SIMULATING THE VISCOELASTIC RESPONSE OF THE SPINAL CORD

by

NINA KRISTINE KYLSTAD

THESIS

for the degree of

MASTER OF SCIENCE

(Master i Anvendt matematikk og mekanikk)

Faculty of Mathematics and Natural Sciences
University of Oslo

June 2014

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Acknowledgments

First and foremost I would like to thank my supervisors, Marie Rognes and Kent-André Mardal. Your advice and encouragement have helped me a great deal in the process of writing this thesis. I am truly grateful for your guidance and support during this time.

I would like to express my gratitude to friends and acquaintances both at the University of Oslo and at Simula, for many interesting, encouraging and fruitful conversations. A special thanks goes out to Karen-Helene Støverud, who provided me with code for the pressure data, and meshes, and was always available to answer any questions I had. Thank you!

My friends and family have been loving and supportive of me throughout this process. You have reminded me that there is a world outside the study hall, and you have kept my spirits up. For that I am very grateful.

Last, but not least, my wonderful Fredrik: Thank you for everything, you are the best.

The simulations in this thesis were performed on the Abel Cluster, owned by the University of Oslo and the Norwegian metacenter for High Performance Computing (NOTUR), and operated by the Department for Research Computing at USIT, the University of Oslo IT-department.
`http://www.hpc.uio.no/`
This research is supported by the Research Council of Norway via NOTUR allocation NN9316K.

Contents

List of abbreviations

BC	Boundary condition
CSF	Cerebrospinal fluid
KV	Kelvin-Voigt
PDE	Partial differential equation
SAS	Subarachnoid space
SLS	Standard Linear Solid

List of Figures

List of Tables

Chapter 1

Introduction

The Chiari I malformation is a condition characterized by the downward displacement of the hindbrain, putting pressure on the spinal cord and potentially blocking the flow of cerebrospinal fluid (CSF) from the brain to the spinal column. Chiari I is relatively common in that approximately 1% of the adult population is believed to have such a malformation. However, not everyone develop symptoms.

Symptomatic Chiari I malformation is often associated with syringomyelia, a condition in which fluid-filled cysts begin to form in the spinal cord. In fact, up to 70% of syringomyelia is related to hindbrain disorders [22], Chiari I being the most common of these [29]. Even though there appears to be a clear connection between the Chiari I malformation and syringomyelia, the reason for it is not well understood. Due to the difficulty in studying spinal cord conditions without the use of invasive procedures, it is thought that numerical simulations may aid in the understanding the connection.

There are several theories as to why syringomyelia develops; many of these are based in mechanics, and it is believed that the obstructed CSF flow may be of importance. *In vivo* measurements of CSF velocity by for example Quigley et al. [39], Haughton et al. [23] and Dolar et al. [18] support the theory that the Chiari I malformation is associated with abnormal CSF velocities.

Computational studies by Hentschel et al. [24], Støverud et al. [49] and Roldan et al. [40] have also investigated CSF flow in the SAS, the two former using idealized geometries and the latter using patient-specific geometries, showing that the presence of an obstruction in the subarachnoid space (SAS) causes abnormal CSF flow. Computational studies by Bertram et al. [8] and Bertram [7] model the fluid/structure interactions in the spinal column, using linear elastic models for the spinal cord. The models in these studies are simplified 2D geometries. To the author's knowledge, several studies have used patient-specific geometries when modelling CSF flow, but few have used such geometries to model the spinal cord

itself. However, a study by Støverud et al. [50] does model the spinal cord as a poro-elastic medium using an anatomically accurate spinal cord geometry.

Many studies have been conducted to investigate the properties of the spinal cord. In particular, the viscoelastic properties of the spinal cord have been extensively tested. A review of experiments on the rheological properties of the spinal cord, both invasive and using magnetic resonance elastography (MRE) is given by Cheng et al. [12]. To the author's knowledge however, there are no studies that have combined the use of a viscoelastic model for the spinal cord with patient-specific geometries under pressure conditions from abnormal CSF flow caused by the Chiari I malformation. This motivates an investigation into the effect of using a viscoelastic model together with an anatomically accurate spinal cord geometry.

This thesis aims at attacking the problem using mathematical modelling. Specifically, the spinal cord is modelled as an elastic solid. A linear elasticity model is used as a reference material, while several options for a linear viscoelasticity model are explored. The mathematical models are combined with the framework of the finite element method in order to solve the problem computationally. This allows for simulations of the response of the spinal cord under pressure.

1.1 Outline of the thesis

In *Chapter 2*, a brief introduction to the medical background of the problem is provided.

In *Chapter 3*, the governing equations for elasticity is presented, followed by an overview of how the equations will be used to simulate the spinal cord under pressure. Constitutive relationships for linear elasticity and linear viscoelasticity are presented, and relevant parameter values are selected from the literature.

In *Chapter 4*, an outline of the numerical methods needed to solve the problems is given; in particular, an overview of the finite element method is provided.

In *Chapter 5*, the numerical methods are applied to the mathematical models. An efficient computational scheme for the viscoelasticity problem is presented. Specific choices for boundary conditions are discussed, and an overview of the computer implementation of the problems is given.

In *Chapter 6*, the results from verification tests of the implementations are shown, followed by the simulation results for the spinal cord.

In *Chapter 7*, a thorough discussion of the results is provided.

In *Chapter 8*, the results are summarized and a conclusion is provided, along with suggestions for further work.

Chapter 2

Medical background

This chapter aims at giving a brief overview of the medical background which provides the fundamental motivation for this thesis.

2.1 The spinal cord

This section addresses the anatomy of the central nervous system, and in particular the spinal cord. The information in this section is compiled from the reference works [36, 20, 14].

The central nervous system consists of the brain and the spinal cord. The spinal cord is continuous with the brain above, and descends down approximately two thirds of the vertebral canal, which lies within the vertebral column. The spinal cord is cylindrical in shape, and is about 40-45 cm long.

Along the spinal cord, 31 pairs of spinal nerves are attached. The spinal nerves are bundles of fibres that conduct nerve impulses from the central nervous system to the muscles in the body. Figure 2.1 shows an illustration of a cross section of a vertebra and the spinal cord.

