The benefit of the source resistance model is that it introduces a load dependence to the varying elastance model without introducing any additional time-varying parameters. Thus the elastance function can be used as a load-independent parameters that produces load dependent results. In the study by Lankhaar et al. simulations using the source resistance did not show any improvement over the classical varying elastance model in terms of accuracy. However as is also shown by Lankhaar et al. that the actual fitted time varying elastance function becomes somewhat different when the systolic resistance is used. E_{max} is approximately 10 % larger and the curve has a different shape. This was not taken into account when implementing the model thus making the the use of the source resistance in the current implementation invalid. In some of the simulations this error was compensated for by increasing the E_{max}.

Elastance function. When using the varying elastance concept in a simulation context, the elastance must be precribed to the system by some kind of analytic function. It is not very important what kind of function is used for the elastance as long as it roughly resembles the curves found from experiments and is easily scalable in amplitude with respect to minimum and maximum elastance, and in duration as prescribed by the time to peak elastance.

Such a funtion is the double-Hill function suggested by Stergiopolous [13]:

$$E(t) = (E_{max} - E_{min})\alpha \left[\frac{(\frac{t}{\alpha_1 T})^{n_1}}{1 + (\frac{t}{\alpha_1 T})^{n_1}} \right] \left[\frac{1}{1 + (\frac{t}{\alpha_2 T})^{n_2}} \right] + E_{min} \tag{6.3}$$

Where E_{max} and E_{min} are maximum and minimum elastances, α normalizes the double-Hill function so that it has a maximum value of 1.0, T is the heart cycle period and $\alpha_1, \alpha_2, n_1 and n_2$ are shape parameters given as:

Table 1

α_1	α_2	n_1	n_2	α
0.303	0.508	1.32	21.9	1.672

Table 2: The parameters that determine the shape of the time varying elastance function.

By keeping the ratio $\frac{\alpha_1}{\alpha_2}$ and parameters n_1 and n_2 constant, a scalable version of this equation is obtained based on the time to peak t_p:

$$E(t) = (E_{max} - E_{min})\alpha \left[\frac{(\frac{t}{0.708t_p})^{1.32}}{1 + (\frac{t}{0.708t_p})^{1.32}} \right] \left[\frac{1}{1 + (\frac{t}{1.187t_p})^{21.9}} \right] + E_{min} \tag{6.4}$$

This formulation makes it possible to set t_p and T independently. The three parameters determining the contractility of the ventricle are thus E_{max}, E_{min} and t_p. A plot of the elastance function is shown in figure 7.

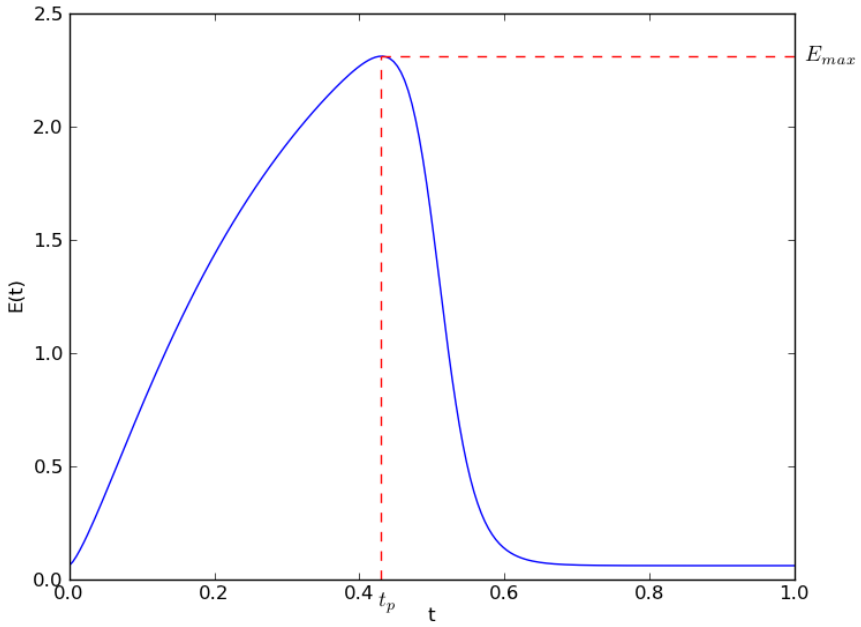


Figure 7: The time-varying elastance function as defined in [13], $E_{max} = 2.31$, $t_p = 0.43$

6.2.1 Arterial stiffness

As mentioned most of the reflected pressure pulse arrives at the heart after the closing of the aortic valve thus not representing a heart afterload. One could say that the body is fine-tuned in this way; the stiffness and geometry of the arteries determine the wavespeed of the pressure pulses so that most of the reflected wave comes during diastole [12]

The reflection itself is wanted because it increases flow in the coronary arteries. This is important since the subendocardial vessels are compressed, and therefore not able to receive blood during systole [12]. The heart is therefore dependent on the heightened pressure during diastole, directly after the dicrotic notch, in order to receive enough oxygen and nutrients.

In people with increased arterial stiffness, such as is often the case in elderly subjects [12], there is a corresponding increase in the wave speed. This causes the reflected wave to arrive

sooner causing a larger part of the reflected wave to arrive at the ventricle during systole. This again causes impaired blood perfusion in the coronary arteries as well as increasing the load on the heart and reducing the total blood flow.

6.3 Aortic and mitral valves

There are four valves in the heart all situated in a plane. The two valves connected to the left ventricle are the aortic and mitral valve. The aortic and mitral valves are essential to the function of the left ventricle as a pump.

The two valves are similar in function, only allowing blood to flow in one direction. Anatomically they differ in that the mitral valve has two cusps whereas the aortic valve has three. It is the cusps that provide the mechanism that controls the blood blow. When the valve is closed they form a tight seal allowing no blood to pass through the valve. In the open state the cusps separate allowing blood to flow through the valve. It is the tranvalvular pressure gradient that causes the cusps to separate or come together.

6.3.1 Pressure gradient across the aortic valve

The viscous resistance of blood passing through the valve is negligible [4, 5] and assuming that the aortic root has the same cross-sectional area as the left-ventricular outflow tract there is no net pressure difference due to conservation of kinetic energy.

The main causes of the pressure gradient are therefore the inertia of blood passing through the valve and the energy lost in the turbulent vortices distal to the valve. In a healthy aortic valve the pressure gradient is very small during most of systole it is only during the beginning of systole that the pressure gradient becomes significant because of the rapid acceleration of flow. In a stenosed valve the valve is not able to fully open, this can cause a large pressure gradient thus reducing the amount of blood that the heart is able to eject and increase the load on the heart. The pressure gradient across the valve is given by the expression:

$$\Delta p = Bq|q| + L\frac{dq}{dt} \tag{6.5}$$

where B is a Borda-Carnot resistance parameter and L is an inertance parameter. Even if the pressure gradient is small it is important because it is responsible for opening and closing the valve. In section 6.4 a dynamic valve model is presented which relates the pressure to the opening and closing of the valve.

Turbulent resistance. The pressure loss due to turbulence can described analytically using the Borda-Carnot effect. Blood flow through a cardiovascular valve can be compared with flow in a pipe where the flow suddenly has to pass through a orifice with a smaller cross-sectional area than the pipe itself. As the fluid jet exits the valve it is separated from the vessel walls and turbulent vortices are formed. It is only after a certain distance, called the recovery distance, that the flow goes back to being a laminar tube flow. Some energy is lost in the turbulent vortices distal to the valve and the turbulent resistance is given by:

$$B = \frac{\rho}{2A_{\text{eff}}} \tag{6.6}$$

where A_{eff} is a computed effective area given by:

$$A_{\text{eff}} = \frac{1}{\frac{1}{A_{eo}} - \frac{1}{A}} \tag{6.7}$$

where A_eo is the effective orifice area. This effective orifice area is generally different from the actual geometrical opening area. In a fully open healthy valve the A_{eo} is approximately equal to the aortic cross sectional area and there is almost no resistance.

In Mynard et al. [7] and Sun et al. [17] the expression used for the turbulent resistance was:

$$B = \frac{\rho}{2A_{\text{eff}}^2} \tag{6.8}$$

where A_{eff}, rather than being defined as in equation 6.7, is computed directly from an interpolation between minimum and maximum aortic valve area. Using this definition is wrong and misleading because then a fully open valve still only has $A_{eff} = A$, where A is the aortic cross-sectional area. This again corresponds to an effective orifice are $A_{eo} = 0.5A$ which is more realistic in a stenosed valve.

Inertia in the aortic valve. As shown by Garcia et al. [4, 5] the orifice introduces an inertia term that is related to the orifice area in a similar way as the Borda-Carnot resistance, approaching infinity as the valve closes and becoming zero when the valve is completely open. The major difference is that it is inversely proportional to the squareroot of the effective area rather than the square. The expression is:

$$L = 4\pi\rho\sqrt{\frac{1}{A_{eo}} - \frac{1}{A}} \tag{6.9}$$

This is also a deviation from the expression used in the valve model by Mynard et al. The expression for inertance used by Mynard et al and Sun et al is correctly assumed to approach infinity as the valve closes and the effective orifice area approaches zero. In this formulation however the inertance is quantified by an effective length l_{eff}, which is not very well defined, according to the expression:

$$L = \frac{\rho l_{eff}}{A_{\text{eff}}} \tag{6.10}$$

also here with the different definition of A_{eff}. Since the expression given by Garcia et al. is better argumented for, this is what is used in the implementation of the varying elastance model.

6.4 Valve model

In the simplest case the aortic valve can be modelled as being either completely open for a positive pressure difference or completely closed for a negative pressure difference. The effective area is then $A_{eff} = A_{eff,max}$ when it is open giving a constant resistance and zero when it is closed giving an infinite resistance. The latter implying zero flow and complete reflection of incoming pressure and flow waves. This type of aortic valve model can be considered the fluid flow equivalent of a perfect diode.

This approach was implemented initially with the varying elastance model and worked fairly well, but had the problem that the valve would flicker between the open and closed position giving unwanted results. This unwanted behaviour could likely have been avoided by introducing a threshold pressure for closing so that the closing criteria requires:

$$\Delta p = p_{\text{ventricle}} - p_{\text{aorta}} < -\Delta p_{open} \tag{6.11}$$

where Δp_{open} is somwhere in the region of 1-5 mmHg. However to get a better representation of the aortic valve a dynamic valve model was implemented.

Dynamic valve model. To get a more realistic representation of the aortic valve one can use a dynamic valve model, such as the one developed by Mynard et al. [7], where the effective area is allowed to vary continously depending on the pressure difference. This model uses an opening parameter ζ so that for a completely open valve $\zeta = 1$, for a closed valve $\zeta = 0$. The effective area is then interpolated between the two extremes so that:

$$A_{eo} = [A_{eo,max}(t) - A_{eo,min}(t)]\zeta(t) + A_{eo,min} \tag{6.12}$$

which has been modified from Mynard et al. by replacing A_{eff} with A_{eo}. The actual effective area is the computed from the EOA and the aortic area according to eq 6.7. For a functioning healthy aortic valve the minimum EOA is $A_{eo,min} = 0$. In the case of regurgitation $A_{eo,min} > 0$ meaning that some blood flows back into the heart during systole. The maximum and minimum orifice areas are given by the parameters M_{st} and M_{rg} so that:

$$A_{eo,max} = M_{st}A \tag{6.13a}$$
$$A_{eo,min} = M_{rg}A \tag{6.13b}$$

Further the rate of opening, which is the time derivative of ζ, is related to the instantaneous pressure:

$$\frac{d\zeta}{dt} = \begin{cases} (1-\zeta)K_{vo}(\Delta p - \Delta p_{open} & \Delta p > \Delta p_{open} \\ \zeta K_{vc}(\Delta p - \Delta p_{close}) & \Delta p < \Delta p_{close} \\ 0 & \Delta p_{close} < \Delta p < \Delta p_{open} \end{cases} \tag{6.14}$$

Where K_{vo} and K_{vc} are the opening and closing rate coefficients respectively. Δp_{close} and Δp_{close} are the threshold pressure differences for opening and closing.

7 Methods and implementations

7.1 Implementation of the varying elastance model as a boundary condition

The varying elastance boundary condition is implemented by combining the varying elastance model from equation 6.2 with the expressions for valve dynamics and transvalvular pressure gradient given in equations 6.14 and 6.5. Also total flow is related to volume by an integral equation. The whole ventricle system can thus be described as a coupled system of the seven variables $q_m, \zeta_m, p_v, V, \zeta_{ao}, p$ and q, where the m subscript refers to the mitral valve and p_v is the ventricular pressure. This whole system is described by the five non-linear differential equations repeated below:

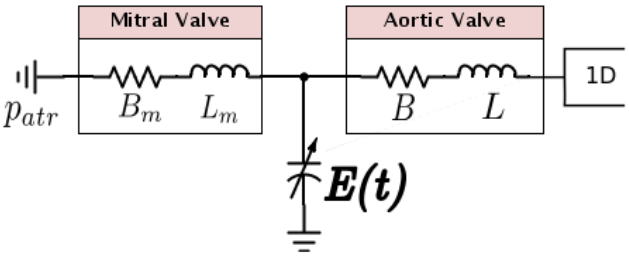


Figure 8: An approximate electrical analog to the varying elastance heart model.

$$p_{atr} - p_v = B_m q_m |q_m| + L_m \frac{dq_m}{dt} \tag{7.1a}$$

$$\frac{d\zeta_{mit}}{dt} = \begin{cases} (1-\zeta_m)K_{vo}(\Delta p_m - \Delta p_{m,\text{open}}) & \text{if } \Delta p_m > \Delta p_{m,\text{open}} \\ \zeta_{mit}K_{vc}(\Delta p_m - \Delta p_{m,\text{close}}) & \text{if } \Delta p_m < \Delta p_{m,\text{close}} \\ 0 & \text{if } \Delta p_{m,\text{close}} < \Delta p < \Delta p_{m,\text{open}} \end{cases} \tag{7.1b}$$

$$p_v = E(V - V_0)(1 - Kq) \tag{7.1c}$$

$$\frac{d\zeta_{ao}}{dt} = \begin{cases} (1-\zeta_{ao})K_{vo}(\Delta p - \Delta p_{open}) & \text{if } \Delta p > \Delta p_{open} \\ \zeta_{ao}K_{vc}(\Delta p - \Delta p_{close}) & \text{if } \Delta p < \Delta p_{close} \\ 0 & \text{if } \Delta p_{close} < \Delta p < \Delta p_{open} \end{cases} \tag{7.1d}$$

$$p_v - p = Bq|q| + L\frac{dq}{dt} \tag{7.1e}$$

as well as the integral equation describing the relation between ventricular volume and flow:

$$V(t) = V(t_0) + \int_{t_0}^{t} q_m - q\, dt \tag{7.2}$$

To avoid having to solve the entire system simultaneously the simplification was made that, equations 7.1b and 7.1d, the equations describing the valve dynamics, are solved explicitly using forward differences giving the expression

$$\zeta^{n+1} = \zeta^n + \begin{cases} (1-\zeta^n)K_{vo}((\Delta p)^n - \Delta p_{open}) & \text{if } (\Delta p)^n > \Delta p_{open}) \\ \zeta^n K_{vc}((\Delta p)^n - \Delta p_{close}) & \text{if } (\Delta p)^n < \Delta p_{close} \\ 0 & \text{if } \Delta p_{close} < (\Delta p)^n < \Delta p_{open} \end{cases} \tag{7.3}$$

where $(\Delta p)^n$ is the transvalvular pressure difference at timestep n given by $(\Delta p)^n = p_v$. The values of B_m^{n+1}, L_m^{n+1}, B^{n+1} and L^{n+1} are then determined by equations 6.6 and 6.9 as well as 6.12. This leaves a system of three equations to be solved 7.1a, 7.1c and 7.1e.

Coupling between the equations. The coupling of the equations depends on the opening state of the ventricles. It is only in the case when both valves are open that the entire system is coupled and has to be solved simultaneously. If both the mitral and aortic valve are healthy and fully functioning this is never the case. The aortic valve is closed during diastole and the mitral valve is closed during systole. There are also short periods between systole and diastole where both valves are closed. When used as a boundary condition for a vascular network it is primarily systolic conditions that are of interest. In that case an accurate solution of equations 7.1c and 7.1e are most important. The diastolic filling is then only important as the determinant of pre-load. Still the choice was made to implement a complete system that included diastolic filling.

7.1.1 Discretization

The three equations were discretized at timestep n+1. Aortic pressure and flow increments were expressed in terms of the characteristic variables and the components of the $\boldsymbol{R}$ matrix so that:

$$p^{n+1} = p^n + \Delta p = p^n + \omega_1 + \omega_2 \tag{7.4a}$$

$$q^{n+1} = q^n + \Delta q = q^n + R_{21}\omega_1 + R_{22}\omega_2 \tag{7.4b}$$

where the components R_{11} and R_{12} are ommited for simplicity since they are both equal to one in the current implementation of *vascular1Dflow*. For R_{21} and R_{22} the values from timestep n are used as an approximation. The characteristic variable ω_2 is extrapolated from the field according to equation 5.24. This leaves ω_1 as an unknown variable. Since the system has to be solved for this characteristic variables, which is an increment, the choice was made to solve for the increments of the other variables as well. Equation 7.1c then becomes:

$$p_v^n + \Delta p_v = E^{n+1}(V^n + \Delta V - V_0)(1 - K(q^n + R_{21}\omega_1 + R_{22}\omega_2)) \tag{7.5}$$

where the volume increment is determined from the mitral and aortic flows by numerical evaluation integration using the trapezoidal rule:

$$V^{n+1} = V^n - (q^n - q_m^n + \frac{1}{2}(\Delta q - \Delta q_m^n)\Delta t \tag{7.6}$$

The time derivatives in equations 7.1a and 7.1e are discretized using second order backward differences so that:

$$(\frac{dq_m}{dt})^{n+1} = \frac{3\Delta q_m - q_m^n + q_m^{n-1}}{2\Delta t} \tag{7.7}$$

for the mitral flow and

$$(\frac{dq}{dt})^{n+1} = \frac{3(R_{21}\omega_1 + R_{22}\omega_2) - q^n + q^{n-1}}{2\Delta t} \tag{7.8}$$

for the aortic flow. Remaining to be solved is now a system of three discretized non-linear equations where the variables are the three increments $\boldsymbol{x} = [\Delta q_m, \Delta p_v, \omega_1]$. The equations are:

$$f_1(\boldsymbol{x}) = B_m^{n+1}(q_m^n + \Delta q_m)|q_m^n + \Delta q_m| + L_m^{n+1}\frac{3\Delta q_m - q_m^n + q_m^{n-1}}{2\Delta t} - p_{atr} + p_v^{n+1} = 0 \tag{7.9a}$$

$$f_2(\boldsymbol{x}) = E^{n+1}(V^n - (q^n - q_m^n + \frac{1}{2}(R_{21}\omega_1 + R_{22}\omega_2 - \Delta q_m))\Delta t - V_0)(1 - K(q^n + R_{21}\omega_1 + R_{22}\omega_2)) - p_v^n - \Delta p_v = 0 \tag{7.9b}$$

$$f_3(\boldsymbol{x}) = B^{n+1}(q^n + \Delta q)|q^n + \Delta q| + L^{n+1}\frac{3(R_{21}\omega_1 + R_{22}\omega_2) - q^n + q^{n-1}}{2\Delta t} - p_v^n - \Delta p_v + p^n + \omega_1 + \omega_2 \tag{7.9c}$$

7.1.2 Iterative solution

Being non-linear the equations can not be solved directly but must be solved iteratively. The equations were solved for each timestep by the Newton-Raphson method given by:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - (\boldsymbol{J}_i)^{-1}\boldsymbol{f}_i \tag{7.10}$$

where $\boldsymbol{J}$ is the Jacobian matrix. The increments from the previous timestep were used as the starting estimate $\boldsymbol{x}_0$ and the Jacobian matrix was updated for each iteration. The iterations are terminated when the relative error given by:

$$\epsilon = \frac{||\boldsymbol{x}^i - \boldsymbol{x}^{i-1}||}{||\boldsymbol{x}^{i-1}||} \tag{7.11}$$

is less than than the tolerance value which was set to $\epsilon_{tol} = 0.0001$. The Jacobi matrix is given by:

$$\boldsymbol{J} = \begin{bmatrix} a_1 & 1 & 0 \\ a_2 & -1 & a_3 \\ 0 & -1 & a_4 \end{bmatrix} \tag{7.12}$$

where:

$$a_1 = 2B_m^{n+1}(q_m^n + \Delta q_m)\text{sign}(q_m^n + \Delta q_m) + \frac{3L_m^{n+1}}{2\Delta t} \tag{7.13a}$$

$$a_2 = \frac{1}{2}E^{n+1}\Delta t(1 - K(q^n + R_{21}\omega_1 + R_{22}\omega_2)) \tag{7.13b}$$

$$\begin{aligned} a_3 = &-\frac{1}{2}E^{n+1}R_{21}\Delta t(1 - K(q^n + R_{21}\omega_1 + R_{22}\omega_2)) \\ &- E^{n+1}R_{21}K(V^n - (q^n - q_m^n + \frac{1}{2}(R_{21}\omega_1 + R_{22}\omega_2 - \Delta q_m))\Delta t - V_0) \end{aligned} \tag{7.13c}$$

$$a_4 = 2B^{n+1}(q^n + R_{21}\omega_1 + R_{22}\omega_2)\text{sign}(q^n + R_{21}\omega_1 + R_{22}\omega_2) + \frac{3R_{21}L^{n+1}}{2\Delta t} + 1 \tag{7.13d}$$

Numerical problems. When a valve is nearly closed, the B and L values of this valve approach infinity. The Jacobi matrix is still invertible so the system can theoretically be solved, but the system becomes very ill-conditioned and thus difficult to solve as the valve approaches closing. At full closure the system is completely invalid, the effective orifice area is zero thus causing division by zero in the expressions for B and L. At the same time the flow through the valve is zero so that there is no coupling between what is happening on either sides of the valve. In the case of the aortic valve being closed this means a complete reflection of incoming waves (i.e. $\omega_1 = \omega_2$). In the cases where one or both valves are completely or nearly closed, a partial system must be solved. To avoid the problem of having very high values of B and L the valve has to be considered closed when the effective orifice area is too small. The condition used for this cut off was chosen to be $\frac{A}{A_{eo}} > 10^5$.

Another consideration must also be made; because the algorithm is not able to solve the system all the way to closure there is always some residual flow through the valve, positive or negative, when the valve is closed. The alorithm therefore needs make sure that flow is set to zero at closure.

Four different solver cases If both valves are healthy and fully functioning and they open and close at the right time the full system of three equations never needs to be solved. During systole only the aortic valve is open and during diastole only the mitral valve is open. These periods where one valve is open are separated by two short periods where both valves are closed. For the sake of generality though, the case of both valves being open is also included in the model. This allows another feature to be included in the model; the case where one or both valves are regurgitant (leaking). The general algorithm thus involves four different cases:

Both valves are open To incorporate the possibility of a leaking (regurgitant) mitral or aortic valve the whole equation system must be solved simultaneously. The iteration procedure becomes:

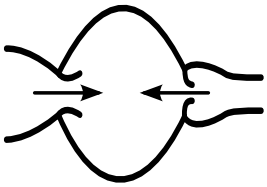


Figure 9
Regurgitation gives simultaneous flow in both valves.

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^i - \frac{1}{det(\boldsymbol{J})} \begin{bmatrix} a_3 - a_4 & -a_4 & a_3 \\ -a_2 a_4 & a_1 a_4 & -a_3 \\ -a_2 & a_1 & -a_2 - a_1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{7.14}$$

where: $det(\boldsymbol{J}) = -a_2 a_4 - a_1 a_4 + a_1 a_3$ is the determinant of the Jacobi matrix. The volume is then updated:

$$V^{n+1} = V^n + (Q_m^n - Q^n + \frac{1}{2}(\Delta Q_m - R_{21}\omega_1 - R_{22}\omega_2))\Delta t \tag{7.15}$$

Aortic valve open If the mitral valve is healthy it is fully closed during systole and only equations 7.9b and 7.9c needs to be solved. Also a partial Jacobi matrix must be used. This gives the iterative procedure:

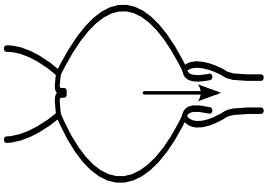


Figure 10
Only flow through the aortic valve during systole.

$$\boldsymbol{x}_{[2,3]}^{i+1} = \boldsymbol{x}_{[2,3]} - \frac{1}{det(\boldsymbol{J}_{[2,3]})} \begin{bmatrix} a_4 & -a_3 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} f_2 \\ f_3 \end{bmatrix} \tag{7.16}$$

where $\boldsymbol{x}_{[2,3]} = [\Delta p_v, \omega_1]$ and $det(\boldsymbol{J}_{[2,3]} = a_3 - a_4$. The flow through the mitral valve is set to zero:

$$Q_m^{n+1} = 0 \tag{7.17}$$

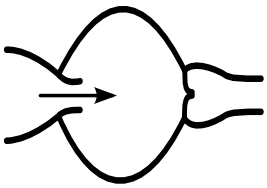


Figure 11
Only flow through the mitral valve during diastole.