The spinal cord is made up of grey matter surrounded by white matter. Covering and protecting the spinal cord are three layers of *meninges* (membranes). The outermost is known as the *dura mater*, and is a tough, fibrous membrane. The innermost membrane is known as the *pia mater*, which covers the surface of the spinal cord. Between the pia and the dura is the *arachnoid mater*, a thin translucent membrane. The space between the pia and arachnoid is known as the subarachnoid space (SAS), and contains CSF. This fluid is a clear liquid that occupies the SAS both in the cranium and in the vertebral canal. CSF flows in and out of the cranium as the brain expands and shrinks to accommodate the

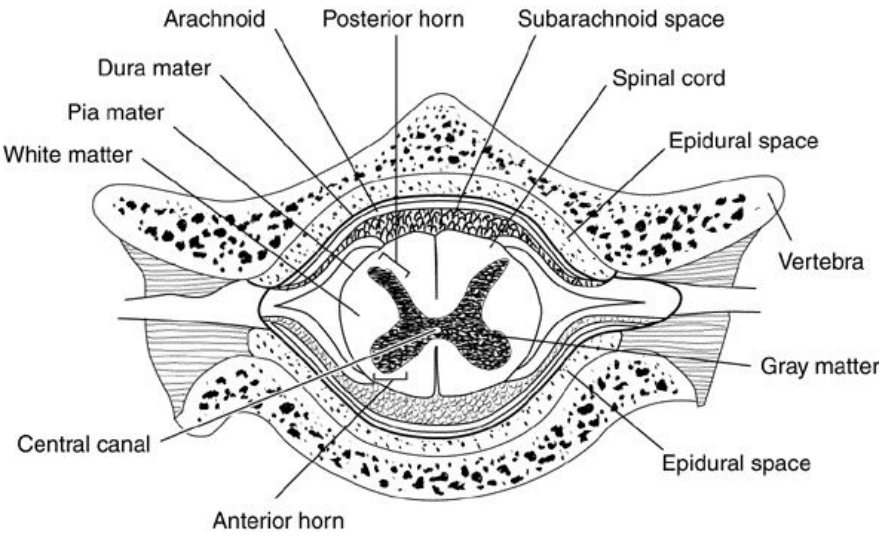


Figure 2.1: Cross section of a vertebra and the spinal cord [2].

blood pulsating in and out of the brain during a cardiac cycle. CSF is produced inside the ventricles of the brain and consists mainly of water with small amounts of glucose, salts and proteins which provide energy for the brain.

The spinal cord connects with the brain via the brain stem at the *foramen magnum*, which is the opening in the base of the skull through which the spinal cord enters the cranium. Above the foramen magnum lies the cerebellum, see Figure 2.2.

The cerebellum is the largest part of the hindbrain. On the bottom surface of the cerebellum are the cerebellar tonsils, which are important in the context of the Chiari malformations.

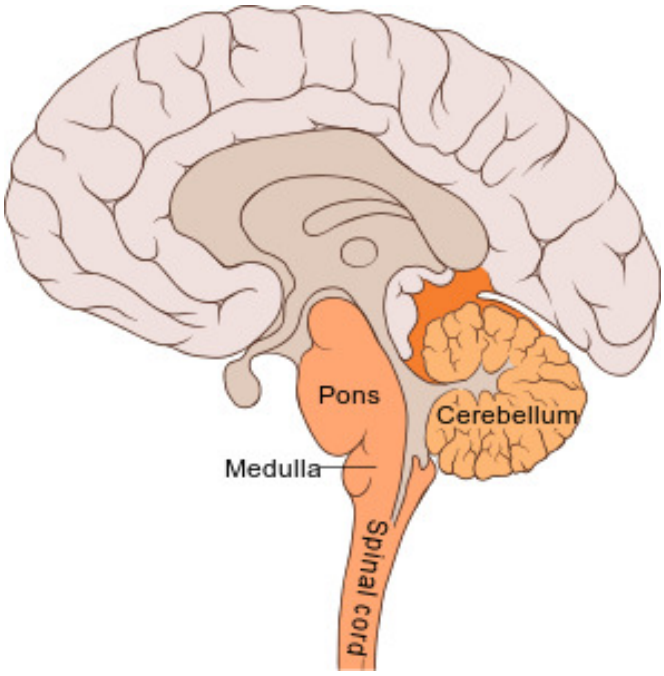


Figure 2.2: Illustration of central nervous system. Illustration created by Patrick J. Lynch, medical illustrator; C. Carl Jaffe, MD, cardiologist.

2.2 Chiari malformation

A Chiari malformation is a malformation of the brain or possibly of the skull. It is characterized by a downward displacement of the cerebellar tonsils through the foramen magnum. This downward displacement is known as a *herniation*, and may may obstruct CSF flow at the foramen magnum, which may cause abnormal CSF velocities and increased pressure gradients.

Chiari Malformations are classified into four main groups, the Chiari I, II, III, and IV malformations, where the scale of severity is I - IV, the last being the most severe and very rare.

The traditional definition of the Chiari I malformation is "a herniation of the cerebellar tonsils past the foramen magnum by $3-5$mm, as diagnosed by magnetic resonance imaging" [43]. However, this definition have been much discussed, since patients with less than 3 - 5 mm herniation may have severe symptoms and people with significant herniation have may be without any symptoms.

Symptoms of Chiari I include head/neck pain, body weakness, numbness and dizzines, although symptoms vary greatly from patient to patient [17], see Table 2.1. It is believed that approximately 1% of normal adults have the Chiari I malformation as defined above, although only about $0.01-0.04\%$ of adults display

symptoms [52]. The reasons for the symptoms are believed to be associated with the flow of CSF in and out of the head. Symptoms may also be caused by associated conditions, such as *syringymyelia*.

Symptom	%
Headache	98
Dizziness	84
Difficulty sleeping	72
Weakness of an upper extremity	69
Neck pain	67
Numbness/tingling of an upper extremity	62
Fatigue	59
Nausea	58
Shortness of breath	57
Blurred vision	57
Tinnitus	56
Difficulty swallowing	54
Weakness of a lower extremity	52

Table 2.1: 13 symptoms were reported by more than 50 % of the 265 participating patients with Chiari I malformation in the study by Diane M. Mueller and John J. Oro' [17].