Mitral valve open If the aortic valve is healthy it is fully closed during diastole and only equations 7.9a and 7.9b need to be solved giving the iterative procedure:

$$\boldsymbol{x}_{[1,2]}^{i+1} = \boldsymbol{x}_{[1,2]} - \frac{1}{det(\boldsymbol{J}_{[1,2]}} \begin{bmatrix} -1 & -1 \\ -a_2 & a_1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{7.18}$$

where $\boldsymbol{x}_{[1,2]} = [\Delta Q_m, \Delta p_v]$ and $det(\boldsymbol{J}_{[1,2]}) = -a_1 - a_2$. Generally waves are reflected at the boundary when the aortic valve is closed. However if flow is non-zero it has to be corrected. To get a smooth transition to zero flow the following expression was used for this correction:

$$\omega_1 = \frac{-0.5Q^n - R_{22}\omega_2}{R_{21}} \text{ if } Q \neq 0 \tag{7.19}$$

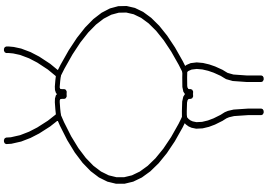


Figure 12
For short periods of time both valves are closed giving no ventricular flow.

Both valves are closed No equation system has to be solved. Mitral flow is zero, incoming waves at the aortic root are reflected, only the ventricular pressure has to be updated according to the change in elastance.

$$Q_m^{n+1} = 0 \tag{7.20a}$$

$$\omega_1 = \omega_2 \tag{7.20b}$$

$$V^{n+1} = V^n \tag{7.20c}$$

$$p_v^{n+1} = E^{n+1}(V^{n+1} - V_0) \tag{7.20d}$$

7.2 Solving the bifurcation equations

As shown in section 5.6 there are nine variables in the governing equations. For each vessel pressure (p), flow (q) and area (A) is unknown. To complete the system six more equations are needed.

The three vessel areas are related to the pressure through the vessel compliance defined as $C = \frac{dA}{dp}$. Pressure and flow increments are related incrementally to the characteristic variables using the $\boldsymbol{L}$ matrix or its inverse the $\boldsymbol{R}$ matrix. Pressure and flow at the three vessel boundaries connected to the bifurcation are thus described by six characteristic variables. The three variables $\omega_{1,1}$, $\omega_{2,2}$ and $\omega_{2,3}$ are entering the bifurcation and can be determined by extrapolation from the internal field. The three variables leaving the bifurcation $\omega_{2,1}$, $\omega_{1,2}$ and $\omega_{1,3}$ are unknown.

In this section a new and improved algorithm for solving the bifurcation equations is presented. But first the existing algorithm is presented.

7.2.1 The existing algorithm

In the bifurcation algorithm currently used by *vascular1Dflow*, updated pressure and flow values are found by solving a set of six equations for the six variables: $p_1^{n+1}, q_1^{n+1}, p_2^{n+1}, q_2^{n+1}, p_3^{n+1}$ and

q_3^{n+1}:

$$f_1(\boldsymbol{x}) = q_1 - q_2 - q_3 = 0 \tag{7.21a}$$

$$f_2(\boldsymbol{x}) = p_1 + \frac{\rho}{2}\frac{q_1^2}{A_1^2} - p_2 - \frac{\rho}{2}\frac{q_2^2}{A_2^2} \tag{7.21b}$$

$$f_3(\boldsymbol{x}) = p_1 + \frac{\rho}{2}\frac{q_1^2}{A_1^2} - p_3 - \frac{\rho}{2}\frac{q_3^2}{A_3^2} \tag{7.21c}$$

$$f_4(\boldsymbol{x}) = \omega_{1,1} - \boldsymbol{l}_{1,1}^T \Delta \boldsymbol{u}_{,1} = 0 \tag{7.21d}$$

$$f_5(\boldsymbol{x}) = \omega_{2,2} - \boldsymbol{l}_{2,2}^T \Delta \boldsymbol{u}_{,2} = 0 \tag{7.21e}$$

$$f_6(\boldsymbol{x}) = \omega_{2,3} - \boldsymbol{l}_{2,3}^T \Delta \boldsymbol{u}_{,3} = 0 \tag{7.21f}$$

Where all the variables are at timestep n+1. The three bottom equations provide the relation between pressure and flow by ensuring that the relation:

$$\boldsymbol{\omega} = \boldsymbol{L}\Delta\boldsymbol{u} \tag{7.22}$$

is fulfilled for the three *known* characteristic variables.

Non-linearity and iterative solution. Only equations 7.21b and 7.21c are non-linear with respect to the variables the system is solved for. The system solved iteratively using the built in function *fsolve* in *numpy*. This function is a wrapper around the *hybrj* and *hybrd* algorithms found in the MINPACK library which is implemented in FORTRAN. Both of which are based on the Newton-Raphson method of root-finding. When supplied with a function giving the Jacobian-matrix of the system the *hybrj* algorithm is used. The inverse Jacobian-matrix is then computed numerically by the system. The iterations are terminated when the relative error is less than the tolerance set by the *xtol* parameter. The relative error is given by the formula:

$$\epsilon = \frac{||\boldsymbol{x}^i - \boldsymbol{x}^{i-1}||}{||\boldsymbol{x}^{i-1}||} \tag{7.23}$$

where euclidean norms are used. In the existing implementation this tolerance was not given to the function meaning that the default value of xtol = 1.49012e-8 was used.

Reuse values from previous time step. There was also a feature in the implementation that checked the size of the residual prior to calling the fsolve function, and if the size of the residuals (i.e. the vector norm) i small enough, values from the previous time-step are instead of finding a solution by iteration. The condition used was that $||\boldsymbol{f}|| < 1e-4$.

Updated values during the iterations. The (L) matrix used is updated according to pressure and flow values from the previous iteration. There is no point in doing this. Because of the mean value theorem we know that the value of $\boldsymbol{L}$ giving the correct increment is $\boldsymbol{L}(\boldsymbol{u_\epsilon})$, where $\boldsymbol{u}^n < \boldsymbol{u}_\epsilon < \boldsymbol{u}^{n+1}$ but generally unknown. There is therefore no reason to think that the updated values of $\boldsymbol{L}$ are a better estimate than $\boldsymbol{L}^n$. A better approach would therefore be to use the average of the updated $\boldsymbol{L}$ and the value from the previous increment $\boldsymbol{L} = \frac{1}{2}(\boldsymbol{L}^n + \boldsymbol{L}^i)$, where $\boldsymbol{L}^i$ is the value of $\boldsymbol{L}$ from the previous iteration. But this seems unnecessary because of the small size of the time-steps.

The value of the area should also be updated for each iteration. In one version of the implemented algorithm the value from the previous timestep is used as an approximation so that $A^{n+1} = A^n$. In another version an updated value from the previous iteration is used giving a more accurate approximation but increasing computational cost.

7.2.2 The new algorithm

Since only two of the equations in the system 7.21a - 7.21f are non-linear it should be unnecessary to solve all of them using iterations. The new algorithm presented in this section takes advantage of this by solving the linear part of the system analytically first, thus reducing the system to two non-linear equations and two unknowns.

Instead of solving for absolute values values of the variables the equations are expressed in terms of increments and solved for the incremental characteristic variables $\omega_{1,2}$ and $\omega_{1,3}$. The choice of which two of the three unknown characteristics to solve for is however arbitrary. The first three equations are discretized by introducing:

$$p = p^n + \Delta p \tag{7.24a}$$
$$q = q^n + \Delta q \tag{7.24b}$$
$$A = A^n + C^n \Delta p \tag{7.24c}$$

Where C^n is the vessel compliance at the bifurcation at timestep n. The value of A^{n+1} is thus estimated by linear extrapolation in contrast to the existing algorithm where the value from the previous increment (or iteration) is used.

Coupling of pressure and flow. Instead of coupling pressure and flow by demanding that they satisfy equations (7.21d) - (7.21f), the inverse relation is used thus giving the following expressions for the increments:

$$\Delta p = \omega_1 + \omega_2 \tag{7.25a}$$
$$\Delta q = R_{21}\omega_1 + R_{22}\omega_2 \tag{7.25b}$$

where R_{11} and R_{12} are omitted since they are equal to one. These expressions are then substituted into the discretized equations. What remains is a system of three equations with the three characteristic variables $\omega_{2,1}$, $\omega_{1,2}$ and $\omega_{1,3}$ as the unknowns:

$$R_{21,1}\omega_{1,1} + R_{22,1}\omega_{2,1} - R_{21,2}\omega_{1,2} - R_{22,2}\omega_{2,2} - R_{21,3}\omega_{1,3} - R_{22,3}\omega_{2,3} = 0 \tag{7.26a}$$

$$p_1^n + \omega_{1,1} + \omega_{2,1} - p_2^n - \omega_{1,2} - \omega_{2,2} + \frac{\rho}{2}\left[\frac{q_1^n + R_{21,1}\omega_{1,1} + R_{22,1}\omega_{2,1}}{A_1^n + C_1(\omega_{1,1} + \omega_{2,1})} - \frac{q_2^n + R_{21,2}\omega_{1,2} + R_{22,2}\omega_{2,2}}{A_2^n + C_2(\omega_{1,2} + \omega_{2,2})}\right] = 0 \tag{7.26b}$$

$$p_1^n + \omega_{1,1} + \omega_{2,1} - p_3^n - \omega_{1,3} - \omega_{2,3} + \frac{\rho}{2}\left[\frac{q_1^n + R_{21,1}\omega_{1,1} + R_{22,1}\omega_{2,1}}{A_1^n + C_1(\omega_{1,1} + \omega_{2,1})} - \frac{q_3^n + R_{21,3}\omega_{1,3} + R_{22,3}\omega_{2,3}}{A_3^n + C_3(\omega_{1,3} + \omega_{2,3})}\right] = 0 \tag{7.26c}$$

The system can then be further reduced by solving equation 7.26a for one of the three unknowns, for example $\omega_{2,1}$, and introducing it into the two other equations. The bifurcation is now described by a system of two nonlinear equations:

$$f_1(\boldsymbol{x}) = p_1^n + \omega_{1,1} + g(\boldsymbol{x}) - p_2^n - \omega_{1,2} - \omega_{2,2} + \frac{\rho}{2}\left[\frac{q_1^n + R_{21,1}\omega_{1,1} + R_{22,1}g(\boldsymbol{x})}{A_1^n + C_1(\omega_{1,1} + g(\boldsymbol{x}))} - \frac{q_2^n + R_{21,2}\omega_{1,2} + R_{22,2}\omega_{2,2}}{A_2^n + C_2(\omega_{1,2} + \omega_{2,2})}\right] = 0 \tag{7.27a}$$

and

$$f_2(\boldsymbol{x}) = p_1^n + \omega_{1,1} + g(\boldsymbol{x}) - p_3^n - \omega_{1,3} - \omega_{2,3} + \frac{\rho}{2}\left[\frac{q_1^n + R_{21,1}\omega_{1,1} + R_{22,1}g(\boldsymbol{x})}{A_1^n + C_1(\omega_{1,1} + g(\boldsymbol{x}))} - \frac{q_3^n + R_{22,3}\omega_{1,3} + R_{22,3}\omega_{2,3}}{A_3^n + C_3(\omega_{1,3} + \omega_{2,3})}\right] = 0 \tag{7.27b}$$

where:

$$g(\boldsymbol{x}) = \omega_{2,1} = \frac{1}{R_{22,1}}(R_{21,2}\omega_{1,2} + R_{22,2}\omega_{2,2} + R_{21,3}\omega_{1,3} + R_{22,3}\omega_{2,3} - R_{21,1}\omega_{1,1}) \tag{7.28}$$

The Jacobian matrix is therefore also simplified from a six-by-six matrix to a two-by-two matrix given by:

$$\boldsymbol{J} = \frac{d\boldsymbol{F}}{d\boldsymbol{x}} = \begin{bmatrix} \frac{R_{21,2}}{R_{22,1}} - 1 + \rho\frac{h_1}{h_2^3}(R_{21,2}h_2 - C_1\frac{R_{21,2}}{R_{22,1}}h_1) & \frac{R_{21,3}}{R_{22,1}} + \rho\frac{h_1}{h_2^3}(R_{21,3}h_2 - C_1\frac{R_{21,3}}{R_{22,1}}h_1) \\ \quad - \rho\frac{h_3}{h_4^3}(R_{21,2}h_4 - C_2h_3) & \\ \frac{R_{21,2}}{R_{22,1}} + \rho\frac{h_1}{h_2^3}(R_{21,2}h_2 - C_1\frac{R_{21,2}}{R_2 2,1}h_1) & \frac{R_{21,3}}{R_{22,1}} - 1 + \rho\frac{h_1}{h_2^3}(R_{21,3}h_2 - C_1\frac{R_{21,3}}{R_2 2,1}h_1) \\ & \quad - \rho\frac{h_5}{h_6^3}(R_{21,3}h_5 - C_3h_6) \end{bmatrix} \tag{7.29}$$

where:

$$h_1 = q_1^n + R_{21,1}\omega_{1,1} + R_{22,1}g(\boldsymbol{x}) \tag{7.30a}$$
$$h_2 = A_1^n + C_1(\omega_{1,1} + g(\boldsymbol{x})) \tag{7.30b}$$
$$h_3 = q_2^n + R_{21,2}\omega_{1,2} + R_{22,2}\omega_{2,2} \tag{7.30c}$$
$$h_4 = A_2^n + C_2(\omega_{1,2} + \omega_{2,2}) \tag{7.30d}$$
$$h_5 = q_3^n + R_{21,3}\omega_{1,3} + R_{22,3}\omega_{2,3} \tag{7.30e}$$
$$h_6 = A_3^n + C_3(\omega_{1,3} + \omega_{2,3}) \tag{7.30f}$$

Another benefit of this system is that it is completely dimensionally homogeneous. All the variables and all the functions have pressure as the dimension and hence also the Jacobian matrix.

Non-linearity and number of iterations required Having determined the Jacobian matrix the system can now be solved using Newton's method. Using values of $\omega_{1,2}$ and $\omega_{1,3}$ from the previous timestep as the starting point $\boldsymbol{x_0}$ the new values are computed using the Newton-Raphson method:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \boldsymbol{J}^{-1}(\boldsymbol{x}^k)\boldsymbol{f}(\boldsymbol{x}^k) \tag{7.31}$$

which was implemented in python rather than using *fsolve*. The error tolerance was set to $\epsilon_{\text{tol}} = 0.001$

Also the solution from the previous timestep was kept if the residuals were small enough. However because the variables being solved for here are the increments $\boldsymbol{x} = [\omega_{2,2}, \omega_{2,3}]$ as opposed to absolute values of pressure and flow, keeping the previous solution still changes the absolute values of pressure and flow. The entire code for the implementation is shown in the appendix.

The degree of non-linearity of the equations depends on the flow rate so that at small flow rates they become nearly linear and they become more non-linear with increasing flow.

Inspection shows however that they are actually quite linear in the region of interest in all realistic cases cases. The solution obtained by Newton's method should therefore converge quickly.

7.3 Improved discretization scheme for Windkessel boundary conditions

The discretization schemes for the two- and three element Windkessel boundary conditions were improved. The existing scheme for WK2 is presented as well as two alternative discretization schemes. A method for obtaining the analytical solution for the reflection of a single pulse is also presented. This method is later used to verify the boundary conditions by comparing analytical and simulated result. The governing equation for the two-element windkessel model was given in section 5.4 but is repeated here:

$$\frac{dp}{dt} + \frac{p - p0}{RC} = \frac{q}{C} \tag{7.32}$$

where R and C respectively are the resistance and compliance of the windkessel model and p_0 is a peripheral pressure assumed to be constant.

7.3.1 Existing solution

In the existing version of the *vascular1Dflow* software this equation was discretized by replacing p and q in the equation with increments of the same values giving:

$$\frac{d(\Delta p)}{dt} + \frac{1}{RC}\Delta p = \frac{\Delta q}{C} \tag{7.33}$$

the reference pressure p_0 being constant vanishes. The pressure and flow increments are then written in terms of the characteristic variables. In particular the derivative is written as:

$$\frac{d(\Delta p)}{dt} = \frac{\omega_1^{n+1} + \omega_2^{n+1} - \omega_1^n - \omega_2^n}{\Delta t} \tag{7.34}$$

In the case of the boundary condition being located at the distal end of a vessel the unknown variable ω_2 is given by:

$$\omega_2 = \frac{-(1 + \frac{RC}{\Delta t} - RR_{21})\omega_1 + \frac{RC}{\Delta t}(\omega_1^n + \omega_2^n)}{1 + \frac{RC}{\Delta t} - RR_{22}} \tag{7.35}$$

7.3.2 Alternative solution

This discretization scheme works well but it seems that the purely incremental form would be less accurate than a full discretization that included pressure and flow values from the previous time steps as well as the increments.

Such a solution would also be able to retain the peripheral pressure p_0 in the equation. If this pressure is not included, the windkessel model acts purely as a reflector and is not able equalize the pressure difference if the initial pressure of the network is different from the peripheral pressure. This means that if no initial flow is prescribed in the network along with the initial pressure, the pressure always returns to the initial pressure.

As shown in section 8.4.2 with the alternative discretization it is possible to initialize a network at a high pressure and then allow the pressure to decay exponentially down to the peripheral pressure. This is also utilized when testing the varying elastance model. Instead of having to start the simulation at zero pressure and then letting the simulation run for several heart cycles until the network has been inflated (i.e. there is a balance between inflow and outflow), the pressure could be initiated somewhere in the region of 80-120 mmHg, and pressure would still decay towards the peripheral pressure instead of the initial pressure. Two alternatives to the existing discretization were implemented and tested.

Alternative 1 - 2nd order backward difference: The equation is discretized at timestep n+1 by setting:

$$p = p^n + \Delta p \tag{7.36a}$$

$$q = q^n + \Delta q \tag{7.36b}$$

The derivative is estimated using a second order backward differential given by:

$$(\frac{dp}{dt})^{n+1} = \frac{3p^{n+1} - 4p^n + p^{n-1}}{2\Delta t} = \frac{\Delta p - p^n + p^{n-1}}{2\Delta t} \tag{7.37}$$

When applied at the distal end of a vessel, the expression for the unknown backward travelling characteristic variable thus becomes:

$$\omega_2 = -\frac{(1 + \frac{3RC}{2\Delta t} - RR_{21})\omega_1 + p^n - \frac{RC}{2\Delta t(p^n - p^{n-1}) - p_0 - Rq^n}}{1 + \frac{3RC}{2\Delta t} - RR_{22}} \tag{7.38}$$

Alternative 2 - Half-step central difference: Another approach is to write the equation at timestep $n + \frac{1}{2}$:

$$\frac{\Delta p}{\Delta t} + \frac{1}{RC}(p^n + \frac{1}{2}\Delta p - p_0) = \frac{1}{C}(q^n + \frac{1}{2}\Delta p) \tag{7.39}$$

The derivative of the original equation is thus represented by a central difference between timestep n and $n + 1$. The resulting expression for ω_2 is similar to the previous expression:

$$\omega_2 = \frac{-(1 + 2\frac{RC}{\Delta t} - RR_{21})\omega_1 + 2(Rq^n - p^n + p_0)}{1 + 2\frac{RC}{\Delta t} - RR_{22}} \tag{7.40}$$

This is a similar but simpler expression in that it does not require the use of p^{n-1} which simplifies the implementation somewhat

7.3.3 Test case and analytical solution

Test case In order to verify the simulated results and make a comparison of the discretization schemes a simple test case was formulated. A single forward propagating pressure pulse is imposed on one end of a vessel using a WK2 as the boundary condition at the other end. The pulse travels through the vessel and into the boundary condition where it is reflected and travels back to the origin. The amplitude of the prescribed pulse is low to prevent non-linear effects. In the linear case it is possible to obtain an analytical solution of the reflected wave. The simulated reflected wave can then be compared with the analytical solution.

Analytical solution As a first step in obtaining an analytical solution pressure and flow are expressed in terms forward and backward propagating pressures so that:

$$p = p_f + p_b \tag{7.41a}$$

$$q = q_f + q_b \tag{7.41b}$$

where subscripts f and b denote forward and backward travelling waves respectively and the relation $\frac{p_f}{q_f} = -\frac{p_b}{q_b} = Z_c$. The equation can now be seen as a non-homogeneous ODE of the variable p_b given by:

$$\frac{dp_b}{dt} + (\frac{1}{RC} - \frac{1}{Z_c C})p_b = -(\frac{dp_f}{dt} + (\frac{1}{RC} + \frac{1}{Z_c C})p_f) \tag{7.42}$$

where the right side of the equation is the driving function determined by the forward propagating pressure. An easy way of solving this is using the reflection coeffient, which can be obtained by taking the fourier transform of the equation and rearranging it so that:

$$\frac{P_b}{P_f} = \frac{Z_c - \frac{R}{1+i\omega RC}}{Z_c + \frac{R}{1+i\omega RC}} \tag{7.43}$$

which could also have been obtained directly from the input impedance. This complex number then translates to a absolute value:

$$|\Gamma| = \frac{\sqrt{((\frac{1}{Z_c} - \frac{1}{R})^2 - \omega^2 C^2)^2 + 4(\frac{1}{Z_c} - \frac{1}{R})^2 \omega^2 C^2}}{(\frac{1}{Z_c} + \frac{1}{R})^2 + \omega^2} \tag{7.44}$$

which gives the amplitude ratio between the bakward and forward propagating wave and a phase shift:

$$\phi = \arctan(\frac{y}{x}) \begin{cases} +0 & \text{if } x > 0 \\ +\pi & \text{if } x < 0 \text{and } y > 0 \\ -\pi & \text{if } x < 0 \text{and } y < 0 \end{cases} \tag{7.45}$$

where:

$$x = (\frac{1}{Z_c} - \frac{1}{R})^2 - \omega^2 C^2 \tag{7.46a}$$

$$y = -2(\frac{1}{Z_c} - \frac{1}{R}) \tag{7.46b}$$

the added and subtracted π makes sure that the resulting angle is in the correct quadrant, the expression corresponds to the *atan2* function found in many programming languages and which was used during the implementation of this verification procedure. Thus if the forward propagating pressure is the harmonic function:

$$p_f = a_f \cos(\omega t) \tag{7.47}$$

the resulting backward pressure becomes:

$$p_b = |\Gamma(\omega)| a_f \cos(\omega t + \phi(\omega)) \tag{7.48}$$

This could be used directly to verify the boundary condition by prescribing a continuous forward propagating harmonic signal for some length of time and then use wavesplitting to compare the forward and backward propagating signals and see if the amplitude relation and phase shift are correct.

Single pulse - fourier series solution. It is more difficult to obtain an exact solution for this equation in the case where forward propagating pressure wave is a single pulse. However if the forward propagating pulse is assumed to be a periodic train of pulses, it can be expressed as a fourier series and the solution from 7.48. Using a half-range expansion of an even function the forward pressure wave can be represented by the fourier cosine-series:

$$p_f(t) = a_0 + \sum_{n=1}^{\infty} a_{f,n} \cos \frac{n\pi}{L} t \tag{7.49}$$

where $a_{f,n}$ are the fourier coefficient and L is the half the period of the pulse train. The resulting backward pressure becomes:

$$p_b(t) = -\frac{\frac{1}{RC} - \frac{1}{Z_c C}}{\frac{1}{RC} + \frac{1}{Z_c C}} a_0 \sum_{n=1}^{\infty} |\Gamma(\omega_n) a_{f,n} \cos(\frac{n\pi}{L} t + \phi(\omega_n)) \tag{7.50}$$

where $\omega_n = \frac{n\pi}{L}$. The resulting backward propagating pressure thus becomes an infinite train of pulses of same period as the forward propagating pressure. If the pulses are spaced far enough apart they to not have any significant influence on each other and make a very good approximation of the analytical solution for a single pulse.

Prescribed pulse The pulse used for the simulations was a single pressure wave consisting of a half cosine wave:

$$p_f(t) = \begin{cases} \frac{a}{2}(1 + \cos\frac{\pi}{2t_0}t) & \text{for } 0 < t < 2t_0 \\ 0 & \text{for } t > 2t_0 \end{cases} \tag{7.51}$$

This type of waveform has the benefit that it is continuous and smooth causing no numerical difficulties and that its fourier coeficcients are easily obtained. The amplitude is chosen to be small so that the system is approximately linear. The fourier coefficients are given by:

$$a_0 = \frac{2t_0}{L} \tag{7.52a}$$

and

$$a_n = \frac{\sin(\frac{nt_0}{L})}{(\frac{nt_0}{L} - (\frac{nt_0}{L})^3)\frac{\pi L}{t_0}} \tag{7.52b}$$

The input waveform and the analytical solution of the reflection is shown in figure 13

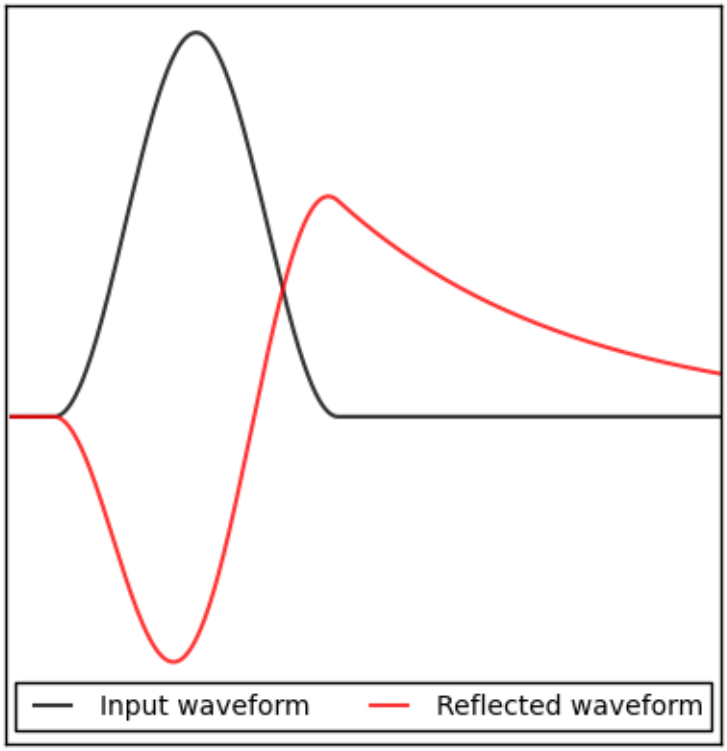


Figure 13: A plot of the input waveform suberimposed on the fourier series solution of the reflected wave

8 Results and discussion

8.1 Lumped model and single vessel testing of the varying elastance model

The varying elastance boundary condition was tested in a single vessel network. The varying elastance boundary condition was applied at one end of the vessel and the other end was terminated by a three element windkessel model (WK3). The model parameters used were taken from Stergiopulos et al. [13], in which a varying elastance model is simulated in a lumped model windkessel. Simulations were done on a very short vessel, thus emulating a lumped model simulation and on a longer vessel. The purpose of the tesing was to:

- See if a realistic pressure and flow waveform could be produced by the varying elastance model in a lumped model.
- Investigate the usefulness of a single vessel simulation.
- See if the valve model worked properly, particularly with regards to closing.
- Demonstrate the two aortic valve features regurgitation and stenosis.
- Investigate the load dependence introduced by the systolic resistance K in the varying elastance model.

Adaptation of parameters for single vessel. The characteristic impedance of the WK3 was matched to the impedance of the vessel and the peripheral resistance of the WK3 was set so that the total resistance was equal to the total resistance given in the article. In the article the characteristic impedance $Rc = 0.51$ $\frac{\text{mmHg s}}{\text{ml}}$ was used and a peripheral resistance $Rp = 1.05$ $\frac{\text{mmHg s}}{\text{ml}}$, thus giving a total resistance of 1.56. The characteristic impedance of the model was thus determined by vessel parameters only.

The parameters for the varying elastance model as well as the boundary conditions are given in table 3.

Total resistance, $\frac{\text{mmHg s}}{\text{ml}}$	1.56
Total compliance, $\frac{\text{ml}}{\text{mmHg}}$	1.60
Heart period, s	1.00
Maximum elastance ($E_{\max}$), $\frac{\text{mmHg}}{\text{ml}}$	2.31
Minimum elastance ($E_{\min}$), $\frac{\text{mmHg}}{\text{ml}}$	0.06
Unloaded volume (V_d), ml	20.0
Venous pressure (P_v), mmHg	7.50
Time to peak elastance, (T_p), s	0.43

Table 3: Model parameters

The parameters for fluid properties and velocity profile were identical in all simulations and were set to: As the diastolic phase of the cardiac cycle is of secondary importance in this study, the mitral flow curves are not shown in the results. The diastolic filling does however have some important as this determines the end diastolic volume (EDV) and thus the preload of the ventricle. Mitral flow and mitral valve dynamics are simulated but the governing parameters are very uncertain giving very uncertain results for the mitral flow curve. The atrial pressure however, which largely determines the EDV, can be used as a parameter for preload because the EDV is always approximately $EDV = \frac{p_{atr}}{E_{min}} + V_0$

Viscosity (μ), Pa s	3.5e-3
Mass density (ρ), $\frac{\text{kg}}{m^3}$	1060.0
Velocity profile (δ)	0.33
Opening area ratio (M_{st})	0.99
Atrial pressure (p_{atr}), mmHg	7.5

Table 4: Fluid parameters

The Young's modulus was kept constant and the following parameters were varied for the different simulations:

Table 5: Overview of changes in simulation parameters for different simulations

The M_{st} parameter determines the effective orifice area of the aortic valve at full opening. For a healthy non-stenotic valve EOA = A, when the valve is completely open [4]. This corresponds to $M_{st} = 1.0$ so that for a fully open valve ($\zeta = 1.0$) B = L = 0 and there is no pressure gradient at all. This is not desireable since the closure of the valve depends on a negative pressure gradient. Thus for the simulations $M_{st} = 0.99$ was used so that there would always be a small pressure gradient. This is still much higher than what was used by Mynard et al. [7], they used $M_{st} = 1$, but by their definition of A_{eff} this represents just $M_{st} = 0.5$ by the definition used in this thesis.

Forward and backward propagating pressure The total pressure is plotted together with the forward- and backward-propagating pressures. These pressures are found using a non-linear wavesplitting function inluded in the *vascular1Dflow* software. The total pressure consists of a reference pressure in addition to the the two propagating parts, so that $p = p_0 + p_f + p_b$. In all these plots the wavesplitting function was applied for just the data that was plotted and not the entire data set. The constant in the expression for the total pressure is thus not identical to the initial pressure of the whole simulation but to the initial pressure of the set of data that the wavesplit function was applied to which is then taken to be a reference pressure. This reference pressure is shown as a horizontal dotted line, the propagating components are shifted, so that they are displayed as fluctuations around this reference pressure.

Short vessel A simulation was made that was intended to replicate the lumped model approach using a short vessel. The characteristic impedance of the vessel was set equal to the characteristic impedance used in the article. This characteristic impedance is much higher than the characteristic impedance of for example the human aortic root. Vessel parameters are given in table 6.

Length (l), cm	1.0
Youngs modulus (E), Pa	400000
Wall thickness (h), cm	0.1
Cross sectional area (A), cm^2	0.9204

Table 6: Vessel parameters

The initial pressure is set to 80 mmHg and the simulation is run for three seconds giving three full cardiac cycles. Thus allowing the system to reach a steady balance between inflow

and outflow. The result of the simulation on the short vessel is shown in figure 14. The first two simulations were made using a source resistance K set to zero and a aortic valve pressure threshold for closing set to $\Delta p_{\text{close}} = 2.0$ mmHg.

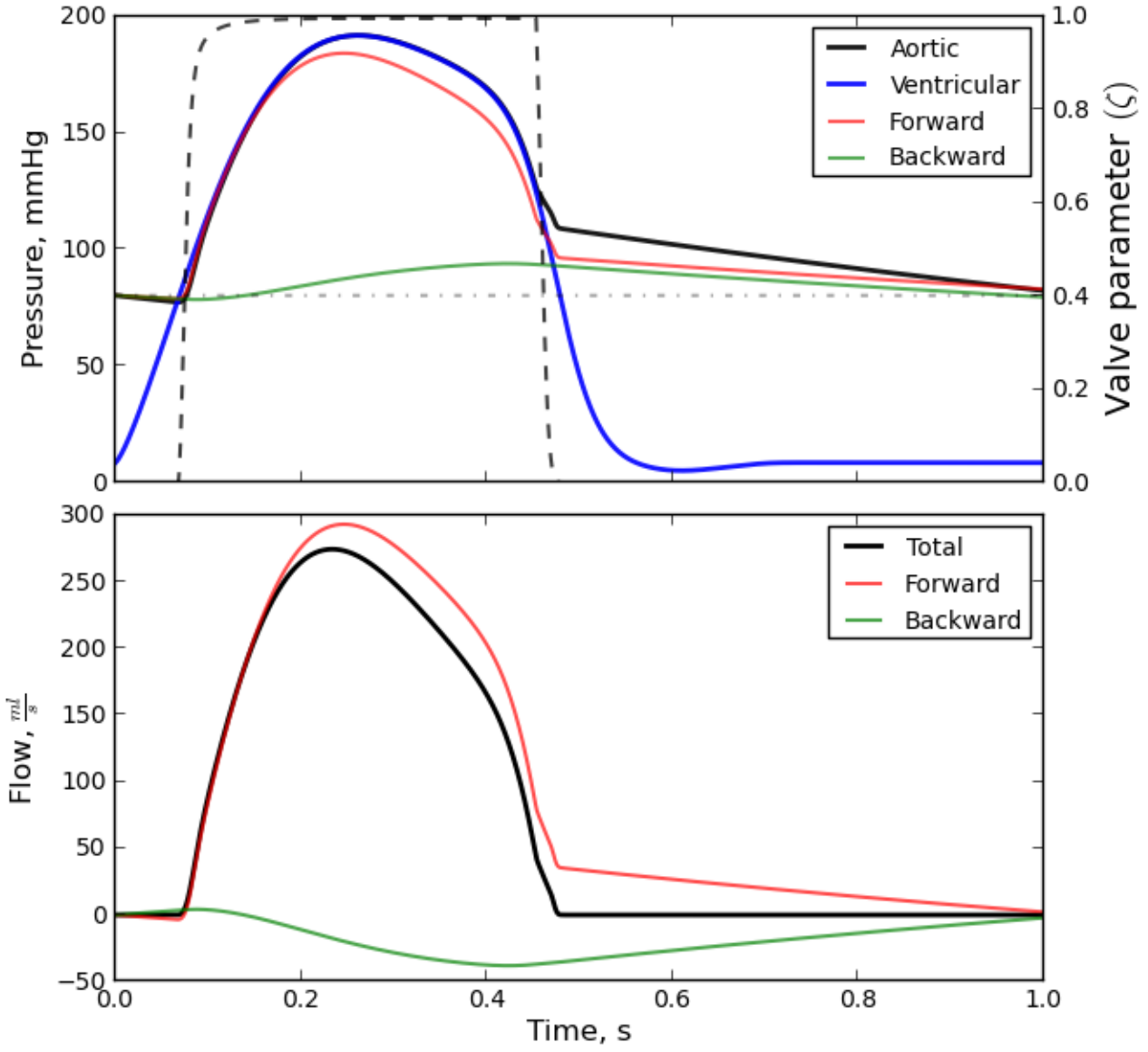


Figure 14: Varying elastance simulation on a short vessel of length 1 cm, with a source resistance of K = 0

The resulting pressure and flow waveforms are not very realistic. The maximum pressure is very high, there is no dicrotic notch and the reflected wave is very small compared to the forward propagating wave. Also because there are almost no reflected waves, the flow waveform is almost identical to the pressure waveform which it shouldn't be. It is believed that the cause of this is that the characteristic impedance of the vessel and windkessel is to high. Another simulation was made where impedance of the vessel was reduced, thus also automatically reducing the characteristic impedance of the windkessel model to about a tenth of the original value by setting $h = 1.63\ mm$ and $A = 6.78\ cm^2$. The impedance of the vessel is now $Z_c = 0.053 \frac{\text{mmHg}s}{\text{ml}}$.

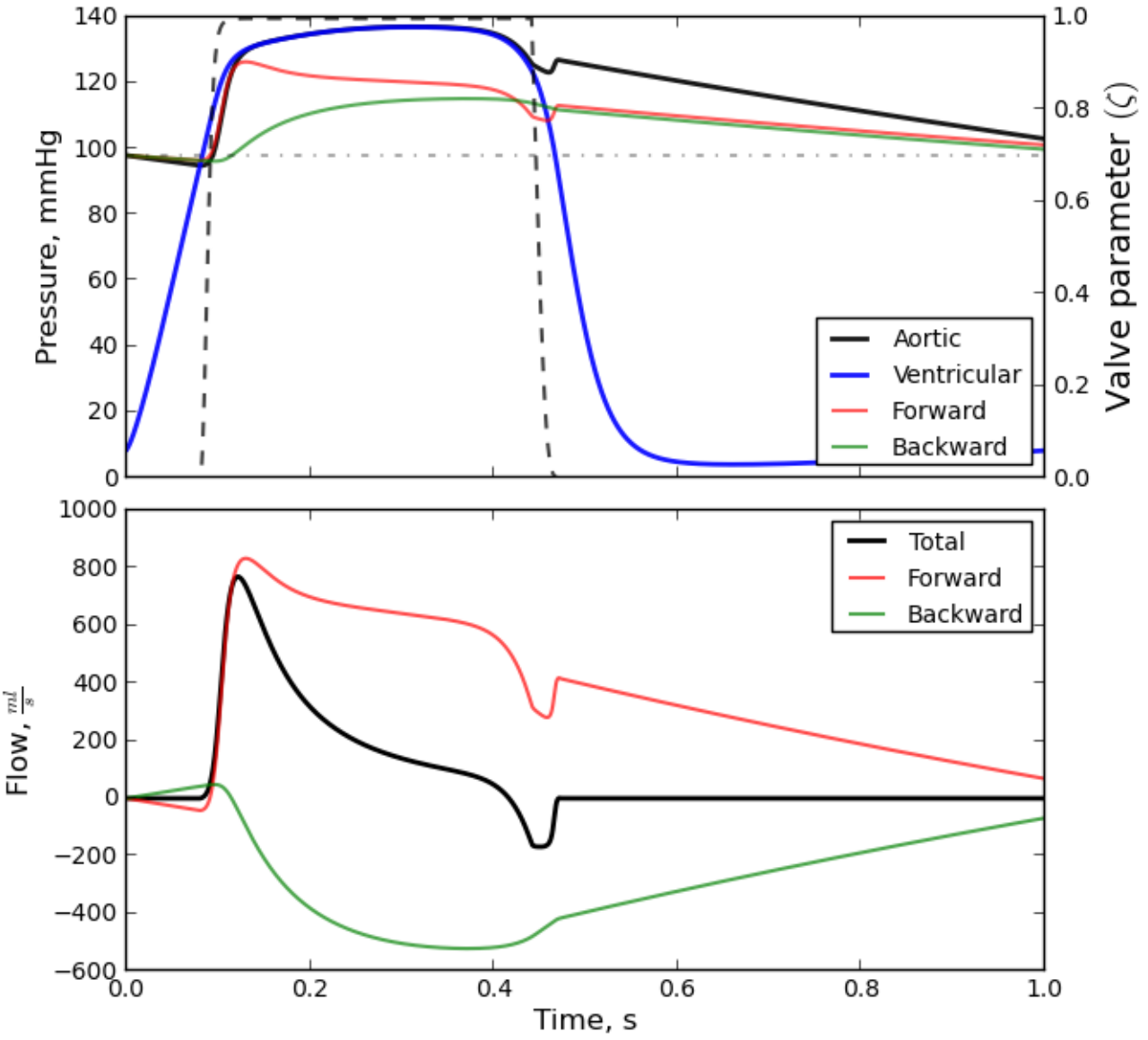


Figure 15: Varying elastance simulation on a short vessel of length 1 cm, with a source resistance of K = 0 and reduced characteristic impedance at $Z_c = 0.053$.

The pressure waveform now looks more realistic. It has a lower maximum pressure and it can also be seen that the flow waveform is different from the pressure waveform. It has a large spike in the beginning and a small backflow at the end of systole causing the dicrotic notch. The reflected pressure wave now also constitutes a larger part of the total pressure. It is clear that the difference lies in the way the input impedance at the aortic root varies during systole. The impedance is the result of two phenomena; one is the characteristic impedance of the vessel the other is the presence of wave reflections. When using a high characteristic impedance this dominates throughout systole. When using a low characteristic impedance and a high peripheral resistance, reflections make up a large part of the impedance. Because of this impedance varies greatly throughout systole thus giving a flow waveform that looks nothing like the pressure waveform.

Systolic resistance To demonstrate the effects of the systolic resistance K, another simulation was run on the short vessel where $K = 1500\frac{s}{\text{ml}}$. To compensate for the systolic resistance the maximum elastance was increased by 20 %. The results are shown in figure 16

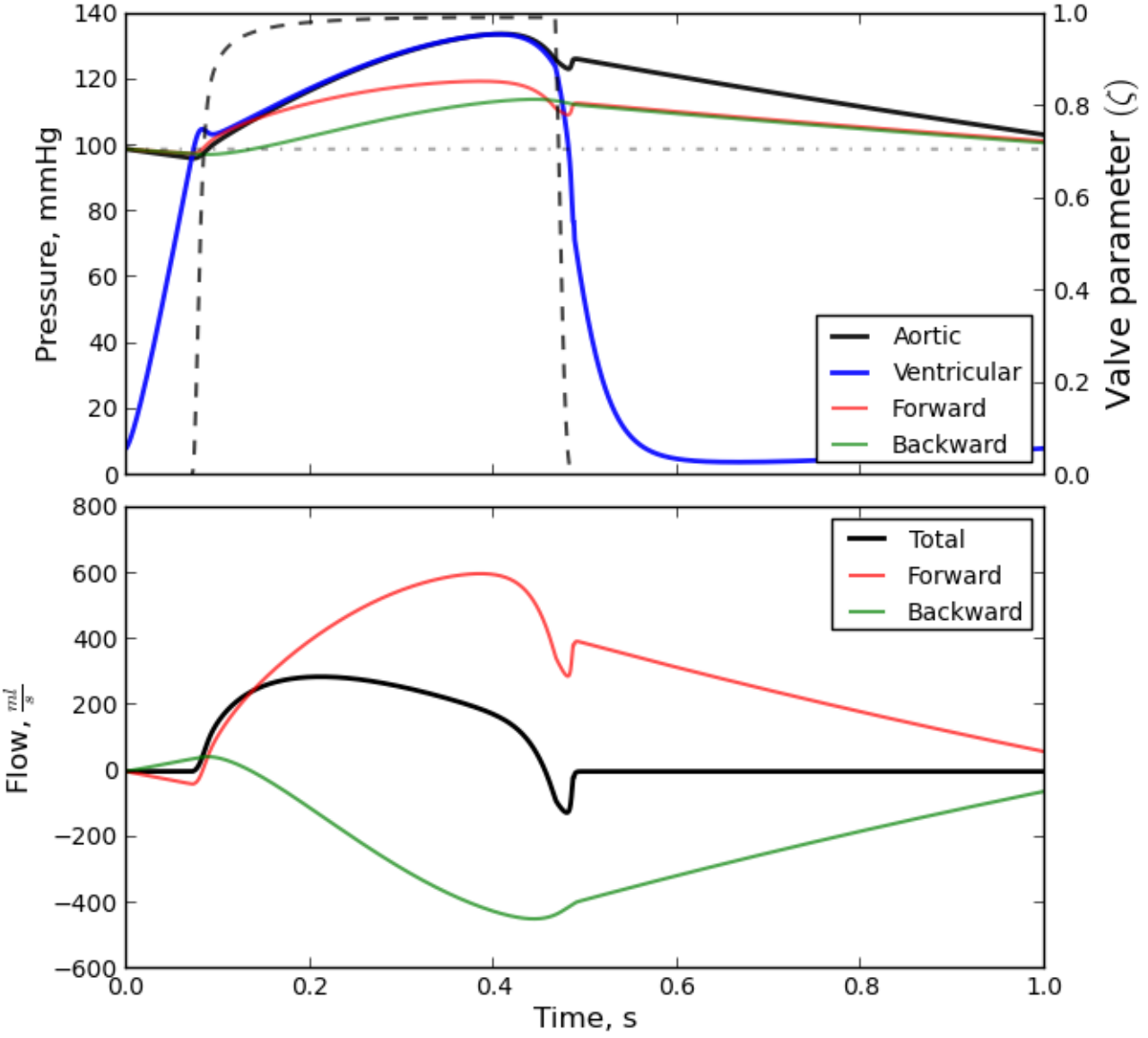
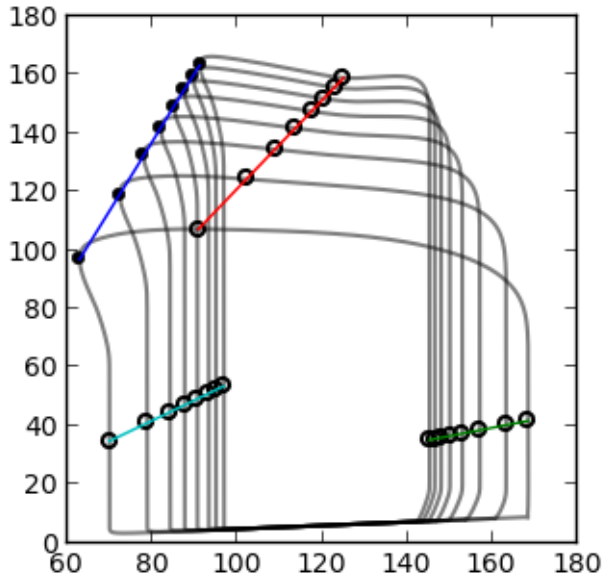


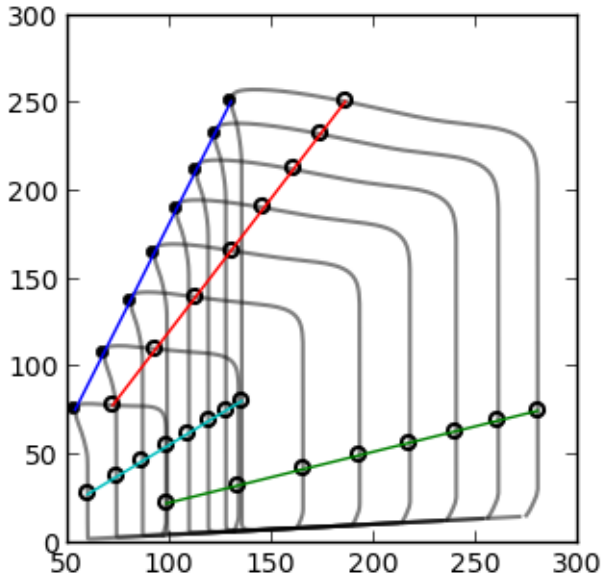
Figure 16: Varying elastance simulation with $K = 1500\frac{s}{\text{ml}}$

The flow waveform is dramatically changed and the peak at the beginning of systole has completely disappeared. This might suggest that the value of K is to high.

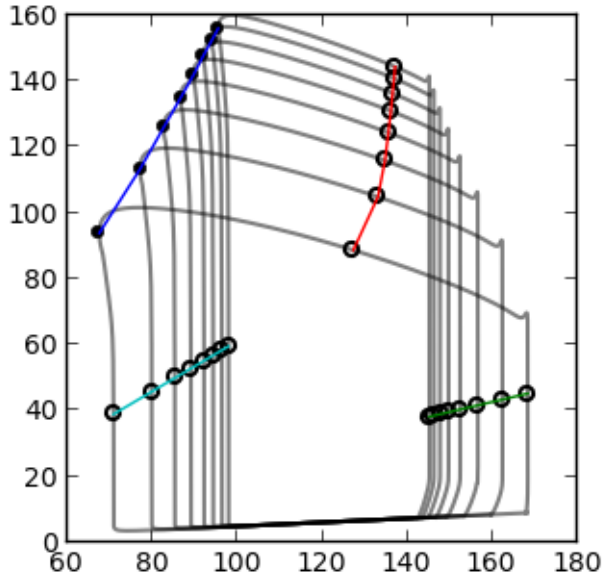
Preload and afterload variation - Load dependence To determine the effects of pre- and afterload on the varying elastance model simulations one were made on the short vessel. A low vessel characteristic impedance was used and simulations were done for five different afterloads, determined by the total resistance of the boundary condition, the results were then compared by plotting pressure versus volume in p-v loops. As with the other simulations data three heart cycles were simulated but only the last one was used in the plot. For each loop three isochrones at times 0.035s, 0.2s and 0.5s were marked by circles. The end systolic points, defined as the minimum of ventricular volume, was marked by solid black dots to give an impression of the ESPVR. This was repeated using a systolic resistance of $K = 1500\frac{s}{m^3}$. To compensate for the systolic resistance E_{max} was increased by 10 % so that $\grave{E}_{max} = 2.54$. The results are shown in figure 17.