2.3 Syringomyelia

Syringomyelia is a condition in which fluid-filled cavities, or *syrinxes*, develop in the spinal cord. The development of syrinxes may cause nerve damage which is irreversible. Estimates suggest that 30% - 50% of Chiari I patients have an associated syrinx, and, as mentioned, up to 70% of syringomyelia is related to hindbrain disorders such as the Chiari I malformation.

The reason behind the development of syrinxes in the spinal cord is unclear, but is thought to have something to do with the abnormal CSF flow caused by the Chiari I malformation. Many theories have been developed to explain exactly what causes syrinx formation, see for example the reviews by Shaffer et al. [43] or Elliot et al. [19].

Chapter 3

Mathematical models for elasticity and viscoelasticity

The spinal cord will be considered as a solid throughout this thesis. In order to develop a model for the deformation of the spinal cord under pressure caused by CSF flow, the underlying physics must be considered. Theory from continuum mechanics becomes central here – in particular theory of elasticity and viscoelasticity. In this chapter, governing equations for the spinal cord are presented, and the simulation scenario is outlined. Following this, models for constitutive relationships describing material properties are presented. A simple analysis of the models for viscoelasticity is performed in order to determine their appropriateness. Finally, a review of parameter values from the literature is presented, and relevant parameter values are selected.

The theory in this chapter is compiled from the works by Shaw [44], Larson and Bengzon [30] and Banks et al. [5].

3.1 Governing equations

Consider a body $\mathcal{G}$ of elastic material occupying an open, bounded domain $\Omega \in \mathbb{R}^d$, $d = 1, 2, 3$. The forces acting upon such a body can be categorized as one of two types:

(i) Body force – a force acting on the whole volume, measured in N/m^3. Examples include gravitational forces and electromagnetic forces.

(ii) Surface force – a traction force, acting on the boundary $\partial\Omega$ of Ω, measured in N/m^2. An example is pressure.

Physical balance laws describe how these forces affect the body in question.

The boundary $\partial\Omega$ of Ω is comprised of two parts such that $\partial\Omega = \Gamma_D \cup \Gamma_N$, where Γ_D denotes the part of the boundary with *Dirichlet* boundary conditions (BCs) and Γ_N denotes the part of the boundary with *Neumann* BCs. See Figure 3.1 for a schematic.

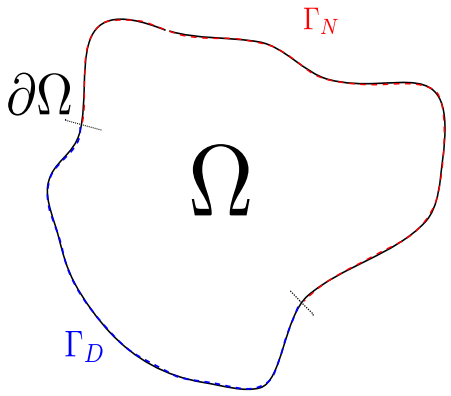


Figure 3.1: A body $\mathcal{G}$ of elastic material occupying Ω, with bounding surface $\partial\Omega \equiv \Gamma_D \cup \Gamma_N$.

Let a point in $\overline{\Omega} = \Omega \cup \partial\Omega$ be denoted by $x = (x_i)_{i=1}^d$. Denote the system of body forces acting on $\mathcal{G}$ by $f = f(x,t)$ and the system of traction forces acting on $\mathcal{G}$ by $g = g(x,t)$. The stress tensor is denoted by $\sigma = (\sigma_{ij})_{i,j=1}^d$ and the strain tensor by $\varepsilon = (\varepsilon_{ij})_{i,j=1}^d$.

Let $\mathcal{B}_\epsilon \equiv \mathcal{B}_\epsilon(x_0)$ be an open ball of radius ϵ centred at the point $x_0 = (x_i)_{i=1}^d \in \overline{\Omega}$. Let the outward surface normal to $\partial\mathcal{B}_\epsilon$ be denoted by $n = (n_i)_{i=1}^d$. The force applied to $\mathcal{B}_\epsilon$ is the sum of

(i) the net force due to body forces acting on the volume of $\mathcal{B}_\epsilon$ and

(ii) the resultant of the surface forces acting on $\partial\mathcal{B}_\epsilon$.

In equilibrium, the sum of the forces acting on the ball must be zero. Thus,

$$\underbrace{\int_{\mathcal{B}_\epsilon} f d\Omega}_{body forces} + \underbrace{\oint_{\partial\mathcal{B}_\epsilon} \sigma \cdot n d(\partial\mathcal{B}_\epsilon)}_{surface forces} = 0. \tag{3.1}$$

Theorem 3.1 (Divergence theorem [34])**.** *Let u be a continuously differentiable vector field, defined in a volume V. Let S be the closed surface forming the boundary of V and let n be the unit outward normal to S. Then the divergence theorem states that*

$$\iiint_V u dV = \oiint_S u \cdot n dS, \tag{3.2}$$

Theorem 3.1 applied to (3.1) gives

$$\int_{\mathcal{B}_\epsilon} (\nabla \cdot \sigma + f) d\Omega = 0. \tag{3.3}$$

Since x_0 and ϵ are arbitrarily chosen the integral sign can be removed, giving the *equilibrium equations* describing the *elastostatic problem*: for all $t \in [0, T]$,

$$-\nabla \cdot \sigma = f, \text{ in } \Omega, \tag{3.4a}$$
$$u = u_D, \text{ on } \Gamma_D, \tag{3.4b}$$
$$\sigma \cdot n = g, \text{ on } \Gamma_N. \tag{3.4c}$$

Here, (3.4b) and (3.4c) are the Dirichlet and Neumann BCs respectively. The system (3.4) represents the governing equations for a general elasticity problem.