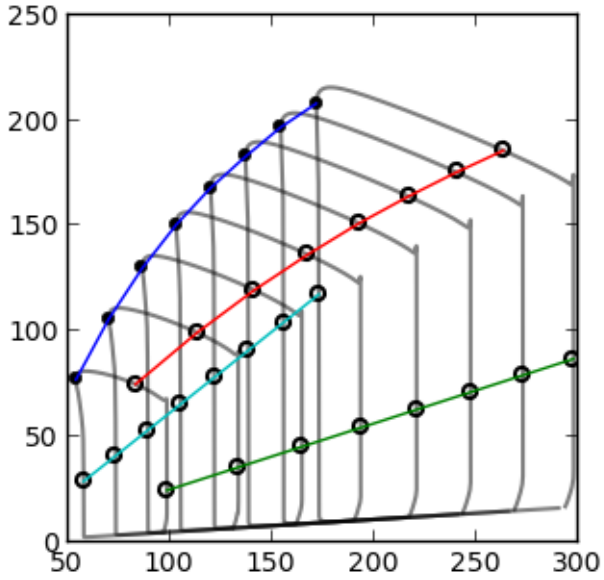
(a) Varying afterload, K = 0, $E_{max} = 2.31$



(b) Varying preload, K = 0, $E_{max} = 2.31$



(c) Varying afterload, K = 1500, $E_{max} = 2.54$



(d) Varying preload, K = 1500, $E_{max} = 2.54$

Figure 17: Pressure-volume loops for varying elastance simulations on a short vessel for different pre- and afterloads. The three isochrones at times 0.035s, 0.2s and 0.5s are marked by circles. Isochrones are chosen so that they take place during the isovolumic contraction, ejection and isovolumic relaxation phases respectively. End systole, defined as the point of minimum ventricular volume is marked by dots.

Something very peculiar about the results when afterload is varied is that the end-diastolic volume is reduced with increasing afterload. Increases in afterload should reduce the stroke-volume but this should of course be an effect of a increased end-systolic volume. The EDV is affected only by preload and should be roughly identical for variations in afterload. This suggests that the parameters governing diastolic filling are not quite right, it seams that the inertia of blood passing through the mitral valve leads to a greater EDV when the flow through the mitral valve is larger (i.e. the stroke volume is larger).

The effects of the source resistance As seen in figure 17d the source resistance produces a convex ESPVR when preload is varied. The ejection phase isochrone is also convex, whereas the two other isochrones are linear which is as expected since there is no flow at these points in the cycle. When the afterload is varied, as seen in figure 17c, the ESPVR is almost completely linear. The isochrone during the ejection phase is also almost linear, there is however a remarkable difference in angle, this is likely not caused by the source resistance, but as mentioned earlier by problems with the diastolic filling.

Aortic valve regurgitation and aortic valve stenosis To demonstrate a regurgitant valve a simulation was run on the short vessel with the M_{rg} parameter set to 0.02.This means that the valve never fully closes and blood leaks back into the ventricle during diastole. The parameters were otherwise the same as in the first simulation (K = 0). The resulting pressure and flow is shown in figure 18.

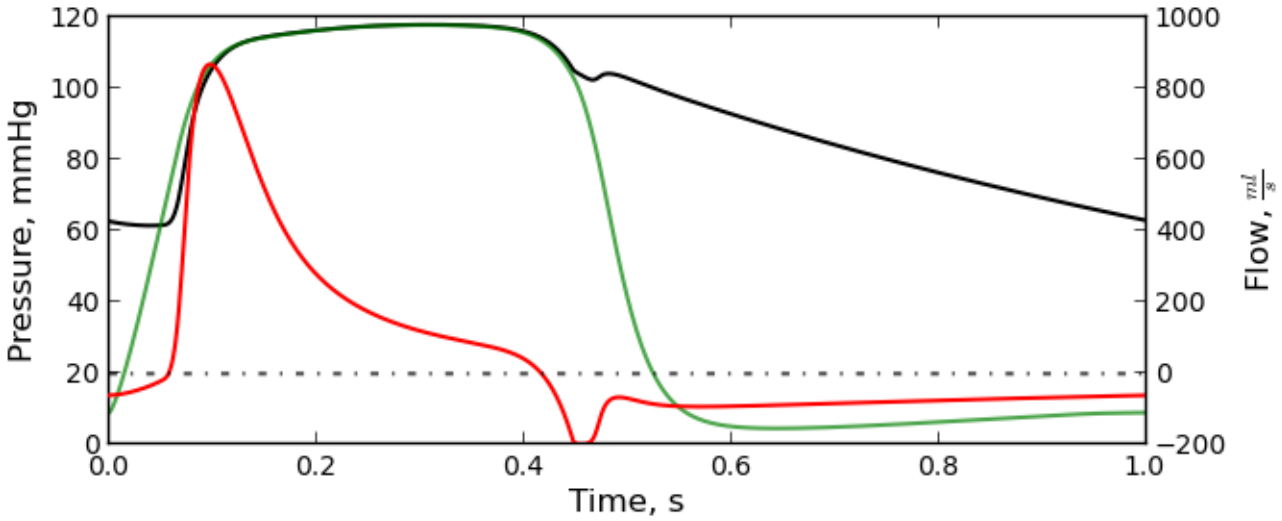


Figure 18: Regurgitation causes a negative flow during diastole.

As expected the flow is negative throughout the diastole. An aortic valve stenosis was then simulated by setting $M_{st} = 0.3$. The other valve parameters were kept the same. The results are shown in figure 19.

The pressure gradient is very large in the beginning of systole, then when the flow deccelerates gradient becomes negative and the aortic valve closes prematurely and causes a bad results. From Mynard et al. [7] the valve parameters should be $K_{vo} = 0.01$ and $K_{vc} = 0.01$ for a mild stenosis. These values were also tested and the results are shown in figure 20. This makes the valve close at at a much slower rate, making it more stable.

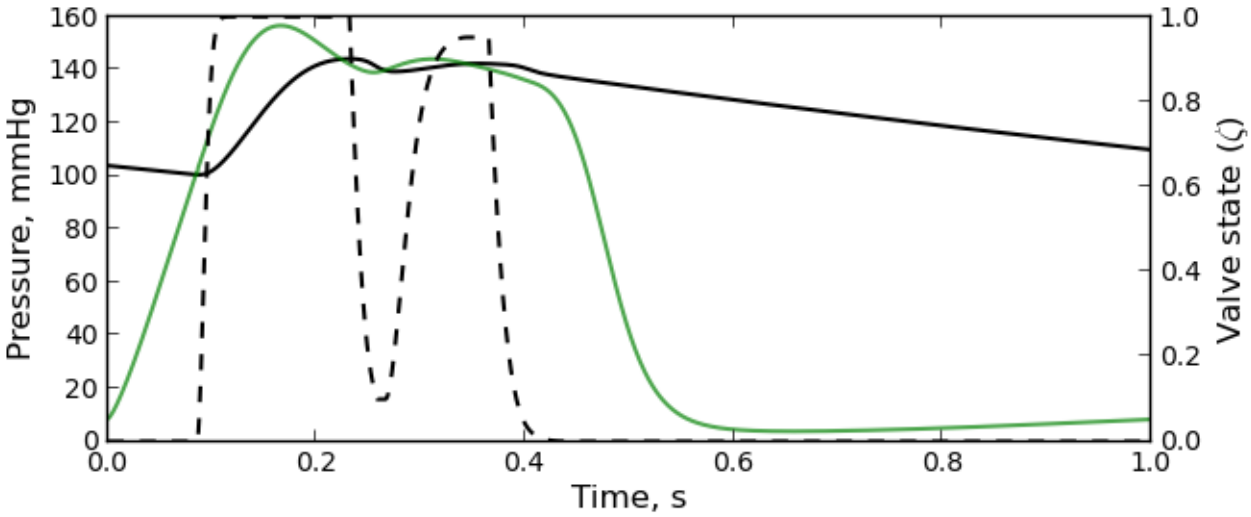


Figure 19: The stenosis causes a large pressure gradient across the aortic valve. The opening and closing rate parameters of the aortic valve causes it to flicker between the open and closed state.

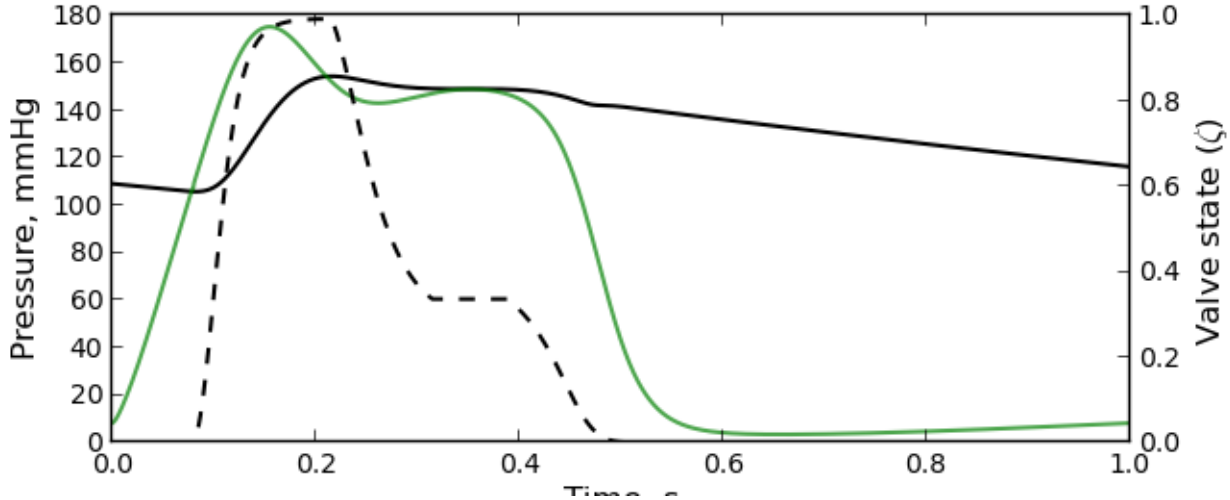


Figure 20: The stenosed valve becomes more stable when the opening and closing rate parameters are reduced.

Long vessel simulation Simulations were made on a longer vessel. The idea was to see the effect of delaying the reflected wave. The vessel parameters were kept the same as for the previous simulations thus giving a vessel characteristic impedance of $Z_c = 0.053$ at zero pressure. Since the vessel now has a significant volume compliance of its own, the compliance of the windkessel model was reduced the total compliance of vessel and windkessel model was equal to the lumped model compliance given in the article.

At the initial pressure of 80 mmHg the wavespeed was $c = 5.2\frac{m}{s}$. The vessel length was set to 30 cm thus delaying the reflection by approximately 0.12 seconds. The resistance of the vessel assuming steady poiseuille flow is:

$$R = \frac{8\pi\mu L}{A^2} = 0.00043 \ \frac{\text{mmHg s}}{\text{ml}} \tag{8.1}$$

which is very small compared to the resistance of the boundary condition. The simulation results are shown in figure 21. The solution doesn't look very realistic. This time it is caused by the

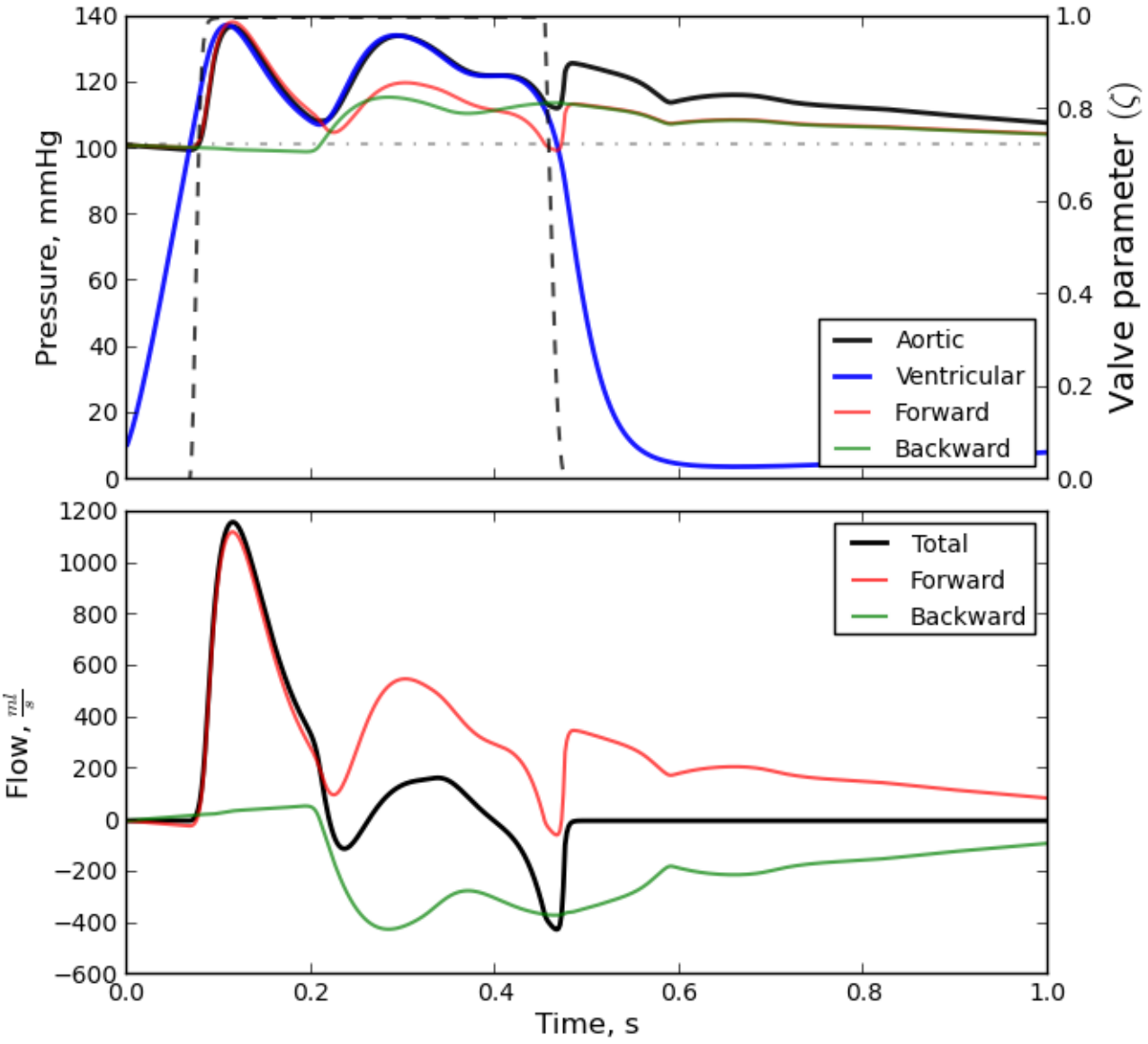


Figure 21: The rapid increase in outflow in the beginning followed by a decrease in flow causes the aortic valve to start closing prematurely.

extreme spike in the flow at the beginning of systole when there are no reflections. There is

almost no afterload and the heart is almost emptied giving a pressure drop in mid-systole. The delay time for the reflection is too large it seems. A somewhat prettier result was obtained by reducing the vessel length to 15 cm, giving half the delay time, as well as introducing a small systolic resistance $K = 250\frac{s}{m^3}$. The results are shown in figure 22.

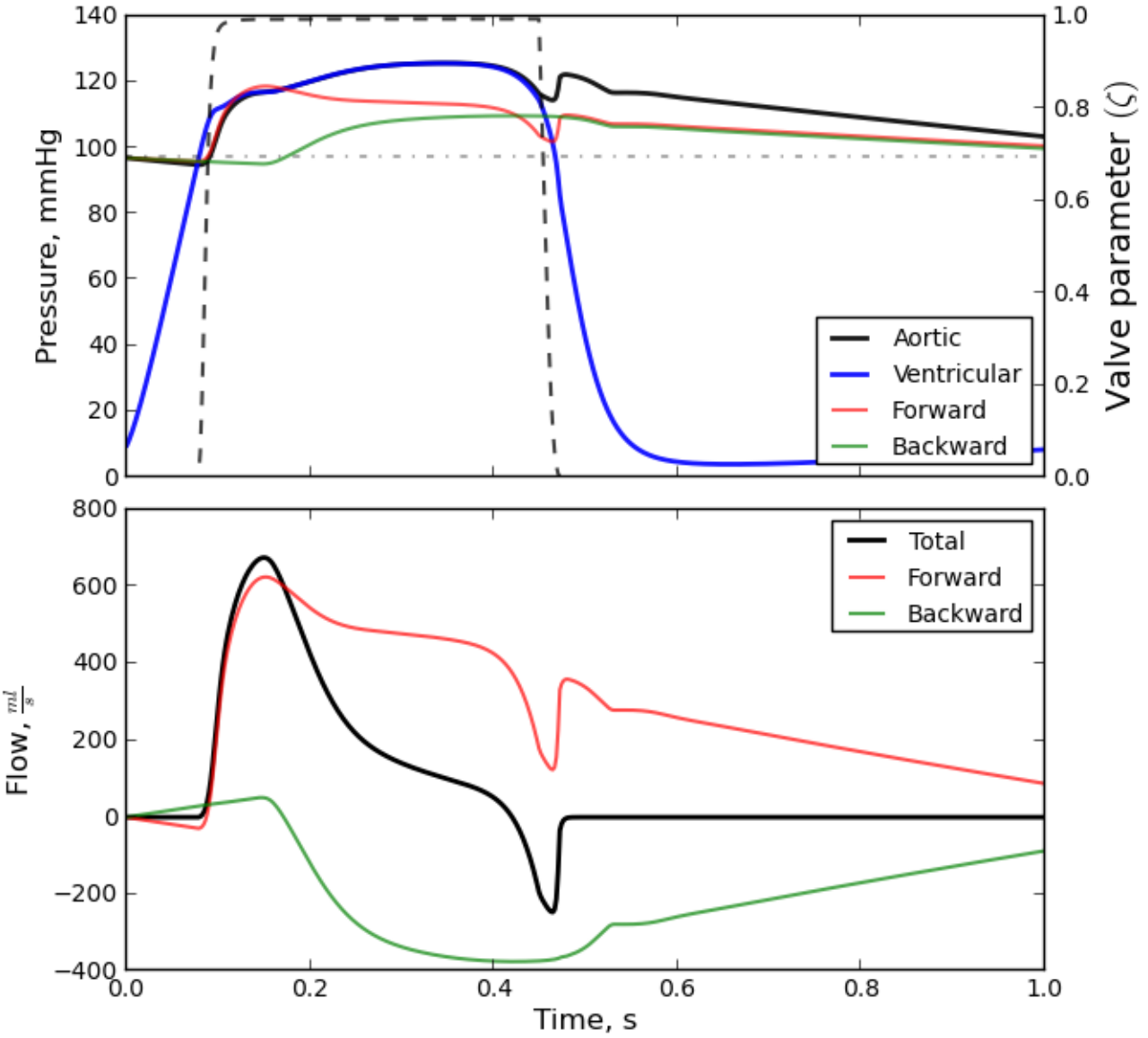


Figure 22: The same simulation as shown in figure 21 but with $K = 250\frac{s}{m^3}$ and a shorter length. The increased value of K smoothes out the large spike in early systole giving a more stable solution.

When using a low impedance long vessel the reflected waves become the dominant determining factor of the input impedance. At the beginning of systole when there are no reflections input impedance is low, causing the spike in ventricular outflow. At the end of systole the input impedance increases due to the large influence of reflected waves. In this particular simulation set-up with just a single vessel the variations in impedance become too large as can be seen in figure 21. The absence of any wave reflections whatsoever in early systole gives an impedance that is too low, giving very a very high flow rate (peaking at $1200\frac{ml}{s}$).

This leads to the conclusion that to get good simulation results in terms of both flow and pressure, some early reflections are needed to limit the outflow, but that the impedance generally needs to be low at the beginning of systole and high at the end of systole.

The major reflection should also be somewhat delayed so that a larger part of it comes after the aortic valve has closed. The weakness of a single vessel simulation is that it can not provide all these characteristic in one simulation. In a long vessel the reflections are delayed creating the dicrotic notch, but the impedance in early systole becomes too low. In a short vessel the reflection arrives too early.

The valve model. There is a great deal of uncertainty concerning the valve model and perhaps most of all when it comes to parameters. The size of the rate coefficients K_{vo} and K_{vc} seem to be not very important as in most cases the valve opens and closes very quickly. The important feature of the valve is that it opens and closes and does so at the right time.

The opening of the valve is no problem, the valve begins to open because the isovolumic contraction of the LV increases ventricular pressure to a level higher than the aortic pressure. At this phase the pressure gradient is always positive because both $Q|Q|$ and $\frac{dq}{dt}$ are positive, thus the valve continues till it is fully open. The threshold pressure Δp_{open} can be set to zero without getting in trouble.

The problem lies in the closing of the valve. The pressure gradient is in many cases negative during a large part of systole [4]. This is particularly true in a healthy valve, in a stenosed valve the resistance part of the pressure gradient dominates and gives a positive pressure gradient as long as the outflow from the ventricle is positive. In a healthy valve the inertance term becomes dominant and causes the pressure gradient to become negative when the flow has reached its peak early in systole and starts to decline. Using a zero pressure threshold for closing thus leads to the valve closing prematurely. Some experimentation with the pressure threshold was necessary. Setting it too low leads to premature valve closure, as shown in figure 21. Setting it too high gives a very large backflow into the ventricle before the valve closes.

All in all there is room for improvement in determining the parameters for the valve model. The most important trait however is that it closes at the right time, this can be achieved by making adjustments to the Δp_{close} parameter.

8.2 Verification of the Windkessel boundary condition and comparison of the existing and alternative discretization schemes

The two-element Windkessel boundary condition (WK2) was tested by comparing the simulated reflection of a single pulse to analytical solution. The discretization of WK2 is described in section 7.3, the metod for obtaining an analytical solution is described in 7.3.3. The parameters for the simulation are shown in table 7.

Parameter	Value
Pulse half length (t0), s	0.05
Pulse half period (L), s	1.0
Vessel impedance (Z), $\frac{\text{mmHg}s}{\text{ml}}$	0.2
Vessel length (l), m	0.5
Vessel area (A), m^2	3.14×10^{-4}
Windkessel resistance (R), $\frac{\text{mmHg}s}{\text{ml}}$	1.0
Windkessel compliance (C), $\frac{\text{mmHg}s}{\text{ml}}$	0.5

Table 7: Parameters used for the prescribed pulse, the geometry and impedance of the vessel and for the windkessel model

The reflected wave was recorded and superposed on a plot of the fourier series solution. The results are shown in figure 23. As can be seen the simulation results converge for smaller time steps. When comparing the possible discretization schemes it is seen that the 2nd order

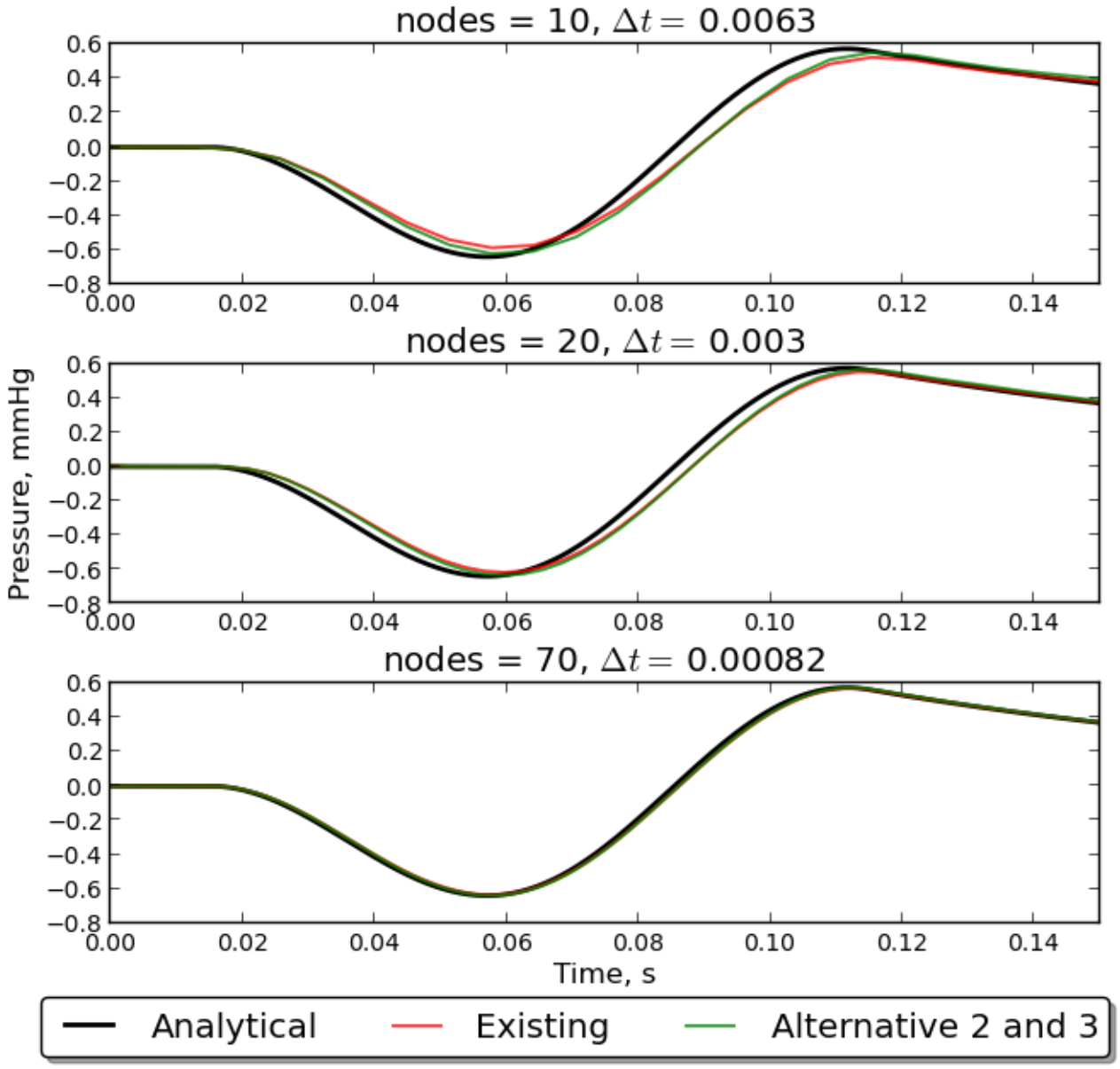


Figure 23: A comparison of numerical results from vascular1Dflow, using different numerical schemes, with analytical results for decreasing time-step sizes. The new algorithms show a slight improvement over the existing ones for large timestep sizes

backward difference scheme and the half step central difference scheme give nearly identical results. They both represent an improvement over the incremental solution especially at large time steps. All the alternatives converge to the correct solution.