The stress tensor is symmetric, i.e. $\sigma_{ij} = \sigma_{ji}$ for $1 \leq i, j \leq d$. Thus, σ is completely described by six (for $\Omega \in \mathbb{R}^3$) or three (for $\Omega \in \mathbb{R}^2$) quantities. The symmetric *strain tensor* $\varepsilon(u) := (\varepsilon_{ij}(u))_{i,j=1}^d$ is defined by

$$\varepsilon(u) = \frac{1}{2} \left(\nabla u + (\nabla u)^T \right). \tag{3.5}$$

The stress tensor σ is defined by a *constitutive relationship* involving ε.

3.2 Simulating the response of the spinal cord

In the hopes of gaining more understanding about conditions affecting the spinal cord, simulations are run using the governing equations (3.4).

The simulations of the response of the spinal cord under pressure are done on a mesh developed from a geometry of a spinal cord segment from a sheep. The mesh was segmented from high resolution diffusion tensor images of a sheep spinal cord [50]. The geometry of the mesh is shown in Figure 3.2. The mesh has 1.15×10^6 cells, and represents a spinal cord segment of approximately 3.4cm in length.

The CSF pressure is simulated by the Neumann boundary condition

$$\sigma \cdot n = g = -pn,$$

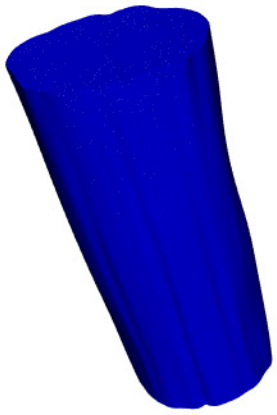


Figure 3.2: The mesh of the spinal cord used in simulations

where σ is the stress tensor, n is the outward surface normal and p is a travelling pressure wave modelled by

$$p(z,t) = p_0(z + ct). \tag{3.6}$$

Here z is the longitudinal coordinate along the spinal cord and c is the wave speed in the z-direction, measured to be $c = 2.0$ m/s. The pressure variation over time is shown in Figure 3.3.

The pressure values displayed in Figure 3.3 are based on inter cranial and lumbar pressure measurements in a Chiari patient [50].

3.2.1 Boundary conditions

The Neumann BCs are defined by $\sigma \cdot n = g = -pn$ and represent the applied pressure on the spinal cord. There is, however, no obvious choice for what the Dirichlet BC u_D should be.

Only a short segment of the spinal cord is used (≈ 3.4cm). Anatomically, this segment would be connected to more spinal cord both above and below. This motivates the use of the same boundary conditions on the top surface and the bottom surface of the spinal cord segment. Denote the top surface by Γ_1 and the bottom surface by Γ_2.

It seems reasonable that the spinal cord segment should be able to be compressed in the radial direction, but not be allowed to move in the axial direction. This

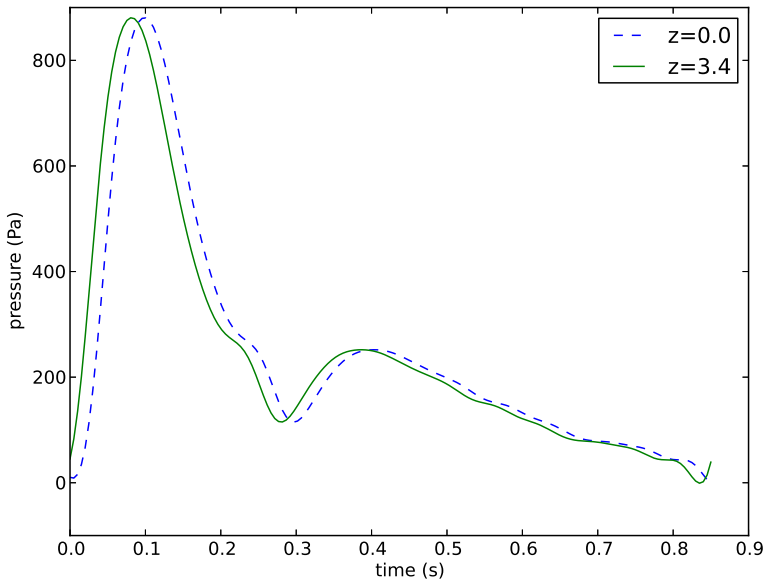


Figure 3.3: Measured inter-cranial pressure variation over time, shown for both $z = 0.0\ cm$ and $z = 3.4\ cm$.

gives rise to the Dirichlet BC $u_z = 0$ on Γ_1 and Γ_2. This means that the spinal cord is allowed to compress / dilate in the xy-plane, but is not allowed to move in the z-direction at the boundaries Γ_1 and Γ_2.

3.3 Constitutive relationships

The equations for the general behaviour of an elastic material when it deforms were derived above and presented in (3.4). A description of how a specific material type reacts to a force is, however, not included in (3.4). There are many types of materials that have elastic properties, but these materials may behave very differently from one another. Thus *constitutive relationships*, describing how a specific material type acts, are needed. In particular a relationship between the stress and the strain of the material, a so-called *stress–strain relationship*, is required.

As mentioned, the constitutive relationship of interest is a viscoelastic one. A purely elastic constitutive relationship is also considered for comparison. For simplicity, it is assumed that the spinal cord is isotropic, and that a linear constitutive relationship may be used to describe it.

3.3.1 Linear elasticity

The linear elastic constitutive relationship is a very simple case. The stress–strain relationship is given by Hooke's law,

$$\sigma = C\varepsilon. \tag{3.7}$$

Here, C is a fourth order tensor known as the *stiffness tensor* or the *elasticity tensor*, which, as implied by its name, gives the stiffness of the material. For an isotropic material, this relationship can be written as

$$\sigma = 2\mu\varepsilon + \lambda \mathrm{tr}(\varepsilon) I. \tag{3.8}$$

Here μ, λ are known as *Lamé parameters*, given as functions of the Young's modulus (E) and Poisson's ratio (ν), which are engineering constants for a given material:

$$\mu = \frac{E}{2(1+\nu)}, \tag{3.9}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}. \tag{3.10}$$

Young's modulus describes the stiffness of the material in question, while Poisson's ratio is a measure of the material's tendency to expand in the direction(s) perpendicular to an applied compression.