Non-linear result. The algorithms were also tested for a larger pulse amplitude. The pulse amplitude was thus set to 100 mmHg. Because of the resulting non linear effects an analytical solution could not be obtained. Instead a reference simulation was made using the new discretization scheme and the node number set to 300 thus guaranteeing a converged result. Simulation results using a lower number of nodes was then compared to this converged solution. Alternatives 1 and 2 were still almost identical and were not plotted separately. The results are shown in figure 24

A similar pattern can be seen in the non-linear solution. The alternative discretization schemes converge slightly faster.

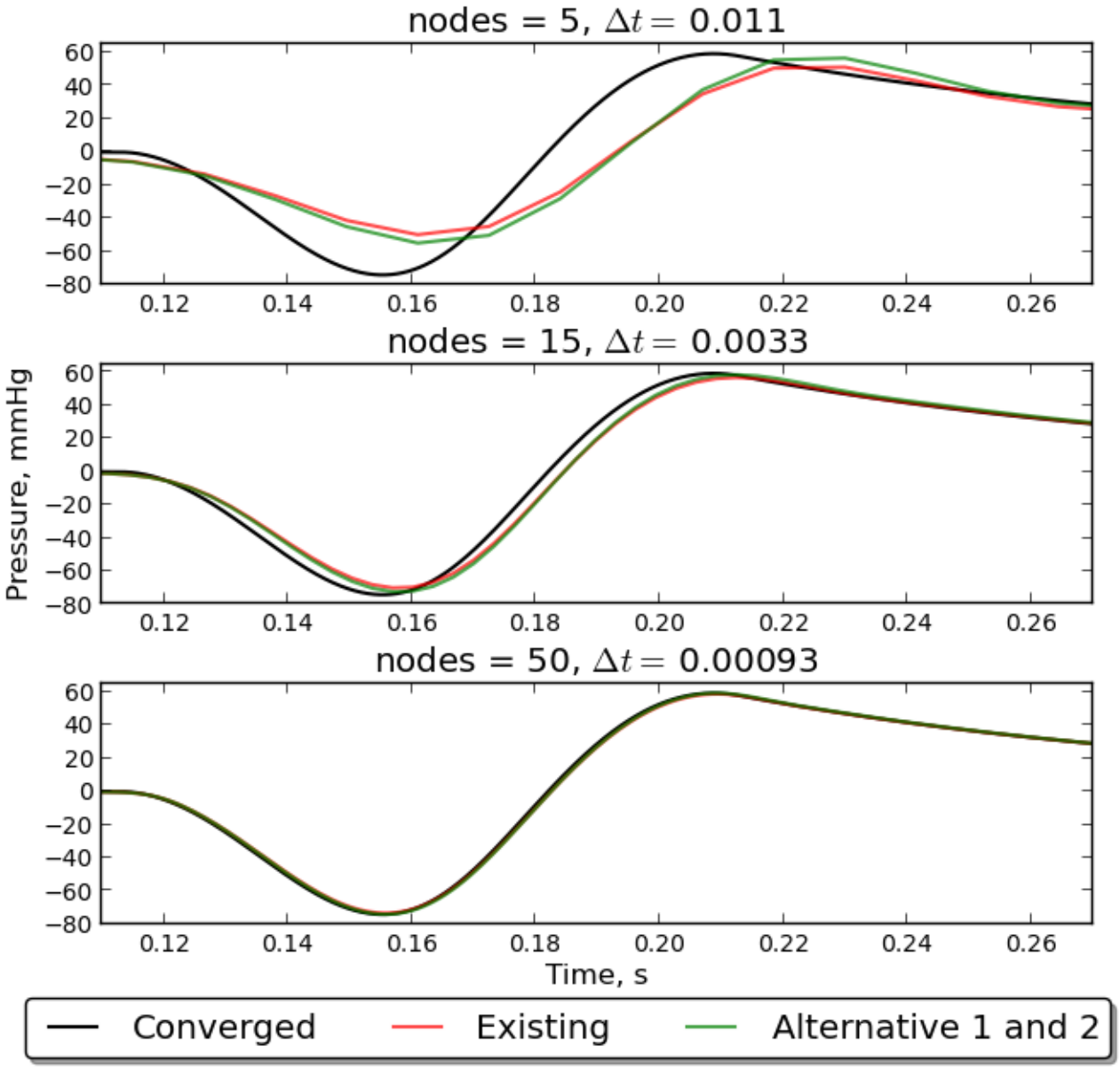


Figure 24: A comparison of numerical results from vascular1Dflow, using different numerical schemes, with analytical results for decreasing time-step sizes. The new algorithms show a slight improvement over the existing ones for large timestep sizes

8.3 Testing of the new bifurcation algorithm and comparison of computational efficiency

The bifurcation equations, being non-linear, can not be solved analytically. Just as in the case of the boundary conditions it is difficult to verify the accuracy of the results because of the non-linear nature of the equations. For small pressure-flow fluctuations however conditions are almost linear. A minimum requirement for the solver should therefore be to be able to give a correct reflection of a single pulse according to the formula for the reflection coefficient of the bifurcation given in equation 5.40. A test is set-up similar to the verification of the existing bifurcation algorithm presented in Leinan [10], to see if the new algorithm can handle this minimum requirement.

Linear case The network setup, consists of three vessels forming a single bifurcation. All three vessels have different cross-sectional areas thus giving them different impedances. At the

proximal end of the mother vessel a forward propagating pressure pulse is prescribed according to the equation 7.51. The values $a = 1\ mmHg$ and $t0 = 0.05\ s$ are used as the parameters fot the pulse.

This setup should cause a reflection of the pulse as it hits the bifurcation having propagated through the mother vessel. The pulse is then transmitted into the two daughter vessels. In the linear case the pressure is transmitted equally into the two daughter vessels according to the transmission coefficient in equation 5.40. However due to the different impedances, flow is not distributed equally between the two vessels.

All three vessels are given the same Young's modulus, wall thickness and length: $E = 4e5\ Pa$, $h = 0.1\ cm$ and $l = 20\ cm$. The blood mass density is set to $\rho = 1060\ \frac{kg}{m3}$. Using the Laplacian compliance model (eq. **??**), the characteristic impedance of a vessel is given by:

$$Z_c = \sqrt{\frac{\rho\sqrt{\pi}hE}{2A^{\frac{5}{2}}}} \tag{8.2}$$

The cross sectional areas with corresponding characteristic impedances are given in table 8:

Vessel	$A[cm^2]$	$Z_c[\frac{\text{mmHg s}}{\text{ml}}]$	Γ	T
1	2	0.011	0.67	1.67
2	0.5	0.061	-0.71	0.29
3	0.1	0.459	0.039	1.039

Table 8: Cross-sectional areas of the vessels, characteristic impedances and reflection and transmission coefficients

Boundary conditions and reflections. To further test the system the two daughter vessels are terminated differently. One with a full reflection ($\Gamma = 1$) and the other with no reflection ($\Gamma = 0$). Thus a secondary reflection travels back through the system and hits the bifurcation, is transmitted into the mother vessel and right daughter vessel. This is also recorded at the proximal end of the mother vessel. This pulse has then traveled through the bifurcation twice. The first reflection arriving at the root of the mother vessel should therefore have the amplitude $a_1 = a\Gamma_1 = 0.67$ mmHg. The next reflection is transmitted first according to $T_1 = 1.67$ then completely reflected and transmitted again according to $T_2 = 0.29$. It should have amplitude $a_2 = aT_1T_2 = 0.49$ mmHg. The results are shown in figure 25

The reflected waves have the correct amplitudes. This shows that the new algorithm works just as well as the existing algorithm at linear conditions.

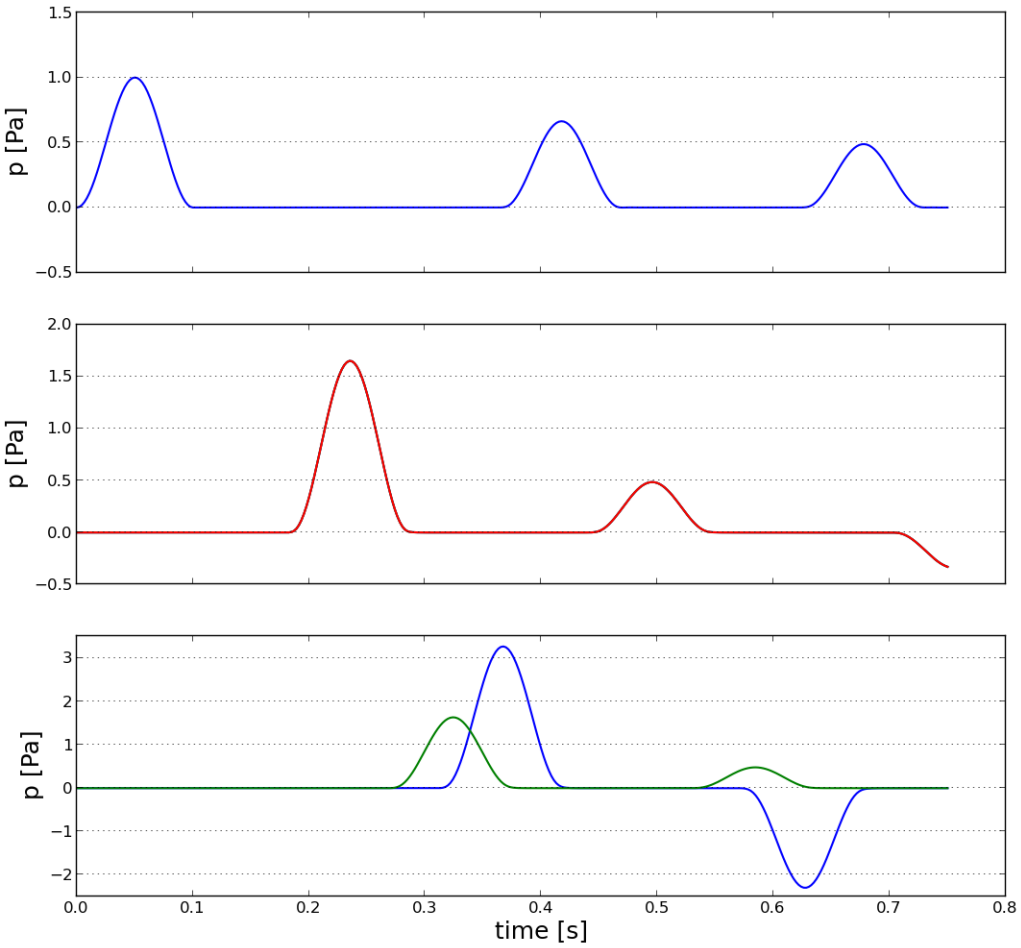


Figure 25: Plot 1: Shows pressure at the root of vessel 1, the first peak is the outgoing wave the next two are the reflected waves. As can be seen they have the correct amplitudes. Plot 2: Pressure in all three vessels at the bifurcation. All pressures are equal due to near-linear conditions. Plot 3: Pressure at the distal end of vessels 2 and 3. In vessel 2 (blue line) one can see that the pressure has been reflected negatively at the bifurcation

Non-linear case - comparison of the two algorithms The real test of the new bifurcation algorithm is to see if it can outperform the existing algorithm in computational efficiency. This was done on the same single bifurcation network as the previous test. The pressure amplitude was now set to 50 mmHg and the pulse was run through the bifurcation. To avoid problems with negative pressures the reflecting boundary condition was removed. Still the high pressure amplitude in combination with the large differences in cross sectional area should provide a challenge for the algorithms.

In all the tests the pressure and flow results from the two algorithms were identical and are therefore not shown. Table 9 shows the results in terms of increments, subiterations and total time required to complete the simulation.

In both cases the number of subiterations refers to the number of function calls. The *fsolve* algorithm did only one computation of the Jacobian matrix per increment. This approach was also used in the implementation of the new algorithm on the suspicion that the equations to be solved were actually quite linear. As can be seen from the table there was a significant difference

Algorithm	Increments	Increments with subiterations	Total subiterations	Subiterations per increment	Total solution time, s
Existing	142	112	533	4.76	1.074
New	142	107	287	2.68	0.774

Table 9: A comparison of the computational efficiency of the existing and new algorithm. The subiterations per increment values does not take into account the increments where no iterations were needed. The total solution time is the average of three simulation runs

in the number of subiterations. This might imply that the error tolerance of existing bifurcation algorithm is too strict.This proved to be the biggest problem with the existing implementation.

Improvements on the existing algorithm. Having found the tolerance to be too strict it was now set at a much higher level (i.e. xtol = 0.1). Which typically gave two iterations per timestep as opposed to around five (or even more) when using the default tolerance. This proved to be more than enough to get accurate results.

Also the tolerance for reusing the solution from the previous timestep could be increased to $||\boldsymbol{f}|| < 1$ without losing accuracy.

The modified existing algorithm was tested and the results are shown in table 10.

One iteration. The new algorithm was finally tested using only one iteration per increment. Also the tolerance for reusing the values from the previous time step was now set to 1. There were still no differences in the produced results. The performance results are shown in table 10.

Algorithm	Increments	Increments with subiterations	Total subiterations	Subiterations per increment	Total solution time, s
Existing modified	142	102	295	2.89	1.015
One iteration	142	64	64	1.0	0.68

Table 10: The existing algorithm with modified tolerances and the new algorithm limited to only one iteration per increment

The modifications to the existing algorithm greatly reduced the number of iterations however the computation time did not reduce significantly. This might suggest that it is the the computation and inversion of the Jacobian matrix that takes up the most computational power since this still is called only once per increment. The fact that the new algorithm worked so well with only one iterations proves that the equation system is very linear and that there is no big need for an iterative procedure for the boundary conditions.

Instability caused by low CFL-numbers The CFL condition for stability demands that $CFL = \frac{c\Delta t}{\Delta z} \leq 1.0$. In *vascular1Dflow* this condition is fulfilled by setting the time step size so that this condition is fulfilled at the vessel (or node) where the ratio $\frac{c}{\Delta z}$ is greatest. If the number of nodes in each vessel is not chosen so that $\frac{c}{\Delta z}$ is approximately equal in all vessels the CFL-numbers will also be different for each vessel. Tests revealed that there was some intability in the solution if one or more of the vessels in the bifurcation had a low CFL-number. This can be seen in the oscillations in figure 26. The simulation was made using the same set-up as in figure 25, but the node number of vessel 1 was reduced to 42 giving the vessel a CFL number of

0.486, whereas the other two vessels still had CFL = 0.99. The resulting oscillations are shown in figure 26 The reason for this instability is unknown. The results were identical in the two

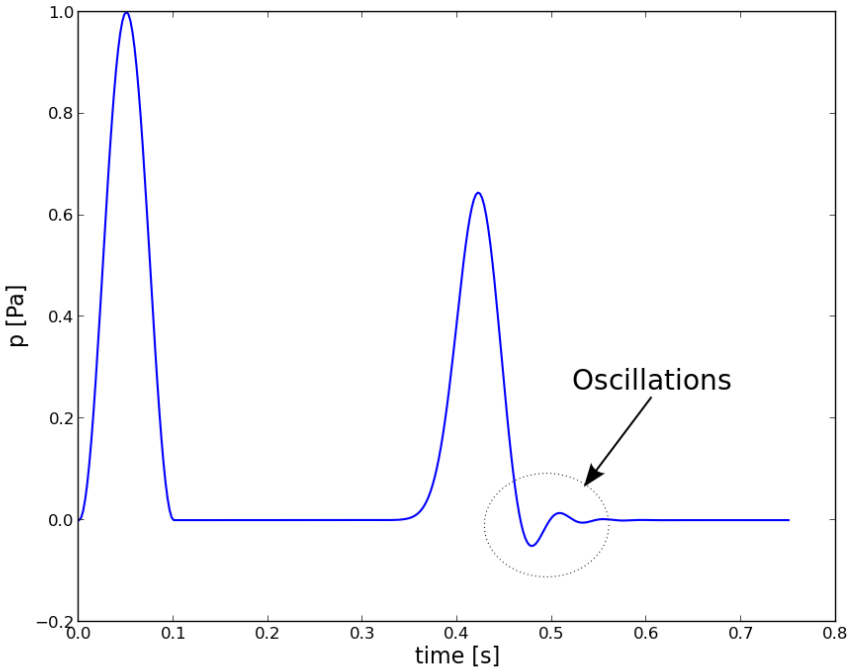


Figure 26: Differences in CFL-numbers causes som unwanted oscillations.

different bifurcation algorithms suggesting that the cause is something else. This shows that it is important to have the correct number of nodes to get good results at bifurcations.

8.4 Arterial network with varying elastance

The purpose of the test case is to see if the varying elastance when connected to a full size arterial network model of a human is able function properly, in particular to see if a realistic waveform can be produced and how these results compare to the results achieved in the single vessel tests.

It is also meant to demonstrate all the implementations and algorithm improvements working together, the varying elastance boundary condition with the dynamic valve model and the improvements on the bifurcation and windkessel models.

Arterial network. The arterial network shown in figure 27, is based on a already existing XML file in *vascular1Dflow* used for simulations in the master thesis of Vinzenz Eck [3]. The geometry and material data for this network is based on Stergiopulos et al. [15]. Vessel data are given in table 11

id	Name	L[m]	r_p[mm]	r_d[mm]	h[mm]	E[MPa]
0	Ascending Aorta	0.04	14.7	14.4	1.63	0.4
1	Aortic Arch I	0.02	11.2	11.2	1.26	0.4
2	Brachiocephalic	0.034	6.2	6.2	0.8	0.4
3	R.Subclavia I	0.034	4.23	4.23	0.67	0.4
4	R.Carotid	0.177	3.7	3.7	0.63	0.4
5	R.Vertebral	0.148	1.88	1.83	0.45	0.8

6	R.Subclavia II	0.422	4.03	2.36	0.67	0.4
7	R.Radial	0.235	1.74	1.42	0.43	0.8
8	R.Ulnar I	0.067	2.15	2.15	0.46	0.8
9	R.Interosseous	0.079	0.91	0.91	0.28	1.6
10	R.Ulnar II	0.171	2.03	1.83	0.46	0.8
11	R.Internal Carotid	0.176	1.77	0.83	0.45	0.8
12	R.External Carotid	0.177	1.77	0.83	0.42	0.8
13	Aortic Arch II	0.039	10.7	10.7	1.15	0.4
14	L.Carotid	0.208	3.7	3.7	0.63	0.4
15	L.Internal Carotid	0.176	1.77	0.83	0.45	0.8
16	L.External Carotid	0.177	1.77	0.83	0.42	0.8
17	Thoracic Aorta I	0.052	9.99	9.99	1.1	0.4
18	L.Subclavian I	0.034	4.23	4.23	0.66	0.4
19	L.Vertebral	0.148	1.88	1.83	0.45	0.8
20	l.Subclavian II	0.422	4.03	2.36	0.67	0.4
21	L.Radial	0.235	1.74	1.42	0.43	0.8
22	L.Ulnar I	0.067	2.15	2.15	0.46	0.8
23	L.Interosseous	0.079	0.91	0.91	0.28	1.6
24	L.Ulnar II	0.171	2.03	1.83	0.46	0.8
25	Intercostales	0.08	2	1.5	0.49	0.4
26	Thoracic Aorta II	0.104	6.75	6.45	1	0.4
27	Abdominal I	0.053	6.1	6.1	0.9	0.4
28	Celiac I	0.02	3.9	2	0.64	0.4
29	Gastric	0.071	1.8	1.8	0.45	0.4
30	Splentic	0.063	2.75	2.75	0.54	0.4
31	Superior Mesenteric	0.059	4.35	4.35	0.69	0.4
32	Abdominal II	0.02	6	6	0.8	0.4
33	R.Renal	0.032	2.6	2.6	0.53	0.4
34	Abdominal IV	0.116	5.8	5.2	0.75	0.4
35	R.Common Iliac	0.058	3.68	3.5	0.6	0.4
36	L.Common Iliac	0.058	3.68	3.5	0.6	0.4
37	L.External Iliac	0.144	3.2	2.7	0.53	0.8
38	L.Internal Iliac	0.05	2	2	0.4	1.6
39	L.Femoral	0.443	2.59	1.9	0.5	0.8
40	l.Deep Femoral	0.126	2.55	1.86	0.47	0.8
41	L.Posterior Tibial	0.321	2.47	1.41	0.45	1.6
42	L.Anterior Tibial	0.343	1.3	1.3	0.39	1.6
43	R.External Iliac	0.144	3.2	2.7	0.53	0.8
44	R.Internal Iliac	0.05	2	2	0.4	1.6
45	R.Femoral	0.443	2.59	1.9	0.5	0.8
46	R.Deep Femoral	0.126	2.55	1.86	0.47	0.8
47	R.Posterior Tibial	0.321	2.47	1.41	0.45	1.6
48	R.Anterior Tibial	0.343	1.3	1.3	0.39	1.6

Table 11: Vessel data for an arterial network taken from Stergiopulos et al. [15].

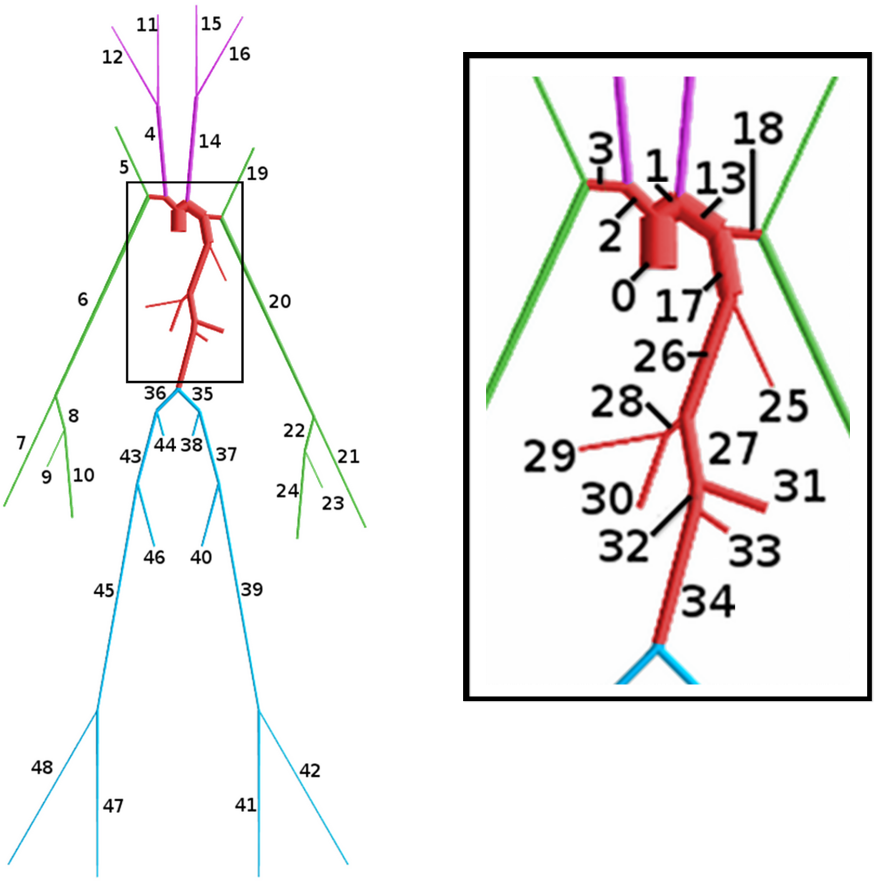


Figure 27: The arterial network used for the simulations. Figure taken from Eck [3]

Boundary conditions. The boundaries are all terminated by three element windkessel models. The boundary condtions are thus able to account for the compliance and resistance found in the smaller vessels after the point of termination [15]. The characteristic resistance of the winkessel models were set to be equal to the characteristic impedances of the vessels thus absorbing higher frequency signals. This is done by *vascular1Dflow* automatically by not declaring this parameter. It is also implemented so that the value used by the boundary condition is updated for each timestep to the current impedance of the vessel which changes with pressure. The peripheral resistances and compliances were also taken from Stergiopulos et al. [15] and is presented in table **??**.