3.3.2 Linear viscoelasticity

In the constitutive relationship (3.8), the stress is dependent on the strain and vice versa, but the relationship is independent of time. This is not the case for viscoelastic constitutive relationships, where the *history of deformations* becomes important.

A viscoelastic material has both elastic and viscous properties, and thus the constitutive relationship must reflect this. Viscoelastic materials have two important responses to loading/unloading, namely *creep* and *relaxation*.

Creep occurs when a constant stress is applied to a viscoelastic material over time. The material will have an instantaneous elastic response, followed by a gradual increase in strain over time. If the applied stress is removed, the same process will

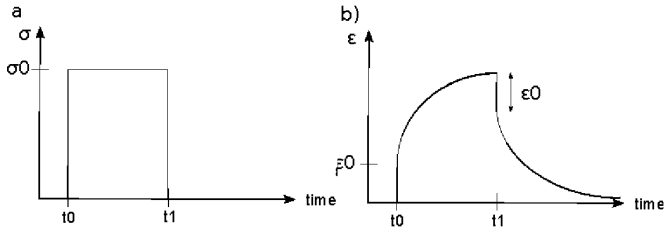


Figure 3.4: a) Applied stress with constant value σ_0 between t_0 and t_1, and b) induced strain due to the applied stress for a viscoelastic material, demonstrating viscoelastic creep.

occur, but the strain will then gradually *decrease*. This phenomenon is illustrated in Figure 3.4.

Relaxation occurs when a constant strain is applied to a viscoelastic material. Again there will be an instantaneous elastic response, this time followed by a gradual decrease in the stress. The relaxation phenomenon is illustrated in Figure 3.5.

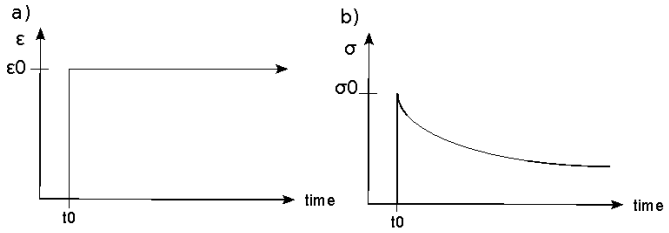


Figure 3.5: a) Applied strain with constant value ε_0 from t_0 and onwards, and b) induced stress due to the applied strain for a viscoelastic material, demonstrating viscoelastic stress relaxation.

Viscoelastic materials are said to have a *continuous memory* – "the state of stress at the instant t depends on the *whole history of the strains* (or of the loads) experienced by the material at the previous instants" [16]. So, a viscoelastic material depends on the history of deformations as well as any forces acting on the body. Mathematically this is represented by a *history integral*, and the constitutive relationship may generally be written on the form

$$\sigma(t) = A(t)\varepsilon(t) + \int_0^t B(t-s)\varepsilon(s)\,ds, \tag{3.11}$$

where A and B are tensors describing material properties. This is an example of a Volterra equation of the second kind. For a detailed review of such equations see for example the text by Linz [31].

Simple constitutive relationships for linearly viscoelastic materials can be built up using various combinations of spring- and dashpot-models. The spring represents the elastic component, and obeys Hooke's law (3.7). The dashpot represents the viscous component and obeys the following stress-strain relationship:

$$\sigma = \eta\dot{\varepsilon}, \tag{3.12}$$

where η gives the viscosity of the dashpot and the superposed dot signifies a derivative in time. In the following, three basic combinations are set up as models for linear viscoelasticity. These are namely the *Maxwell*, the *Kelvin-Voigt (KV)* and the *Standard Linear Solid (SLS)* models.

The Maxwell model

The Maxwell model combines a spring and a dashpot in *series*, using an electric circuit analogue, as shown in Figure 3.6. The stiffness of the spring is given by E and the viscosity of the dashpot is given by η.

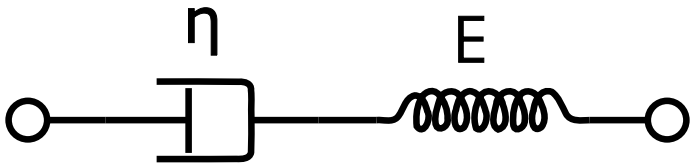


Figure 3.6: Schematic of the Maxwell model.

The total strain is equal to the sum of the strains of the two components,

$$\varepsilon = \varepsilon_S + \varepsilon_D,$$

where the subscript S denotes the spring and the subscript D denotes the dashpot. The stress is the same for both components,

$$\sigma = \sigma_S = \sigma_D.$$

Combining these two relationships gives the constitutive relationship for the Maxwell model:

$$\sigma = \sigma_D = \eta\dot{\varepsilon}_D = \eta(\dot{\varepsilon} - \dot{\varepsilon}_S) = \eta\dot{\varepsilon} - \eta\dot{\sigma}/E.$$

Rearranging, we find that

$$\frac{1}{\eta}\sigma + \frac{1}{E}\dot{\sigma} = \dot{\varepsilon}. \tag{3.13}$$

The Kelvin-Voigt model

This is another simple model, consisting of one dashpot component and one spring component, connected in parallel as shown in Figure 3.7. As with the Maxwell model, E denotes the stiffness of the spring while η denotes the viscosity of the dashpot.

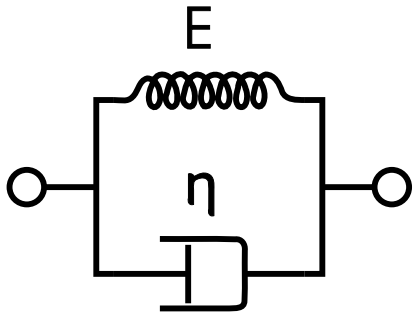


Figure 3.7: Schematic of the Kelvin-Voigt model.

In this case, the strain is the same for the two components,

$$\varepsilon = \varepsilon_S = \varepsilon_D,$$

while the stress is given by the sum of the stresses of the components,

$$\sigma = \sigma_S + \sigma_D.$$

Combining these two relationships gives the constitutive relationship for the KV-model:

$$\sigma = E\varepsilon + \eta\dot{\varepsilon}. \tag{3.14}$$

The Standard Linear Solid model

The SLS model is a combination of a spring and a Maxwell element (a spring and a dashpot in series) in parallel, as shown in Figure 3.8. E_1 and E_2 represent the stiffness of the top and bottom spring respectively, while η represents the viscosity of the dashpot.

As with the KV-model, the strains of the two branches must be equal,

$$\varepsilon = \varepsilon_1 = \varepsilon_2,$$

where the subscript 1 denotes the top branch and the subscript 2 denotes the bottom branch. The strain of the Maxwell element (the bottom branch) is given by the sum of the strains from the dashpot and spring respectively,

$$\varepsilon_2 = \varepsilon_{S2} + \varepsilon_D.$$

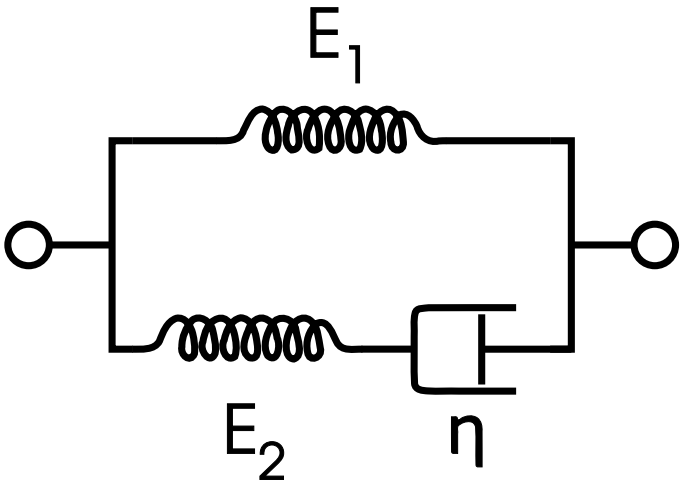


Figure 3.8: Schematic of the SLS model.

The total stress is equal to the sum of the stresses of the branches,

$$\sigma = \sigma_1 + \sigma_2.$$

In the Maxwell branch, the stress is the same in both of the components,

$$\sigma_2 = \sigma_{S2} = \sigma_D.$$

Combining these relationships gives the constitutive relationship for the SLS model. First,

$$\sigma_2 = \eta\dot{\varepsilon}_D = \eta(\dot{\varepsilon} - \dot{\varepsilon}_{S2}) = \eta\dot{\varepsilon} - \eta\dot{\sigma}_2/E_2,$$

and,

$$\eta\dot{\varepsilon} = \sigma_2 + \eta\dot{\sigma}_2/E_2 = \sigma - \sigma_1 + \eta(\dot{\sigma} - \dot{\sigma}_1)/E_2 = \sigma - E_1\varepsilon + \eta\dot{\sigma} - \eta E_1\dot{\varepsilon}/E_2.$$

Therefore,

$$\sigma + \eta\dot{\sigma} = E_1\varepsilon + \eta\left(1 + \frac{E_1}{E_2}\right)\dot{\varepsilon}.$$

Letting $\tau_\varepsilon = \frac{\eta}{E_2}$ and $\tau_\sigma = \eta\frac{E_1+E_2}{E_1E_2}$, the constitutive relationship using the SLS model is given by

$$\sigma + \tau_\varepsilon\dot{\sigma} = E_1(\varepsilon + \tau_\sigma\dot{\varepsilon}). \tag{3.15}$$

3.3.3 Behavioral analysis of the viscoelastic models

So far, three different constitutive relationships for linear viscoelasticity have been introduced. In order to find out which of the relationships is the more accurate for modelling the spinal cord, a creep/relaxation analysis is performed for all three models, (3.13), (3.14) and (3.15). To analyse the models, the stress relaxation and creep functions of the models are considered. Creep and relaxation experiments are simulated, testing how the models respond when a step strain or step stress are applied respectively. Before starting the analysis, some terminology is defined.

Definition 3.2 (Laplace transform)**.** A one-dimensional Laplace transform is given by

$$f(s) = \mathcal{L}\{F(t)\} = \int_0^\infty e^{-st} F(t) dt, \tag{3.16}$$

where $F(t)$ is a function of the real variable t and s is a complex variable [3].

Definition 3.3 (Heaviside step function [3])**.** The Heaviside step function is defined by its properties

$$H(t) = \begin{cases} 0, & t < 0, \\ \frac{1}{2}, & t = 0, \\ 1, & t > 0. \end{cases} \tag{3.17}$$

Definition 3.4 (Dirac delta function)**.** The *Dirac delta function* can be viewed as the derivative of the Heaviside step function [10],

$$\frac{d}{dt}[H(t)] = \delta(t). \tag{3.18}$$

The *Dirac delta function* centred at t_0 has the properties

$$\delta(t) = 0, \quad x \neq t_0, \tag{3.19}$$

$$f(t_0) = \int_a^b f(t)\delta(t - t_0) dx, \tag{3.20}$$

where $f(t)$ is any well-behaved function and $t_0 \in [a, b]$ (adapted from [4]) .

The *stress relaxation function* corresponds to the relaxation that occurs as a result of a set constant strain imposed on the body. Specifically, the stress relaxation function is given by the solution $\sigma(t)$ to the constitutive relationships (3.13), (3.14) or (3.15) when the strain is given by

$$\varepsilon(t) = \varepsilon_0 H(t - t_0), \tag{3.21}$$

where ε_0 is a constant. In the stress relaxation test $\sigma(0) = 0$.

The *creep function* corresponds to the creep that occurs due to an applied constant stress. Specifically, the creep function is given by the solution $\varepsilon(t)$ to (3.13), (3.14) or (3.15) when the stress is given by

$$\sigma(t) = \sigma_0 H(t - t_0), \tag{3.22}$$

where σ_0 is a constant. In the creep test $\varepsilon(0) = 0$.

In both cases, the time $t = t_0$ is when the step strain/stress is applied.