It should be noted that the compliances of the boundary conditions are relatively small making the BCs function almost like pure resistors. This also means that the compliances of the BCs make only a small contribution of the system. For the resistances it is clear that it is the other way around, where the BCs provide much more resistance than the viscous resistances of the network.

Vessel ID	Resistance $[\frac{\text{mmHg s}}{\text{ml}}]$	Compliance $[\frac{\text{ml}}{\text{mmHg}}]$
5	45.08	0.004127
7	39.60	0.004698
9	632.30	0.0002942
10	39.60	0.004698
11	104.26	0.001784
12	104.26	0.001784
15	104.26	0.001784
16	104.26	0.001784
19	45.08	0.004127
21	39.60	0.004698
23	632.30	0.0002942
24	39.60	0.004698
25	10.43	0.01784
29	40.58	0.004585
30	17.40	0.01069
31	6.98	0.02667
33	8.48	0.02195
38	59.52	0.003125
40	35.78	0.005110
41	35.78	0.005110
42	41.93	0.004437
44	59.52	0.003125
46	35.78	0.005110
47	35.78	0.005110
48	41.93	0.004437

Table 12: Compliance and total resistance of all terminal boundary conditions

8.4.1 Network results

The varying elastance BC is initialized with the same parameters as in the single vessel simulations, as given in table **??**. When using all these standard parameters there was no problem with the aortic valve. However to be on the safe side a small threshold pressure for the closing of the aortic valve was used. The aortic valve parameters were thus set to $K_{vo} = 0.12$, $K_{vc} = 0.12$, $\Delta p_{\text{open}} = 0$, $\Delta p_{\text{close}} = 2.0$mmHg and $M_{st} = 0.99$. The total simulation time is 3.0s, giving three heart cycles with the data from the last cycle being presented. The initial pressure is set to 80 mmHg. The results from the simulation using the standard parameters from tables 12 and 11 are shown in figure 28

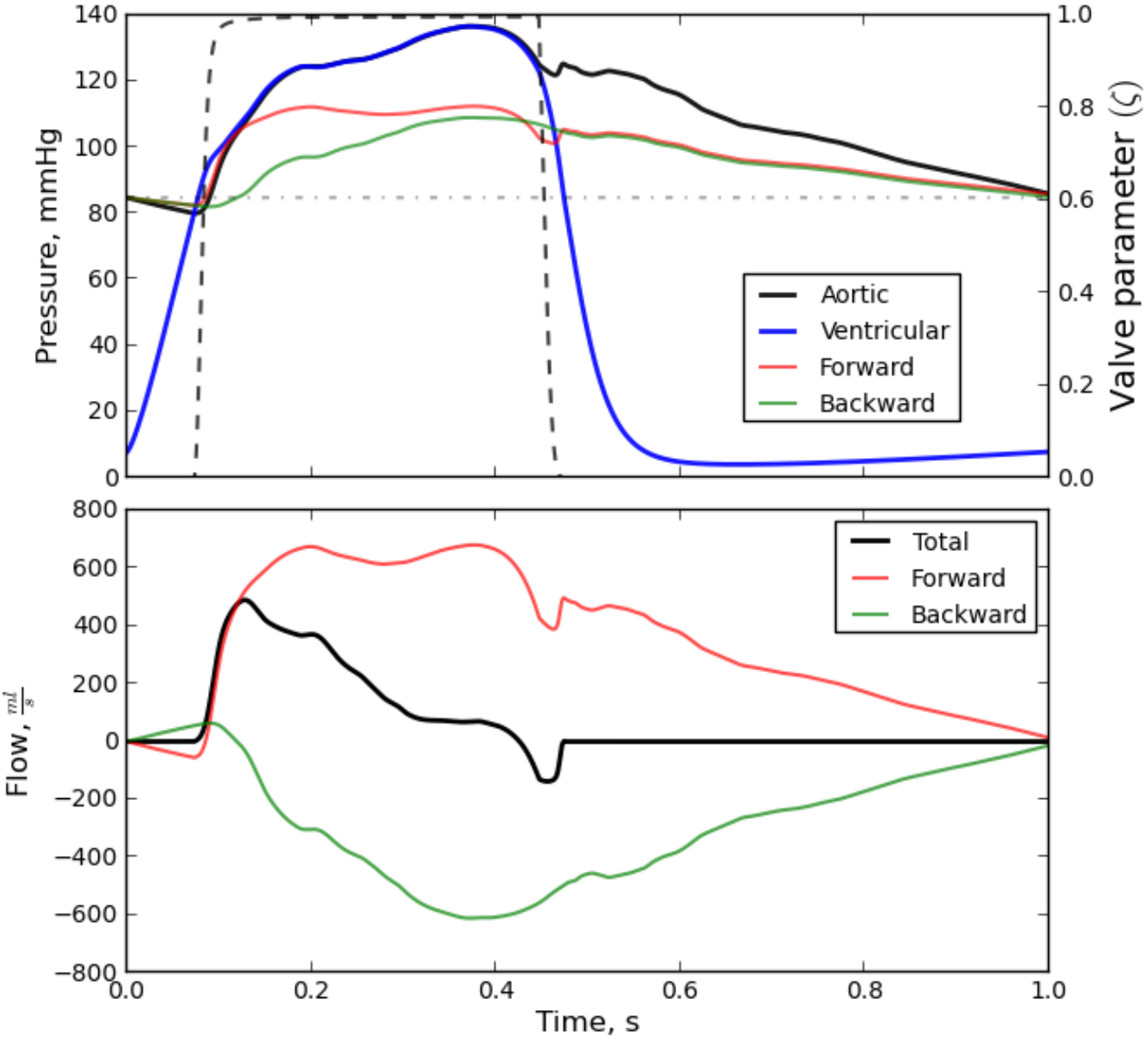


Figure 28: Varying elastance simulation on an arterial network. The opening and closing of the aortic valve represented by the valve parameter ζ is given by the dashed line. The horizontal dotted line shows the reference pressure.

Variations in vessel stiffness Two simulations were made on the network where the stiffness of the vessels were changed. This was done by multiplying all the vessels Young's moduli with a stiffness factor. The stiffness factors used were 0.6, 1.4. The pressure and flow in the aortic root of these simulations are shown in figures 29, 28 and 30 along with the valve opening parameter and the corresponding wavesplitted waveforms.

Changes in vessel stiffness have two important effects changing the total compliance of the network as well as changing the wavespeed. With the decreased stiffness, shown in figure 29, the compliance is reduced and so is the wavespeed. The decreased wavespeed makes reflections arrive later, this means that more of the total reflection comes after the valve has closed. In figure 30 the increased wavespeed makes more reflections during systole thus increasing the load on the heart.

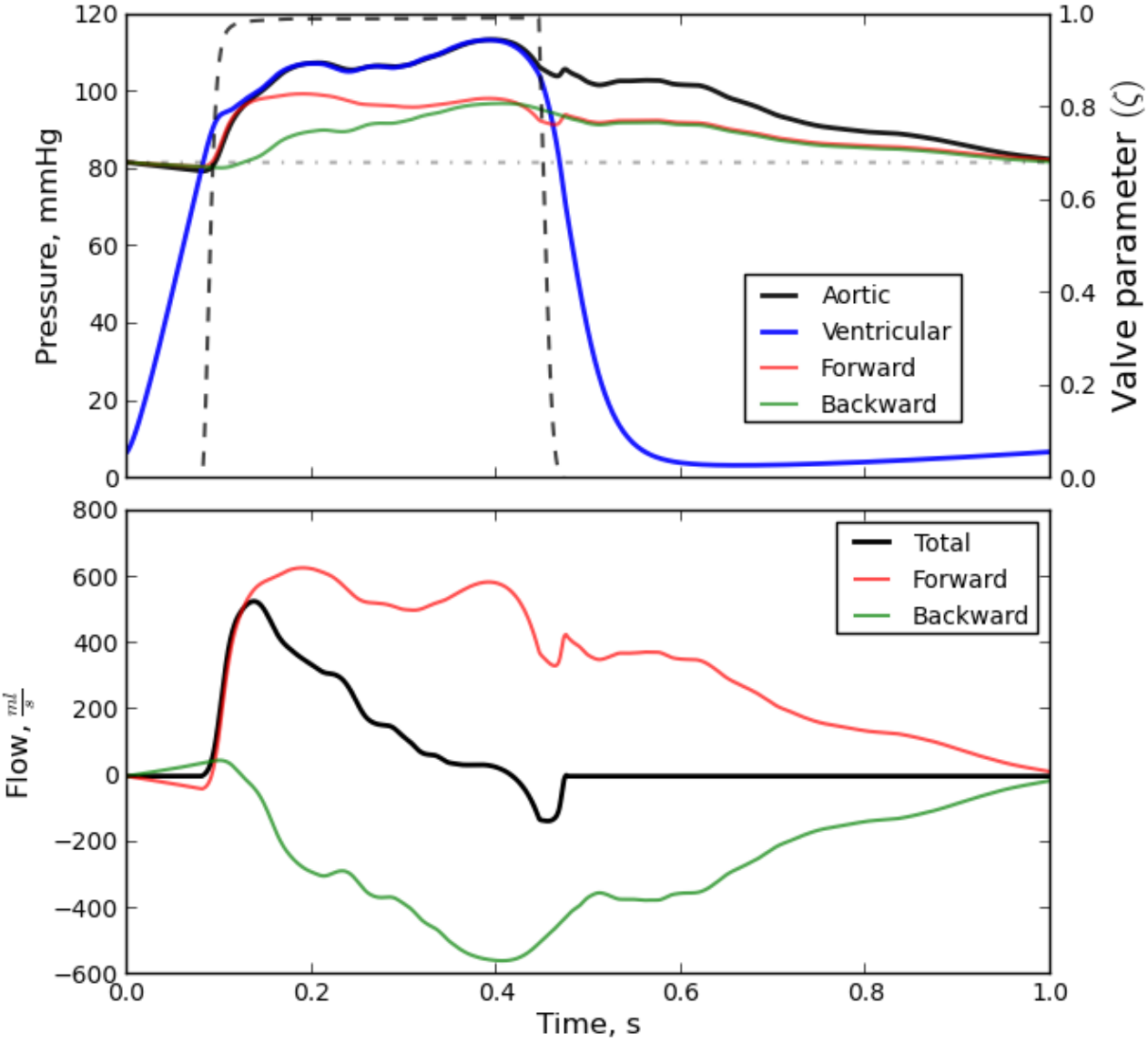


Figure 29: Varying elastance simulation on an arterial network. Vessel stiffnesses are scaled by a stiffness factor of 0.6. The opening and closing of the aortic valve represented by the valve parameter ζ is given by the dashed line. The horizontal dotted line shows the reference pressure.

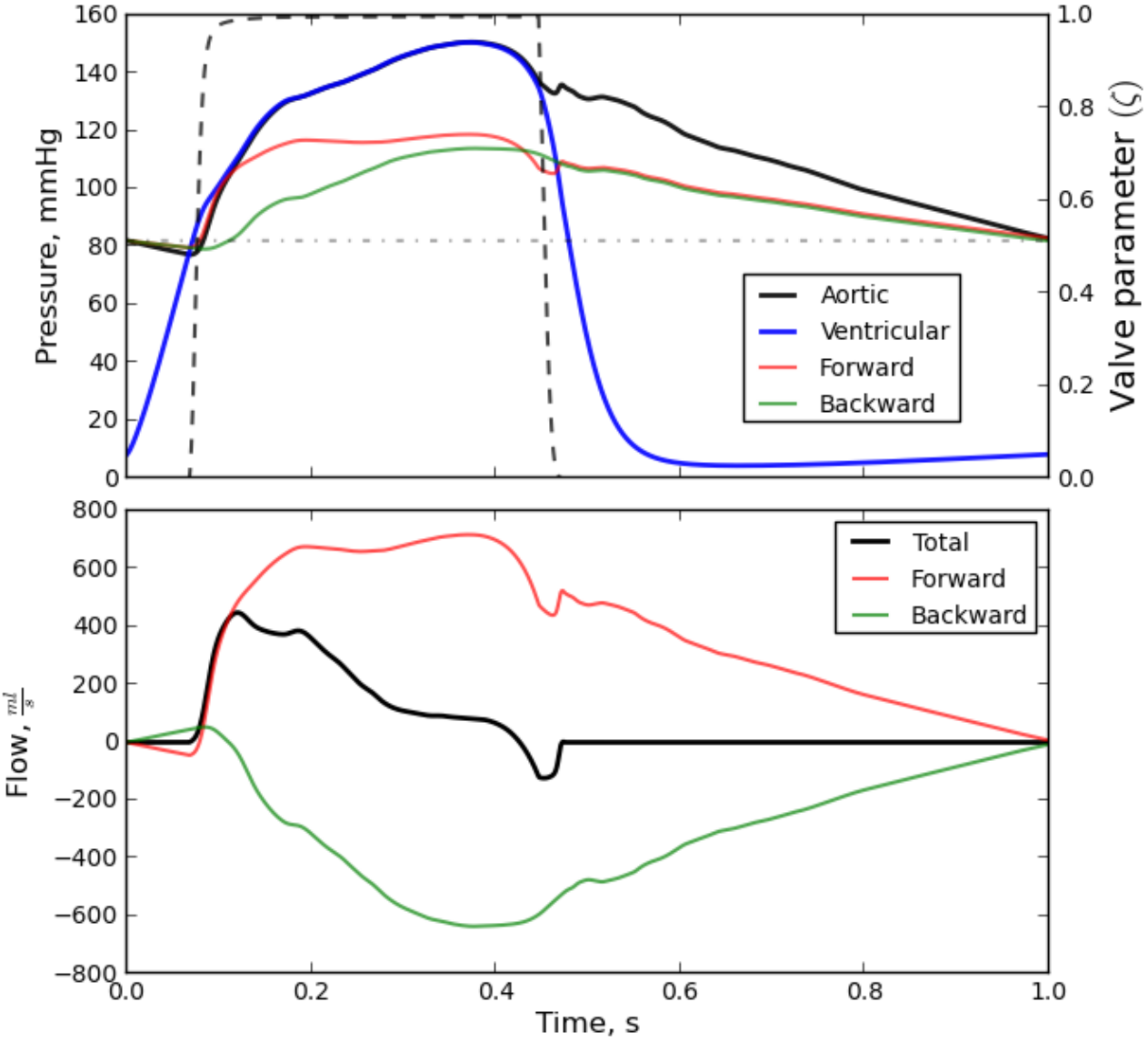


Figure 30: Varying elastance simulation on an arterial network. Vessel stiffnesses are scaled by a stiffness factor of 1.4. The opening and closing of the aortic valve represented by the valve parameter ζ is given by the dashed line. The horizontal dotted line shows the reference pressure.

Variations in afterload. The same network simulation was made but now changing the afterload of the heart by varying the resistances of the boundary conditions. The total resistance of each boundary conditions was now multiplied with the factors 0.5 and 1.5 while keeping all other parameters constant. The results can be seen in figures 31 and 32.

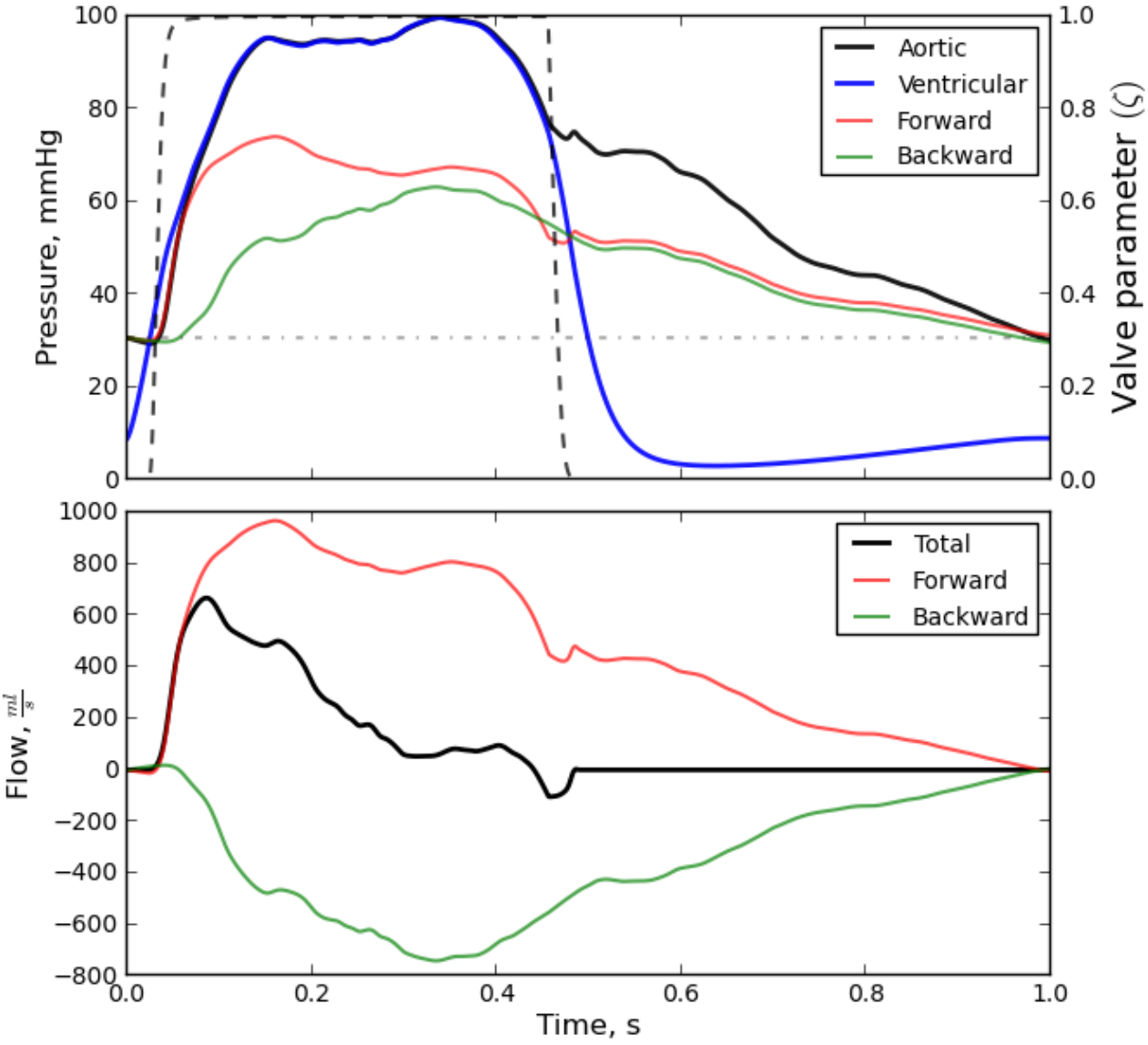


Figure 31: Varying elastance simulation on an arterial network. The boundary condtition total resistances were multiplied by a factor 0.5. The opening and closing of the aortic valve represented by the valve parameter ζ is given by the dashed line. The horizontal dotted line shows the reference pressure.

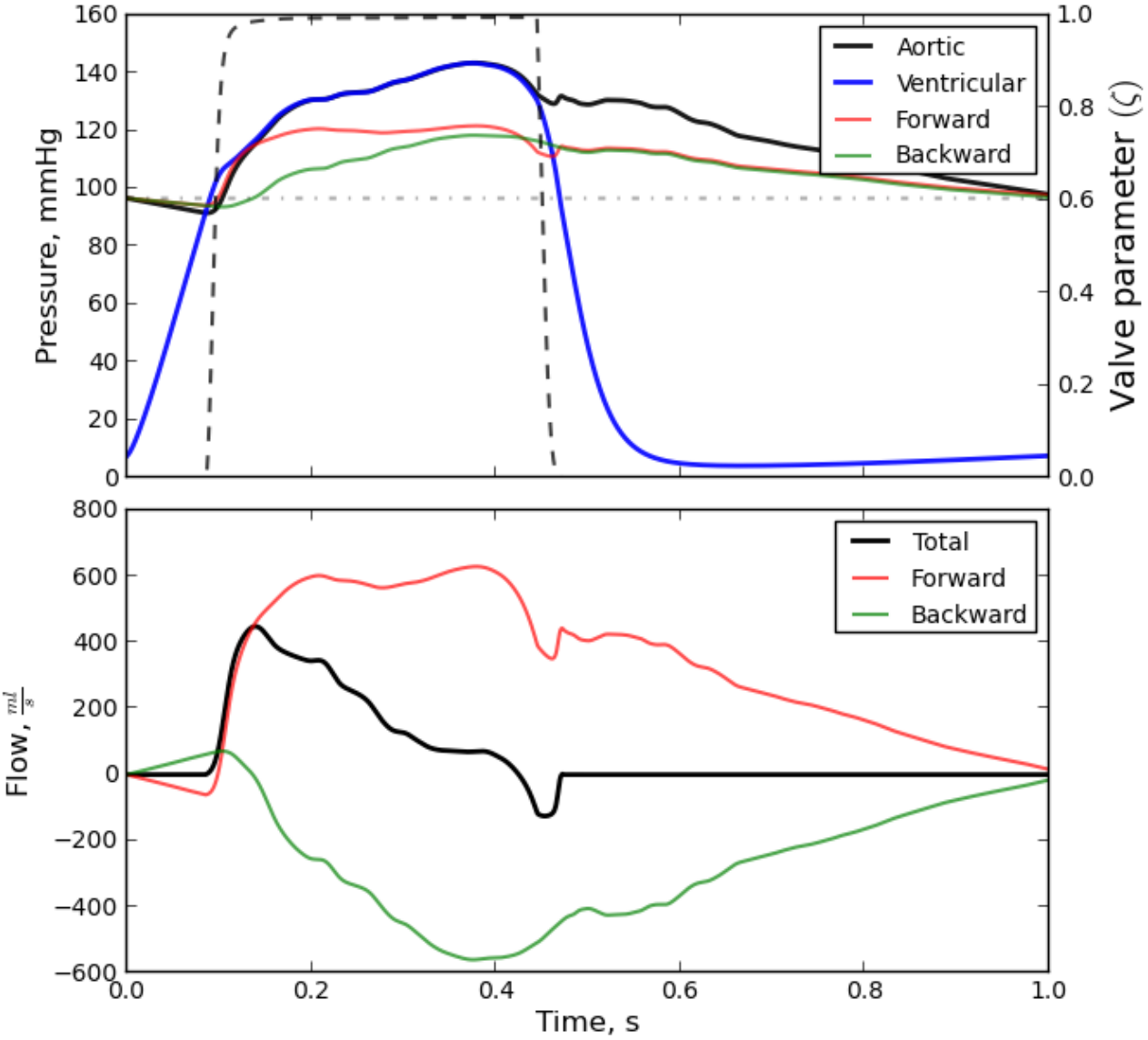


Figure 32: Varying elastance simulation on an arterial network. The boundary condtition total resistances were multiplied by a factor 1.5. The opening and closing of the aortic valve represented by the valve parameter ζ is given by the dashed line. The horizontal dotted line shows the reference pressure.

8.4.2 Network pressure decay

Some simple simulations were made on the network, where the point was to see the pressure in the network decayed similarly to a lumped windkessel model in the network. And to see if the decay rate of the simulation could be predicted based on vessel properties and boundary conditions.

Simulations where made by setting the initial pressure to 1 mmHg in the network and allowing it to decay throughout the total simulation time of 2.0 seconds. All the boundary conditions used were two element windkessel models using the parameter values given in 12, but with half the compliance. In the aorta the varying elastance model was replaced by a complete reflection ($\Gamma = 1$).

The pressure was recorded at the aortic root and τ was determined by curve-fitting the exponential decay function as given in eq.5.10. The results of one such curve fitting is shown in figure 33 where the simulated pressure is compared with the fitted analytical curve. The

simulations were repeated with different peripheral resistances and tau values were computed. The scaling factors used were 0.5, 1.0 and 1.5. The resulting pressures curves are shown in figure 33.

The compliance of the whole system was computed from vessel so that:

$$C_{tot} = C_{\text{network}} + C_{\text{BC}} \tag{8.3}$$

where the network compliance was computed as the sum of the volume compliances of all the vessels, which was computed based on the Laplace compliance. For the vessels with a tapered cross section this meant integrating along the length of the vessel giving the expression for the volume compliance of a vessel:

$$C_v = \frac{A_2^2 - A_1^2}{2\pi hE(r_2 - r_1)} L \tag{8.4}$$

where A_1 and A_2 are the cross sectional areas at each end and r_1 and r_2 are the radii. The total resistance of the network was computed from the boundary condition total resistances:

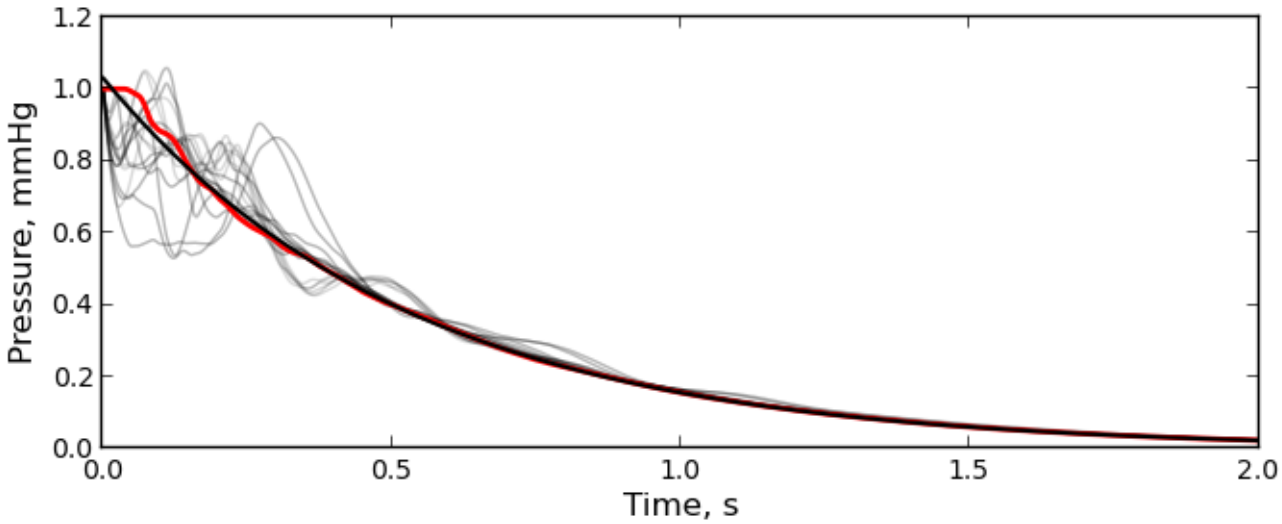
$$R_{tot} = \frac{1}{\sum_{BC} \frac{1}{R_t}} \tag{8.5}$$

the viscous resistance of the network itself was considered negligible. The measured and fitted pressures from the aortic root are shown in figure 33 along with plots of the pressures at all the other boundary nodes. The calculated and fitted values of τ are shown in table 13.

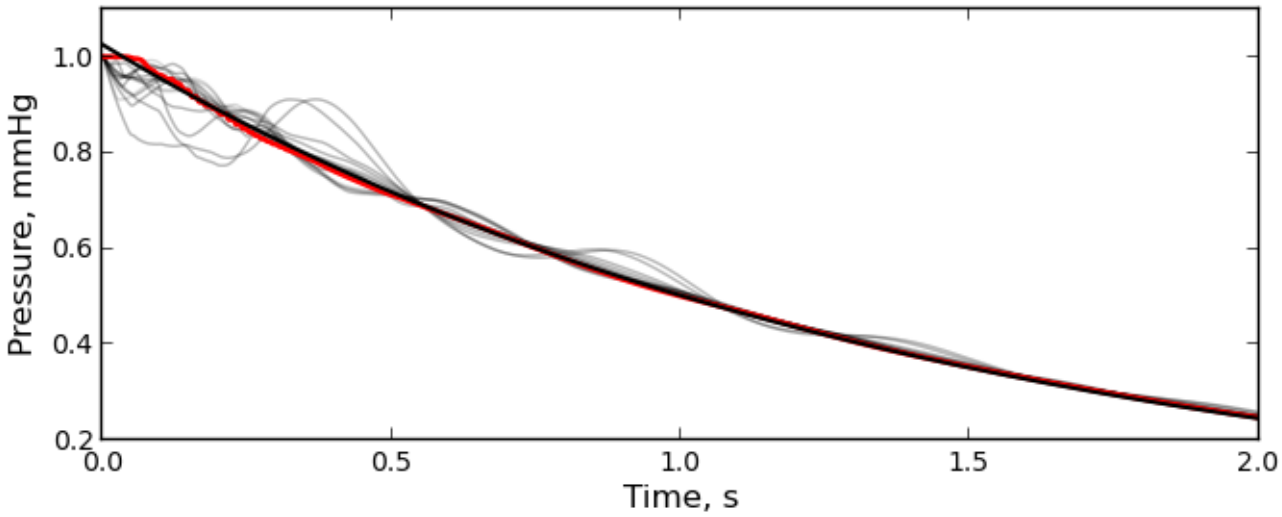
	$R_{tot}, \frac{\text{mmHg}s}{ml}$	$C_{tot}, \frac{\text{ml}}{\text{mmHg}}$	τ_{calc}, s	τ_{fit}, s
Simulation 1	0.61	0.878	0.53	0.530
Simulation 2	1.22	0.878	1.06	1.082
Simulation 3	1.83	0.878	1.59	1.637

Table 13: Table showing calculated and fitted values og τ

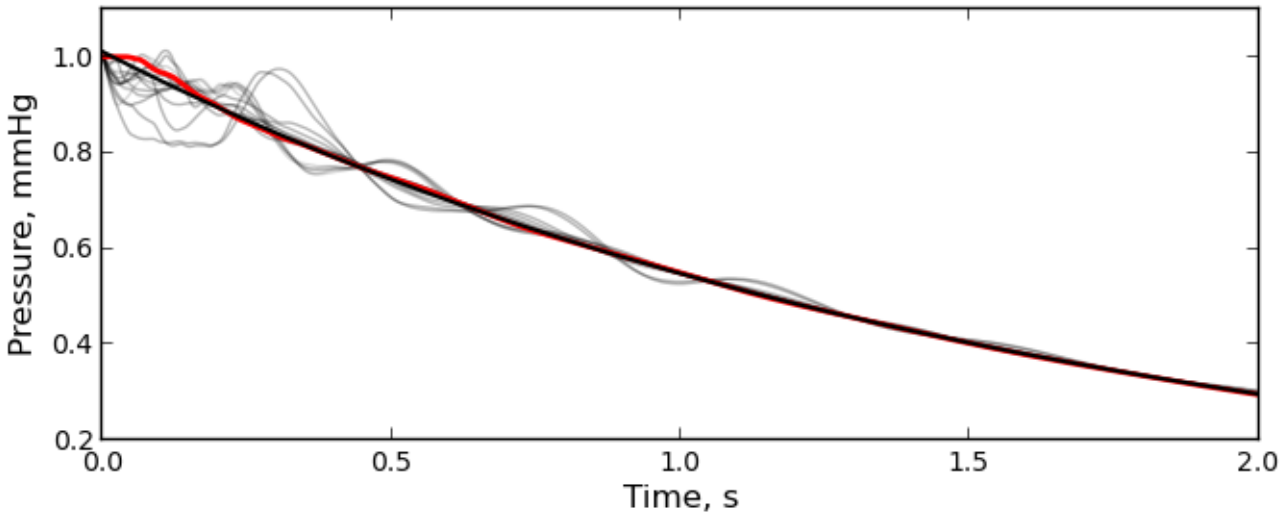
The intention of these pressure decay simulations was to investigate whether an arterial network model exhibits windkessel-like pressure decay which it clearly does. It is also interesting to see that this behaviour arises as the result of seemingly chaotic wave propagation which can be seen particularly well in figure 33a. One could also say that the rate of pressure decay was well predicted by the vessel and BC parameters with the biggest error being in the high resistance simulation.



(a) Resistance scaling factor 0.5



(b) Resistance scaling factor 1.0

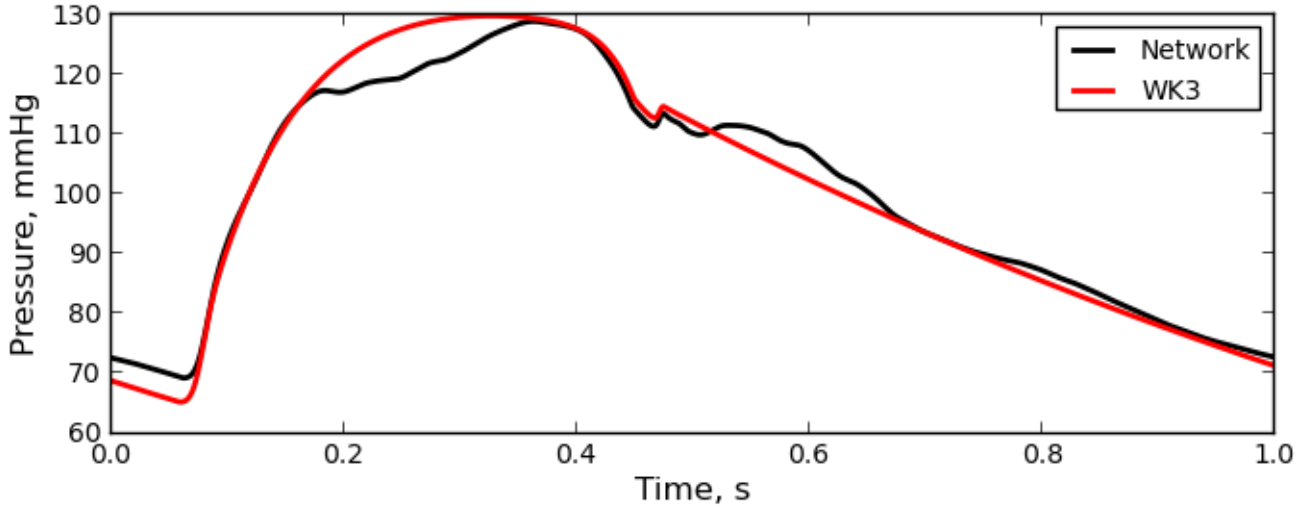


(c) Resistance scaling factor 1.5

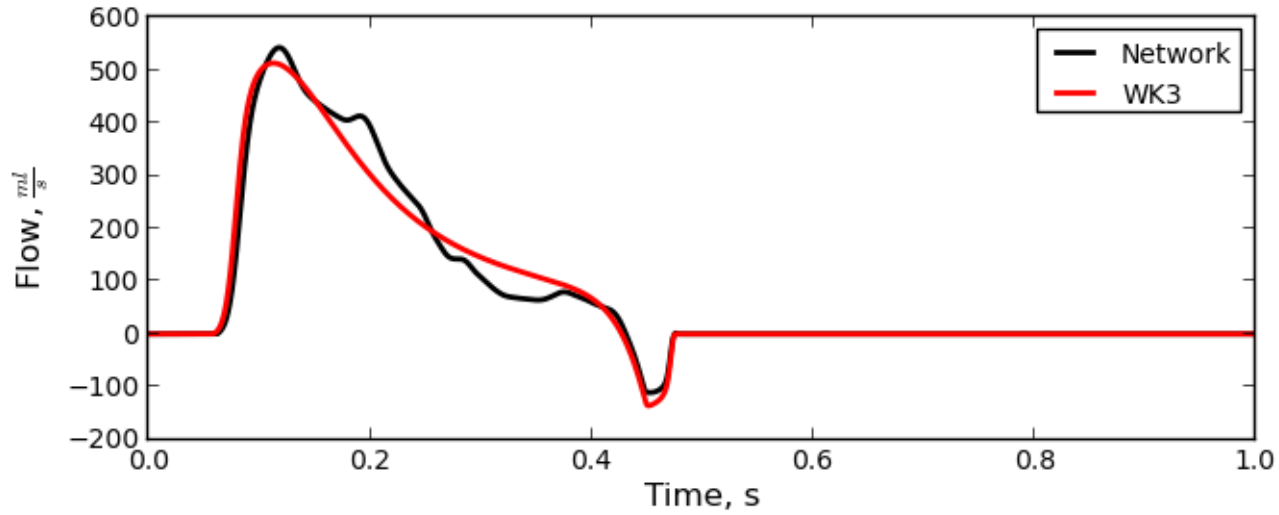
Figure 33: Exponential pressure decay in a network. The red line is the pressure at the aortic root, the thick black line is the fitted pressure decay function. The thin black lines are the pressures at all the boundary nodes. The results show that the complex behaviour of the system with wave propagation and reflections result in a reservoir-like behaviour.

8.4.3 Comparison of network and lumped model

To study the similarities and differences of the full network varying elastance simulation and lumped model simulations the results of two such studies were compared. The network simulation using the standard parameters, as shown in figure 28, was recreated as a lumped model simulation using a very short vessel and a three element windkessel model. The windkessel parameters C and R were computed from the network in the same way as described in the previous section. The results of this comparison is shown in figure 34.



(a) Comparison of pressures



(b) Comparison of flows

Figure 34: A comparison of pressure and flow at the aortic root between an arterial network and lumped model simulation

The comparison shows a significant similarity between the arterial network model and the three element windkessel model as an afterload on the heart. The maximum pressures are similar, the closing of the aortic valve happens at the same time and the diastolic pressure decay is similar. The main difference is that in the network model more of the reflected waves come after the aortic valve has closed

8.4.4 Reservoir-wave separation of pressure

The same simulation results were also subjected to the reservoir-wave separation technique described in section 5.7. The network compliance was C = 0.88 $\frac{\text{ml}}{\text{mmHg}}$ The wave pressure and reservoir pressure are shown in figure 35 along with the flow.

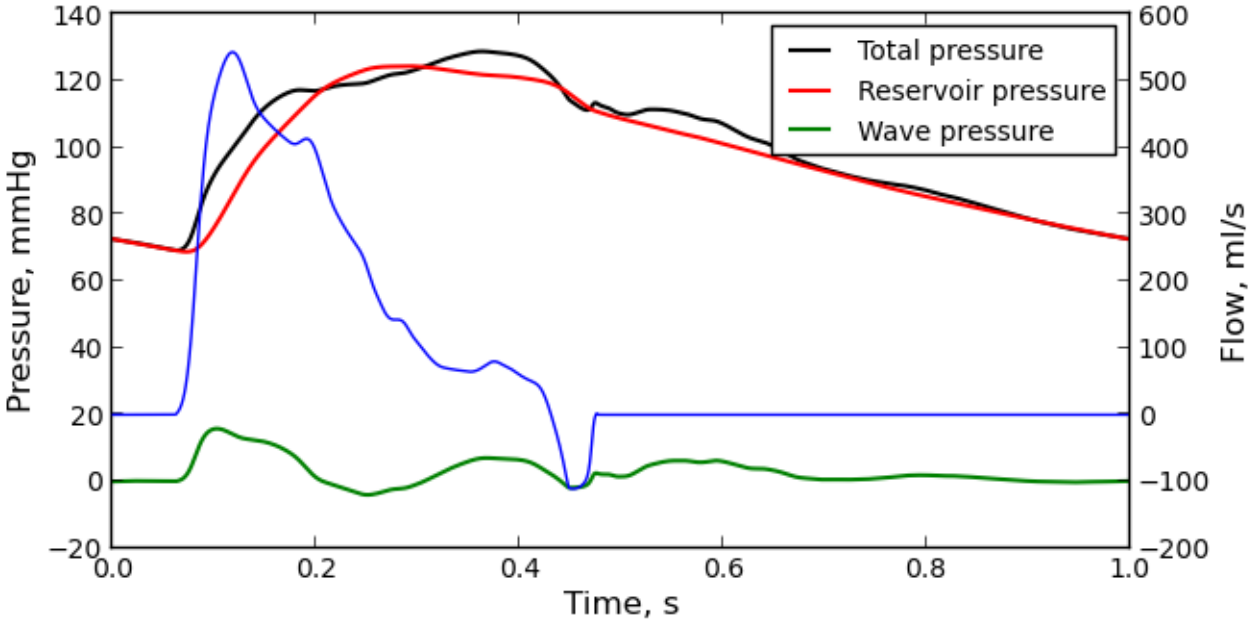


Figure 35: Pressure from the network simulation has been separated into reservoir and wave pressure.

As can be seen from the figure the shape wave pressure does somewhat resemble the shape of the flow, however not as obviously as in the results presented by Tyberg et al. [19].

8.5 Conclusion

The varying elastance model was successfully implemented with a functioning valve model for the aortic and mitral valve. It was found out that the model was very sensitive to loading conditions and in particular the premature closing of the aortic valve would cause problems. With the right loading conditions and valve parameters the model was found to be quite robust. The model was able function well with both lumped model simulations and in a arterial network. The simulations on a long single vessel were not successful not very successful, this showed that the varying elastance model was dependent on reflections in early systole to function properly, something which both the lumped model and network simulations provide.

The varying elastance model was implemented using a source resistance. It did introduce a load dependence in the model, however it is uncertain how well this worked, the source resistance was also not tested with an appropriate varying elastance function. There is a lot of uncertainty regarding the source resistance, and at this stage in the development it is probably not really necessary to have it.

The varying elastance simulations on the arterial network were successful in producing realistic pressures and flows at the aorta. The details of the wave-propagations throughout the network were not studied, but it was shown that as a whole the arterial network worked very similarly to a three-element windkessel model. This was shown both by the fact that pressure decay in the network followed the exponential windkessel decay very closely and that it worked similarly to the Windkessel model as an afterload on the heart.

The bifurcation algorithm was improved, it was discovered that the non-linear equations were actually quite linear and that one iteration per increment was enough to produce good results. The new algorithm gives a significant reduction in computation time.

An alternative discretization scheme to the WK2 and WK3 models was provided. Numerically the new schemes gave a slight increase in accuracy. Also the peripheral pressure was included in the schemes thus allowing pressure decay from an initial pressure

8.6 Further work

There is a lot of uncertainty about the exact values of the valve parameters. Particularly the closing time of the aortic valve has a big influence on the flow and thus the waveforms produced by the varying elastance model. A study should therefore be done on the estimation of valve parameters and perhaps also on the validity of valve the model itself.

Since a the varying elastance model is up and running with both aortic and mitral flow being simulated, the next logical step is to expand the model to include a varying elastance left atrium and have the model work as a connection between two 1D vessel segments or perhaps even in a closed loop.

References

[1] Grégoire Blaudszun and Denis R Morel. Relevance of the volume-axis intercept, v0, compared with the slope of end-systolic pressure–volume relationship in response to large variations in inotropy and afterload in rats. *Experimental Physiology*, 96(11):1179–1195, 2011.

[2] Tom E Claessens, Dimitrios Georgakopoulos, Marina Afanasyeva, Sebastian J Vermeersch, Huntly D Millar, Nikos Stergiopulos, Nico Westerhof, Pascal R Verdonck, and Patrick Segers. Nonlinear isochrones in murine left ventricular pressure-volume loops: how well does the time-varying elastance concept hold? *American Journal of Physiology-Heart and Circulatory Physiology*, 290(4):H1474–H1483, 2006.

[3] Vinzenz Gregor Eck. Arterial flow and pulse wave propagation in one dimensional arterial networks with statistically distributed model parameters. Master's thesis, Norwegian University of Science and Technology, 2012.

[4] Damien Garcia, Lyes Kadem, David Savéry, Philippe Pibarot, and Louis-Gilles Durand. Analytical modeling of the instantaneous maximal transvalvular pressure gradient in aortic stenosis. *Journal of biomechanics*, 39(16):3036–3044, 2006.

[5] Damien Garcia, Philippe Pibarot, and Louis-Gilles Durand. Analytical modeling of the instantaneous pressure gradient across the aortic valve. *Journal of biomechanics*, 38(6):1303–1311, 2005.

[6] Leif Rune Hellevik. Cardiovascular biomechanics compendium.

[7] D. J. Penny J. P. Mynard, M. R. Daavidson and J. J. Smolich. A simple, versatile model for use in lumped parameter and one-dimensional cardiovascular models. *Int. J. Numer. Meth. Biomed. Engng.*, 2011.

[8] PAUL STEENDIJK THEO J. C. FAES BEREND E. WESTERHOF TACO KIND ANTON VONK-NOORDEGRAAF JAN-WILLEM LANKHAAR, FLEUR A. ROVEKAMP and NICO WESTERHOF. Modeling the instantaneous pressure–volume relation of the left ventricle: A comparison of six models. *Annals of Biomedical Engineering*, 2009.

[9] Jan P. Mulier Joseph L. Palladino and Abraham Noordergraaf. Defining ventricular elastance. In *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*.

[10] Paul Roger Leinan. *Biomechanical of fetal veins: The umbilical vein and ductus venosus bifurcation*. PhD thesis, Norwegian University of Science and Technology, 2012.

[11] Louis J. Dell'Italia John Mancini John C. Lasher Jack L. Lancaster Mark R. Starling, Richard A. Walsh. The relationship of various measures of end-systole to left ventricular maximum time-varying elastance in man. *Circulation*, 1978.

[12] Gary F Mitchell, Helen Parise, Emelia J Benjamin, Martin G Larson, Michelle J Keyes, Joseph A Vita, Ramachandran S Vasan, and Daniel Levy. Changes in arterial stiffness and wave reflection with advancing age in healthy men and women the framingham heart study. *Hypertension*, 43(6):1239–1245, 2004.

[13] Jean-Jacques Meister Nikos Stergiopulos and Nico Westerhof. Determinants of stroke volume and systolic and diastolic aortic pressure. 1996.

[14] Joseph S. Janicki Sanjeev G. Shroff and Karl T. Weber. Evidence and quantification of left ventricular systolic resistance. *American Journal of Physiology - Heart and Circulatory Physiology*, 1985.

[15] N Stergiopulos, DF Young, and TR Rogge. Computer simulation of arterial flow with applications to arterial and aortic stenoses. *Journal of biomechanics*, 25(12):1477–1488, 1992.

[16] HIROYUKI SUGA and KIICHI SAGAWA. Instantaneous pressure-volume relationships and their ratio in the excised, supported canine left ventricle. *Circulation Research*, 1974.

[17] Y. Sun, B. J. Sjoberg, P. Ask, D. Loyd, and B. Wranne. Mathematical model that characterizes transmitral and pulmonary venous flow velocity patterns. *American Journal of Physiology - Heart and Circulatory Physiology*, 268(1):H476–H489, 1995.

[18] Theo J. C. Faes Jan-Willem Lankhaar Paul Steendijk Taco Kind, Nico Westerhof and Anton Vonk-Noordegraaf. Cardiac phase-dependent time normalization reduces load dependence of time-varying elastance. *Am J Physiol Heart Circ Physiol*, 2009.

[19] John V Tyberg, Justin E Davies, Zhibin Wang, William A Whitelaw, Jacqueline A Flewitt, Nigel G Shrive, Darryl P Francis, Alun D Hughes, Kim H Parker, and Jiun-Jr Wang. Wave intensity analysis and the development of the reservoir–wave approach. *Medical & biological engineering & computing*, 47(2):221–232, 2009.

[20] ET Van der Velde, D Burkhoff, P Steendijk, J Karsdon, K Sagawa, and J Baan. Nonlinearity and load sensitivity of end-systolic pressure-volume relation of canine left ventricle in vivo. *Circulation*, 83(1):315–327, 1991.

[21] Nico Westerhof, Jan-Willem Lankhaar, and Berend E Westerhof. The arterial windkessel. *Medical & biological engineering & computing*, 47(2):131–141, 2009.