Laplace transforms are used in the analysis to obtain the stress relaxation/creep functions for the different models. The following properties of the Laplace transform for the Heaviside function, the Dirac delta function and for a single derivative are used:

$$\mathcal{L}\{H(t - t_0)f(t)\} = e^{-t_0 s}\mathcal{L}\{f(t)\} \tag{3.23}$$

$$\mathcal{L}\{\delta(t - c)\} = e^{-cs}, \tag{3.24}$$

$$\mathcal{L}\{\dot{y}(t)\} = s\mathcal{L}\{y(t)\} - y(0) \tag{3.25}$$

Here, the *overdot* in (3.25) denotes a derivative with respect to t.

Analysis of the Maxwell model

Solving the differential equation

$$\frac{1}{\eta}\sigma + \frac{1}{E}\dot{\sigma} = \varepsilon_0\delta(t - t_0) \tag{3.26}$$

for $\sigma(t)$ yields the stress relaxation function for the Maxwell model. Taking the Laplace transform on both sides of the equation gives

$$\frac{1}{\eta}\mathcal{L}\{\sigma(t)\} + \frac{1}{E}\mathcal{L}\{\dot{\sigma(t)}\} = \varepsilon_0\mathcal{L}\{\delta(t - t_0)\}. \tag{3.27}$$

Using the properties (3.24) and (3.25) of the Laplace transform for the Dirac delta function and for the single derivative respectively results in

$$\frac{1}{\eta}\mathcal{L}\{\sigma(t)\} + \frac{1}{E}s\mathcal{L}\{\sigma(t)\} = \varepsilon_0\, e^{-t_0 s} \tag{3.28}$$

Letting $\hat{\sigma}(s) = \mathcal{L}\{\sigma(t)\}$, and rearranging:

$$\hat{\sigma}(s) = E\varepsilon_0 \frac{e^{-t_0 s}}{s + \frac{E}{\eta}}. \tag{3.29}$$

The inverse transform of (3.29) gives

$$\sigma(t) = E\, e^{-\frac{E}{\eta}(t-t_0)} \varepsilon_0 H(t - t_0), \tag{3.30}$$

which is the stress relaxation function for the Maxwell model. Plots of the step strain and the stress relaxation function $\sigma(t)$ are shown in Figure 3.9.

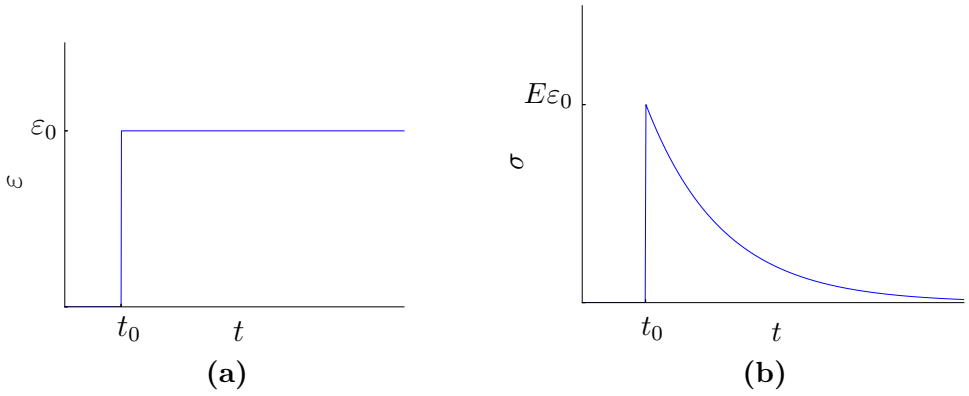


Figure 3.9: Plot of the step strain (a) and the stress relaxation function (b) for the Maxwell model.

The shape of the plot in Figure 3.9b is similar to the shape of the plot of stress relaxation in Figure 3.5 in that there is an instantaneous elastic response followed by a gradual decrease in the stress over time. This suggests that stress relaxation is modelled well by the Maxwell model.

The creep function for the Maxwell model is given by the solution $\varepsilon(t)$ of the differential equation

$$\dot{\varepsilon}(t) = \frac{\sigma_0}{\eta} H(t - t_0) + \frac{\sigma_0}{E} \delta(t - t_0). \tag{3.31}$$

The calculations for the creep function are analogous to the calculations above, and yield

$$\varepsilon(t) = \left[\frac{1}{E} + \frac{1}{\eta}(t - t_0) \right] \sigma_0 H(t - t_0). \tag{3.32}$$

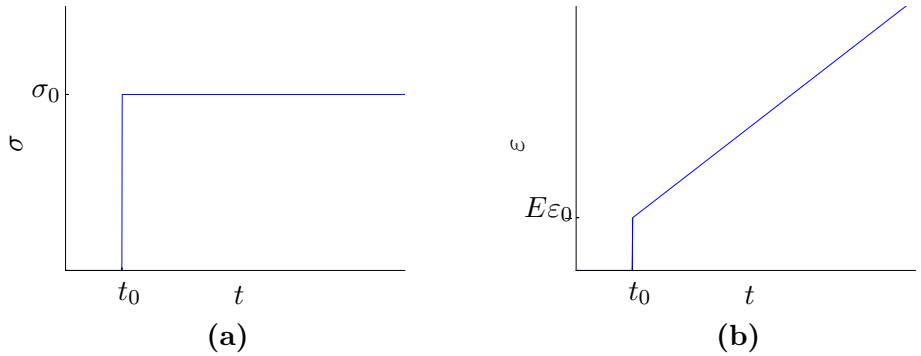


Figure 3.10: Plot of the step stress (a) and the creep function (b) for the Maxwell model.

Plots of the step stress and the creep function $\varepsilon(t)$ are shown in Figure 3.10.

The shape of the plot in Figure 3.10b does not display the creep property shown in Figure 3.4. There is an instantaneous elastic response as expected, but the strain is allowed to increase unbounded over time. A material displaying this property will be allowed unrestricted flow, suggesting that the Maxwell model models the behaviour of a viscoelastic *fluid* rather than a solid.

Analysis of the Kelvin-Voigt model

The KV model is formulated as an explicit expression for the stress, and inserting (3.21) gives the stress relaxation function directly,

$$\sigma(t) = E\varepsilon_0 H(t - t_0) + \eta\varepsilon_0 \delta(t - t_0). \tag{3.33}$$

Plots of the step strain and the stress relaxation function $\sigma(t)$ are shown in Figure 3.11.