A Varying elastance - implemented code

```
class VaryingElastance(BoundaryConditionType2):

    def __init__(self):
            self.type = 2

            self.subiterations = 0
            #Default parameters
            self.T = 1
            self.Emax  = 2.31 * 133.3e6
            self.Emin = 0.06 * 133.3e6
            self.Tpeak = 0.4
            self.V0 = 20e-6
            self.K  =  1.5e3

            self.alpha = 1.672
            self.n1 = 1.32
            self.n2 = 21.9

            #n-1 values
```

```
        #self.aorticPressurePreviousTimestep
        #self.aorticFlowPreviousTimestep

        self.system = {'both_open':np.array([0,1,2]), 'mitral
            _open': np.array([0,1]), 'aortic_open':np.array
            ([1,2])}


        self.cycleNumber = 0
        self.venousPressure = 7.5 * 133.32

        self.x0 = np.array([0.0, 0.0, 0.0])

        self.initializeValves()

    def initializeSolutionVectors(self, Tsteps):
        self.mitral.initializeSolutions(Tsteps)
        self.aortic.initializeSolutions(Tsteps)

        self.pressure = np.zeros(Tsteps)
        self.volume = np.zeros(Tsteps)
        self.mitralQ = np.zeros(Tsteps)
        self.pressure[0] = self.venousPressure
        self.volume[0] = self.venousPressure/self.E(0) + self
            .V0

    def initializeValves(self):
        #Mitral valve parameters
        self.mitral_annulus_area = 0.0007
        mitral_M_st = 0.7
        mitral_M_rg = 0.0
        mitral_delta_p_open = 0
        mitral_delta_p_close = 0
        mitral_K_v_open = 0.3
        mitral_K_v_close = 0.4
        mitral_l_eff = 0

        #Aortic valve parameters
        #aortic_annulus_area = 0 #Updated values are used in
            stead
        aortic_M_st = 0.99
        aortic_M_rg = 0.00
        aortic_delta_p_open = 0*133.32
        aortic_delta_p_close = 2.0*133.32
        aortic_K_v_open = 0.12
        aortic_K_v_close = 0.12
        aortic_l_eff = 0.00

        #Create valves
        self.mitral = Valve(mitral_M_st, mitral_M_rg,
            mitral_delta_p_open, \
```

```
                                                mitral_delta_p_close,
                                                    mitral_K_v_open,
                                                mitral_K_v_close,
                                                1060, mitral_l_eff
                                                )
        self.aortic = Valve(aortic_M_st, aortic_M_rg,
            aortic_delta_p_open, \
                                                aortic_delta_p_close,
                                                    aortic_K_v_open,
                                                aortic_K_v_close,
                                                1060, aortic_l_eff
                                                )

def __call__(self,_domega_,dO,du,R,L, n,dt, P,bP, Q,bQ, A):

        self.updateValves(P,A,Q, bQ, n, dt)
                                                # Update the
            state of the mitral and aortic valve at timestep n
             + 1
        self.startNewCycleIfCriteriaIsMet(n, dt)
        #self.computeMitralFlow(n, dt)
                                                # Compute
            the flow through the mitral valve at timestep n+1
            explicitly

        domega = self.returnFunction(_domega_, dO, du, R, n,
            dt, P,bP, Q,bQ, A)      # Compute the riemann
            variant going into the vessel

        #self.computeVolume(n, dt, Q, domega[0], domega[1], R
            )  # Compute the volume of the ventricle at
            timestep n+1
        #self.computePressure(n, dt, Q, A, np.dot(R[1,:],
            domega))                                    #
            Compute the pressure in the ventricle at timestep
            n+1
        return domega

def updateValves(self, P, A,Q, Qn, n, dt):
        mitralPressureDifference = self.venousPressure - self
            .pressure[n]

        aorticPressureDifference  = self.pressure[n] - P# -
            1060*0.02/A*(Q - Qn)/dt
        #print mitralPressureDifference
        self.mitral.updateValveState(mitralPressureDifference
            , n, dt)
        #print self.aortic.state[n]
        self.aortic.updateValveState(aorticPressureDifference
            , n, dt)
```

```
def getCycleTime(self, n, dt):
        return n*dt - self.T*self.cycleNumber

def startNewCycleIfCriteriaIsMet(self, n, dt):
        if self.getCycleTime(n+1, dt) > self.T:
                self.cycleNumber += 1

def funcPos0(self, _domega, dO, du, R, n, dt, Pn,Pn1, Qn, Qn1
    , A):

        r21,r22 = R[1][0],R[1][1]
        #L = self.aortic.computeL(A, n+1)
        LdivB = self.aortic.LdivideB(A, n+1)
        B = self.aortic.computeB(A, n+1)
        #mitrL = self.mitral.computeL(self.
            mitral_annulus_area, n+1)
        mitrLdivB = self.mitral.LdivideB(A, n+1)
        mitrB = self.mitral.computeB(self.mitral_annulus_area
            , n+1)
        mitrQn = self.mitralQ[n]
        mitrQn1 = self.mitralQ[n-1]
        venoP = self.venousPressure
        t = self.getCycleTime(n+1, dt)
        E = self.E(t)
        Vn = self.volume[n]
        ventrPn = self.pressure[n]

        B_ref = 1060/(2*A**2)

        n_p = self.Emax*self.V0
        n_q = (n_p/B_ref)**0.5
        n_o = n_q/r21

        #n_q = (mitrQn - mitrQn1)*100
        #if n_q == 0.0: n_q = 1e-3

        #n_p = 0.5*(Pn - Pn1 + ventrPn -self.pressure[n-1])
            *100
        #if n_p == 0.0: n_p = 1e3

        #n_q = 1e-3
        #n_p = 1e3

        #print n_q, n_p

        args = dt, mitrLdivB, mitrB,LdivB,B, mitrQn1, mitrQn,
            ventrPn, venoP, E, Vn, Qn, Qn1, r21, r22, Pn,
            _domega, n_q, n_p, B_ref

        #B_limit = 1e20
```

```
if not B and not mitrB:#== np.inf:
        #print 'both closed'
        self.mitralQ[n+1] = 0
        domega_ = _domega
        self.volume[n+1] = self.volume[n] - 0.5*(Qn -
            mitrQn)*dt
        self.pressure[n+1] = E*(self.volume[n+1] -
            self.V0)
        if Qn == 0:
                domega_ = _domega#return [_domega,
                    _domega]
        else:
                domega_ = (-0.5*Qn - r22*_domega)/r21
        self.x0 = np.array([0,0,domega_])
else:
        if not mitrB:
                #print 'aortic open'
                x = self.newtonSolver(self.x0,args,
                    partialSystem='aortic_open')
                #print x[0]*n_p, x[1]*n_p
                self.mitralQ[n+1] = 0
                self.pressure[n+1] = self.pressure[n]
                    + x[0]*n_p
                domega_  = x[1]*n_o
                self.x0 = np.concatenate((np.array
                    ([0]), x))
        elif not B:
                #print 'mitral open'
                x = self.newtonSolver(self.x0,args,
                    partialSystem='mitral_open')
                #print x
                self.mitralQ[n+1] = self.mitralQ[n] +
                    x[0]*n_q
                self.pressure[n+1] = self.pressure[n]
                    + x[1]*n_p
                if Qn == 0:
                        domega_ = _domega
                else:
                        domega_ = (-0.5*Qn - r22*
                            _domega)/r21
                self.x0 = np.concatenate((x, np.array
                    ([0])))
        else:
                #print 'both open'
                x = self.newtonSolver(self.x0,args)
                self.mitralQ[n+1] = self.mitralQ[n] +
                    x[0]*n_q
                self.pressure[n+1] = self.pressure[n]
                    + x[1]*n_p
                domega_ = x[2]*n_o
```

```
                    self.x0 = x
                dQ = r21*domega_ + r22*_domega
                self.volume[n+1] = Vn - (Qn + 0.5*(-self.
                    mitralQ[n] - self.mitralQ[n+1] + dQ))*dt


        return [domega_, _domega]


def funcPos1(self, _domega, dO, du, R, L, n, dt, P, Q, A):
        pass

def E(self, t):

        alpha1 = 0.708*self.Tpeak/self.T
        alpha2 = 1.677*alpha1
        T, n1, n2 = self.T, self.n1, self.n2
        shapeFunction1 = (t/(alpha1*T))**n1/(1+(t/(alpha1*T))
            **n1)
        shapeFunction2 = (1 + (t/(alpha2*T))**n2)**(-1)
        return (self.Emax-self.Emin)*self.alpha*
            shapeFunction1*shapeFunction2 + self.Emin

def newtonSolver(self, x0, args,  partialSystem = 'both_open'
    ):

        #dt, mitrL, mitrB,L,B, mitrQn1, mitrQn, venoP, E, Vn,
            Qn, Qn1, r21, r22, Pn, _domega = args


        #print partialSystem
        iterations = 0
        #print np.size(args)
        xn = x0[self.system[partialSystem]]
        res = self.solverResiduals(xn,*args, partialSystem =
            partialSystem)
        #print partialSystem
        error = np.linalg.norm(res, 2)

        #print x0[1]
        #print res
        while True:#iterations <3:#
                #print x0
                iterations +=1
                J_inv = self.solverInverseJacobian(xn, *args,
                    partialSystem = partialSystem)
                x = xn - np.dot(J_inv, res)
                error = np.linalg.norm(x - xn, 2)/np.linalg.
                    norm(xn, 2)
                if error < 0.0001:
                        break
```

```
                xn = x
                res = self.solverResiduals(x, *args,
                    partialSystem = partialSystem)

                #J_inv = self.solverInverseJacobian(x0, *args
                    , partialSystem = partialSystem)
                if iterations > 20:
                        x *= 0#x0[self.system[partialSystem]]
                        break
        #print args[11]
        #print x0[1]
        print iterations
        #print error

        return x

    def solverResiduals(self, x_partial, dt, mitrLdivB, mitrB,
        LdivB,B, mitrQn1, mitrQn,ventrPn, atrP, E, Vn, Qn, Qn1,
        r21, r22, Pn, _domega, n_q, n_p, B_ref, partialSystem = np
        .array([0,1,2])):#dt, mitrL, mitrB,L,B, mitrQn1, mitrQn,
        venoP, E, Vn, Qn, Qn1, r21, r22, Pn, _domega):
            # n -> values at timestep n ; n1 -> values at
                timestep n-1; rest is at timestep n+1
            #dt, mitrL, mitrB,L,B, mitrQn1, mitrQn, venoP, E, Vn,
                 Qn, Qn1, r21, r22, Pn, _domega = args

            x = np.array([0.0, 0.0, 0.0])
            x[self.system[partialSystem]] += x_partial
            dQm, dPv, domega_ = x

            def f1():
                    a = mitrQn/n_q + dQm
                    return a*abs(a) + mitrLdivB/(2*n_q*dt)*(3*dQm
                        + (mitrQn1 - mitrQn)/n_q) + (n_p*dPv +
                        ventrPn - atrP)/(mitrB*n_q**2)
            def f2(): return E/n_p*(Vn - (Qn - mitrQn + 0.5*(n_q*
                domega_ + r22*_domega - n_q*dQm))*dt - self.V0)
                *(1-self.K*(Qn + n_q*domega_ + r22*_domega)) -
                ventrPn/n_p - dPv
            def f3():
                    a = (Qn + r22*_domega)/n_q + domega_
                    return a*abs(a) + LdivB/(2*n_q*dt)*(3*domega_
                        + (3*r22 - Qn + Qn1)/n_q) + (n_q/r21*
                        domega_ + _domega + Pn - ventrPn - n_p*dPv
                        )/(B*n_q**2)

            functions = np.array([f1, f2, f3])
            return np.array([f() for f in functions[self.system[
                partialSystem]]])
```

```
def solverInverseJacobian(self, x_partial, dt, mitrLdivB,
    mitrB,LdivB,B, mitrQn1, mitrQn,ventrPn, venoP, E, Vn, Qn,
    Qn1, r21, r22, Pn, _domega, n_q, n_p,B_ref, partialSystem
    = np.array([0,1,2])):
        # n -> values at timestep n ; n1 -> values at
            timestep n-1; rest is at timestep n+1
        x = np.array([0, 0, 0])
        x[self.system[partialSystem]] += x_partial
        dmQ, dvP, domega_ = x
        #print n_q, n_p, E, dt, self.K, mitrL, mitrB, mitrQn,
            dmQ,

        expressions = np.array([
        '2*(mitrQn/n_q + dmQ)*np.sign(mitrQn/n_q + dmQ) +
            1.5*mitrLdivB/(n_q*dt)',
        'B_ref/mitrB',
        '0.5*n_q/n_p*E*dt*(1-self.K*(Qn + n_q*domega_ + r22*
            _domega))',#0.5*(1 - self.K*(Qn + r21*domega_ +
            r22*_domega))*E*dt**2/mitrL
        '-0.5*n_q/n_p*E*dt*(1-self.K*(Qn + n_q*domega_ + r22*
            _domega)) - E*self.K*n_q/n_p*(Vn -(Qn - mitrQn +
            0.5*(n_q*domega_ + r22*_domega - n_q*dmQ))*dt -
            self.V0)',
        '-B_ref/B',
        '2*((Qn + r22*_domega)/n_q + domega_)*np.sign((Qn +
            r22*_domega)/n_q + domega_) + 1.5*LdivB/(n_q*dt) +
            1/(r21*B*n_q)'])

        if partialSystem == 'mitral open':
                a1, a2, a3 = [eval(e) for e in expressions
                    [[0,1,2]]]

                J_inv = np.array([[-1, -a2],
                                                    [-a3, a1]])
                                                      /(-a1-a2
                                                      *a3)
                return J_inv
        elif partialSystem == 'aortic open':
                a4, a5,a6 = [eval(e) for e in expressions
                    [[3,4,5]]]
                J_inv =np.array([[a6, -a4],
                                                  [-a5, -1]])
                                                    /(-a6 -
                                                    a4*a5)
                return J_inv
        else:
                a1, a2, a3, a4, a5, a6 = [eval(e) for e in
                    expressions]

                J_inv = np.array([[ a6+a4*a5,  a2*a6,  -a2*a4
```

```
                                ],
                                                    [     a3*a6, -a1
                                                    *a6,    a1*a4
                                                        ],
                                                    [   -a3*a5,   a1
                                                    *a5,    a1+a2
                                                    *a3]])/(a3*
                                                    a2*a6 + a1*
                                                    a6 + a1*a4*
                                                    a5)


                        return J_inv


class Valve:
        def __init__(self, M_st, M_rg, delta_p_open, delta_p_close,
           K_v_open, K_v_close, rho, l_eff):
                self.M_st = M_st
                self.M_rg = M_rg
                self.delta_p_open = delta_p_open
                self.delta_p_close = delta_p_close
                self.K_v_open = K_v_open
                self.K_v_close = K_v_close

                self.l_eff = l_eff

                self.rho = 1060

        def initializeSolutions(self, Tsteps):
                self.A_s = np.zeros(Tsteps)
                #self.L = np.zeros(Tsteps)
                self.state = np.zeros(Tsteps)

        def computeB(self, A, n):
                A_s = self.effectiveOrificeArea(A,n)#self.A_s[n]
                if A_s == 0:
                        B = None
                elif A/A_s > 1e4:
                        B = None
                else:
                        B = 0.5*self.rho*(1/A_s - 1/A)**2
                return B

        def computeL(self, A, n):
                A_s = self.effectiveOrificeArea(A,n)#self.A_s[n]
                if A_s == 0:
                        L = None
                elif A/A_s > 1e4:
                        L = None
                else:
```

```
                L = 2*np.pi*self.rho*(1/A_s - 1/A)**0.5# +
                    self.rho*self.l_eff/A
        return L

#def A_eff(self, A, state):
#       A_s = self.A_s(A, state)
#       return A*A_s/(A-A_s)

def LdivideB(self, A, n):
        A_s = self.effectiveOrificeArea(A,n)# self.A_s[n]
        #self.A_s[n] = A_s
        if A_s == 0:
                return None
        elif A/A_s > 1e4:
                return None
        else:
                return 4*np.pi*(1/A_s - 1/A)**(-1.5)# + 2*
                    self.l_eff*A*A_s**2/(A - A_s)**2

def effectiveOrificeArea(self, A, n):
        return (self.A_max(A) - self.A_min(A)) * self.state[n
            ] + self.A_min(A)

def A_max(self, A):
        return self.M_st * A

def A_min(self, A):
        return self.M_rg * A

def updateValveState(self, delta_p, n, dt):

        if delta_p > self.delta_p_open:
                if self.state[n] == 1.0:
                        self.state[n+1] = 1.0
                else:
                        #self.state[n+1] = self.state[n-1] +
                            2*dt*(1-self.state[n])*self.
                            K_v_open*(delta_p - self.
                            delta_p_open)
                        self.state[n+1] = self.state[n] + (1
                            - self.state[n])*self.K_v_open*(
                            delta_p - self.delta_p_open)*dt
                        if self.state[n+1] > 1.0:
                                self.state[n+1] = 1.0
        elif delta_p < -self.delta_p_close:
                if self.state[n] == 0.0:
                        self.state[n+1] = 0.0
                else:
                        #self.state[n+1] = self.state[n-1] +
                            2*dt*self.state[n]*self.K_v_open*(
                            delta_p - self.delta_p_close)
```

```
                    self.state[n+1] = self.state[n] +
                        self.state[n]*self.K_v_close*(
                        delta_p - self.delta_p_close)*dt
                    if self.state[n+1] < 0:
                            self.state[n+1] = 0.0
        else:
                self.state[n+1] = self.state[n]
```

B Bifurcations - implemented code

```
def call1(self, P, Q, A, dt):#(self, P,Pn, Q, Qn, A, An, dt):

    P1, P2, P3, Q1, Q2, Q3, A1, A2, A3 = P[0][-1], P[1][0], P
        [2][0], Q[0][-1], Q[1][0], Q[2][0], A[0][-1], A[1][0], A
        [2][0]
    C1, C2, C3 = self.mothers[0].C_nID(P[0], -1), self.daughters
        [0].C_nID(P[1], 0), self.daughters[1].C_nID(P[2], 0)
    #rho1, rho2, rho3 = self.rho[0], self.rho[1], self.rho[2]
    rho = self.rho[0]

    r221, r212, r213 = self.systemEquations[0].R[-1][1][1], self.
        systemEquations[1].R[0][1][0], self.systemEquations[2].R
        [0][1][0]
    r211, r222, r223 = self.systemEquations[0].R[-1][1][0], self.
        systemEquations[1].R[0][1][1], self.systemEquations[2].R
        [0][1][1]
    L1, L2, L3 = self.systemEquations[0].L[-1][0], self.
        systemEquations[1].L[0][1], self.systemEquations[2].L
        [0][1],


    """Calculate domega_1"""
    z1 = self.z[0][-1] + self.vz[0]  * self.c_func[0](A[0],P
        [0],-1) * dt
    du1 = np.array([np.interp(z1,self.z[0],P[0]) - P1 ,np.interp(
        z1,self.z[0],Q[0]) - Q1])
    #du1 = np.array([np.interp(z1,self.z[0],P[0]) - np.interp(z1,
        self.z[0],Pn[0]),np.interp(z1,self.z[0],Q[0]) - np.interp(
        z1,self.z[0],Qn[0])])
    domega_1 =  np.dot(L1,du1)


    """Calculate _domega2 """
    z2 = self.z[1][0] + self.vz[1]  * self.c_func[1](A[1],P[1],0)
         * dt
    du2 = np.array([np.interp(z2,self.z[1],P[1])-P2 ,np.interp(z2
        ,self.z[1],Q[1])-Q2])
    _domega2 =  np.dot(L2,du2)


    """Calculate _domega2 """
```

```
z3 = self.z[2][0] + self.vz[2]  * self.c_func[2](A[2],P[2],0)
    * dt
du3 = np.array([np.interp(z3,self.z[2],P[2])-P3,np.interp(z3,
   self.z[2],Q[2])-Q3])
_domega3 =  np.dot(L3,du3)


sol = self.newtonSolver(self.domega_previous_timestep, P1, P2
   , P3, Q1, Q2, Q3, A1, A2, A3, C1, C2, C3, rho,\
                                domega_1, _domega2, _domega3,
                                   r221, r212, r213, r211, r222,
                                    r223)
self.domega_previous_timestep = sol[0:2]
_domega1, domega_2, domega_3 = sol


"""
args = [P1, P2, P3, Q1, Q2, Q3, A1, A2, A3, C1, C2, C3, rho,\
                           r221, r212, r213,r211, r222, r223,
                              domega_1, _domega2, _domega3]
error = sum([abs(i) for i in self.fsolvefunction(self.
   domega_previous_timestep,args)])

if error < 1.E-2:
    return P1, Q1, A1, P2, Q2, A2, P3, Q3, A3


sol,infodict,a,b = fsolve(self.fsolvefunction, self.
   domega_previous_timestep, args=args, fprime = self.
   jacobianInverseMatrix, full_output = True, xtol = 10.0)
self.subiterations += infodict['nfev']
print self.subiterations

domega_2, domega_3 = sol
_domega1 = (r212*domega_2 + r222*_domega2 + r213*domega_3 +
   r223*_domega3 - r211*domega_1)/r221
"""


domega1 = np.array([domega_1, _domega1])
domega2 = np.array([domega_2, _domega2])
domega3 = np.array([domega_3, _domega3])

R1, R2, R3 = self.systemEquations[0].R[-1], self.
   systemEquations[1].R[0], self.systemEquations[2].R[0]

du1 = np.dot(R1, domega1)
du2 = np.dot(R2, domega2)
du3 = np.dot(R3, domega3)

P1 += du1[0]
```

```
    Q1 += du1[1]
    A1 = self.A_func[0]([P1],-1)

    P2 += du2[0]
    Q2 += du2[1]
    A2 = self.A_func[1]([P2],0)

    P3 += du3[0]
    Q3 += du3[1]
    A3 = self.A_func[2]([P3],0)

    return P1, Q1, A1, P2, Q2, A2, P3, Q3, A3

def newtonSolver(self,x0, P1, P2, P3, Q1, Q2, Q3, A1, A2, A3, C1,
    C2, C3, rho, domega_1, _domega2, _domega3, r221, r212, r213,
   r211, r222, r223):
    args = [P1, P2, P3, Q1, Q2, Q3, A1, A2, A3, C1, C2, C3, rho,\
                            r221, r212, r213,r211, r222, r223,
                               domega_1, _domega2, _domega3]
    res = self.residualFunctionNewAlgorithm(x0, args)
    iterations = 0

    if np.linalg.norm(res,2)< 1:#< 0.1:#
        x = x0
    else:
        xn = x0
        J_inv = self.jacobianInverseMatrixNewAlgorithm(xn, args)
        while True:
            iterations +=1
            x = xn - np.dot(J_inv, res)
            #error = (np.linalg.norm(x - xn, 2))/np.linalg.norm(
               xn,2)
            xn = x
            if iterations >0:#error< 0.001:#
                break
            res = self.residualFunctionNewAlgorithm(x, args)

    domega_2, domega_3 = x
    _domega1 =   (r212*domega_2 + r222*_domega2 + r213*domega_3 +
        r223*_domega3 - r211*domega_1 )/r221
    return _domega1, domega_2, domega_3


def residualFunctionNewAlgorithm(self, x, args):
    domega_2, domega_3 = x

    P1, P2, P3, Q1, Q2, Q3, A1, A2, A3, C1, C2, C3, rho,\
                            r221, r212, r213,r211, r222, r223,
                               domega_1, _domega2, _domega3 =
                               args
```

```
        _domega1 = (r212*domega_2 + r222*_domega2 + r213*domega_3 +
            r223*_domega3 - r211*domega_1)/r221

        a = P1 + domega_1 + _domega1 + 0.5*rho*((Q1 + r211*domega_1 +
            r221*_domega1)**2/(A1 + C1*(domega_1 + _domega1))**2)
        b = P2 + domega_2 + _domega2 + 0.5*rho*((Q2 + r212*domega_2 +
            r222*_domega2)**2/(A2 + C2*(domega_2 + _domega2))**2)
        c = P3 + domega_3 + _domega3 + 0.5*rho*((Q3 + r213*domega_3 +
            r223*_domega3)**2/(A3 + C3*(domega_3 + _domega3))**2)

        f1 = a - b
        f2 = a - c

        #print P1, P2, P3, Q1, Q2, Q3, A1, A2, A3, C1, C2, C3, rho,
            r221, r212, r213,r211, r222, r223, domega_1, _domega2,
            _domega3, _domega1, domega_2, domega_3


        return [f1, f2]

    def jacobianInverseMatrixNewAlgorithm(self, x, args):

        domega_2, domega_3 = x
        P1, P2, P3, Q1, Q2, Q3, A1, A2, A3, C1, C2, C3, rho,\
                                r221, r212, r213,r211, r222, r223,
                                    domega_1, _domega2, _domega3 =
                                    args
        _domega1 = (r212*domega_2 + r222*_domega2 + r213*domega_3 +
            r223*_domega3 - r211*domega_1)/r221

        h1 = Q1 + r211*domega_1 + r221*_domega1
        h2 = A1 + C1*(domega_1 + _domega1)
        h3 = Q2 + r212*domega_2 + r222*_domega2
        h4 = A2 + C2*(domega_2 + _domega2)
        h5 = Q3 + r213*domega_2 + r223*_domega3
        h6 = A3 + C3*(domega_3 + _domega3)

        J11 = r212/r221 - 1 + rho*h1*(r212*h2 - C1*r212/r221*h1)/h2
            **3 - rho*h3*(r212*h4 - C2*h3)/h4**3
        J12 = r213/r221 + rho*h1*(r213*h2 - C1*r213/r221*h1)/h2**3
        J21 = r212/r221 + rho*h1*(r212*h2 - C1*r212/r221*h1)/h2**3
        J22 = r213/r221 - 1 + rho*h1*(r213*h2 - C1*r213/r221*h1)/h2
            **3 - rho*h5*(r213*h5 - C3*h6)/h6**3

        det = J11*J22-J12*J21

        J_inv = 1/det*np.array([[ J22,-J12],
                                [-J21, J11]])
        return J_inv

    def jacobianMatrixNewAlgorithm(self, x, args):
```

```
domega_2, domega_3 = x
P1, P2, P3, Q1, Q2, Q3, A1, A2, A3, C1, C2, C3, rho,\
                        r221, r212, r213,r211, r222, r223,
                           domega_1, _domega2, _domega3 =
                           args
_domega1 = (r212*domega_2 + r222*_domega2 + r213*domega_3 +
   r223*_domega3 - r211*domega_1)/r221

h1 = Q1 + r211*domega_1 + r221*_domega1
h2 = A1 + C1*(domega_1 + _domega1)
h3 = Q2 + r212*domega_2 + r222*_domega2
h4 = A2 + C2*(domega_2 + _domega2)
h5 = Q3 + r213*domega_2 + r223*_domega3
h6 = A3 + C3*(domega_3 + _domega3)

J11 = r212/r221 - 1 + rho*h1*(r212*h2 - C1*r212/r221*h1)/h2
   **3 - rho*h3*(r212*h4 - C2*h3)/h4**3
J12 = r213/r221 + rho*h1*(r213*h2 - C1*r213/r221*h1)/h2**3
J21 = r212/r221 + rho*h1*(r212*h2 - C1*r212/r221*h1)/h2**3
J22 = r213/r221 - 1 + rho*h1*(r213*h2 - C1*r213/r221*h1)/h2
   **3 - rho*h5*(r213*h5 - C3*h6)/h6**3

#det = J11*J22-J12*J21

J = np.array([[ J11,J12],
              [J21, J22]])
return J
```