Comparing Figure 3.11 and Figure 3.5, we see that the plots do not have similar shapes. In fact, the plot of the stress relaxation function in Figure 3.11b does not display the property of gradually decreasing stress under a constant applied strain at all. So, the KV model does not model stress relaxation.

Solving the differential equation

$$E\varepsilon + \eta\dot{\varepsilon} = \sigma_0 H(t - t_0) \tag{3.34}$$

for $\varepsilon(t)$ yields the creep function for the KV model. The calculations are analogous

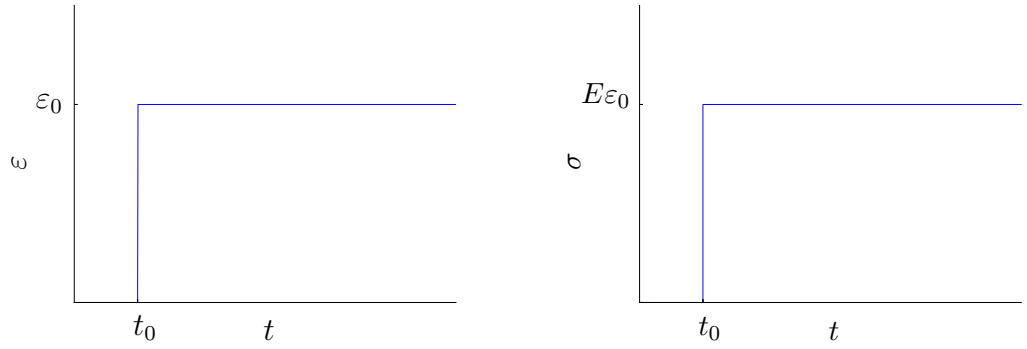


Figure 3.11: Plot of the step strain (a) and the stress relaxation function (b) for the KV model.

to the calculations for the Maxwell model, and yield

$$\varepsilon(t) = \frac{1}{E}\left[1 - e^{-E(t-t_0)/\eta}\right]\sigma_0 H(t-t_0). \tag{3.35}$$

Plots of the step stress and the creep function $\varepsilon(t)$ are shown in Figure 3.12.

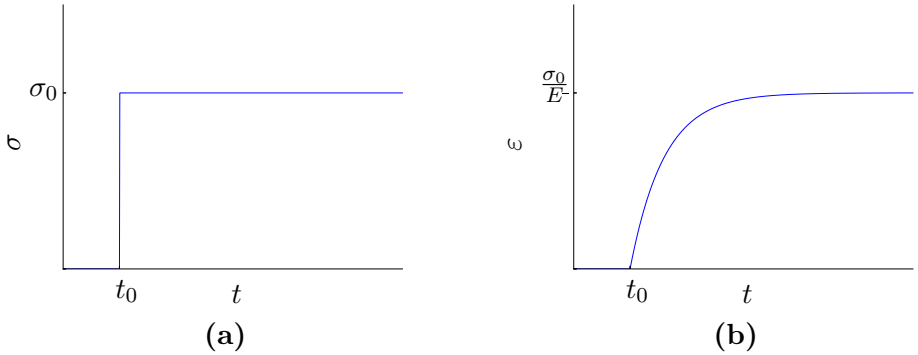


Figure 3.12: Plot of the step stress (a) and the creep function (b) for the KV model.

The creep function in Figure 3.12b has a similar shape as the plot in Figure 3.4, in that there is a gradual, but bounded, increase in the strain when a constant stress is applied. This strain reaches an equilibrium at the value $\frac{\sigma_0}{E}$. Thus it would seem that the KV model works for predicting creep in a viscoelastic solid.

Analysis of the Standard Linear Solid model

The analysis is similar to the analyses above. Laplace transforms are used to find expressions for both the stress relaxation function and the creep function.

Beginning with the stress relaxation function, the differential equation

$$\sigma(t) + \tau_\varepsilon \dot{\sigma}(t) + E_1(\varepsilon_0 H(t - t_0) + \tau_\sigma \varepsilon_0 \delta(t - t_0)) \tag{3.36}$$

is solved for $\sigma(t)$. As before, the Laplace transform is applied on both sides of the equation, and the properties (3.23), (3.24) and (3.25) of the Laplace transform are used. This results in the stress relaxation function for the SLS model,

$$\sigma(t) = [E_1 + E_2\, e^{-(t-t_0)/\tau_\varepsilon}]\varepsilon_0 H(t - t_0). \tag{3.37}$$

The calculations for the creep function are similar, and yield

$$\varepsilon(t) = \frac{1}{E_1}\left[1 + \left(\frac{\tau_\varepsilon}{\tau_\sigma} - 1\right) e^{-(t-t_0)/\tau_\sigma}\right] \sigma_0 H(t - t_0). \tag{3.38}$$

The plots of the stress relaxation and creep functions are shown in Figure 3.13 and Figure 3.14 respectively.

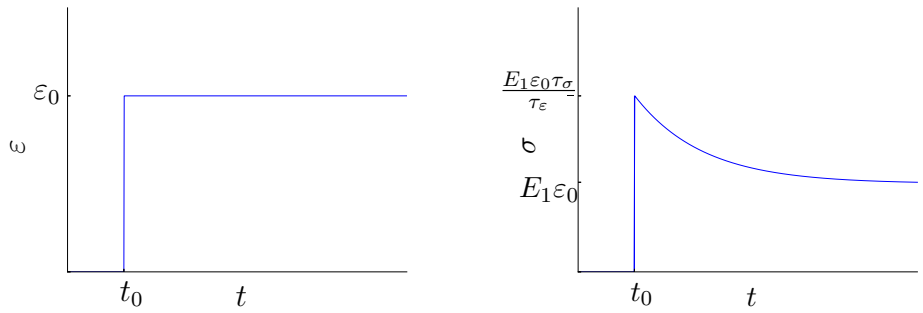


Figure 3.13: Plot of the step strain (a) and the stress relaxation function (b) for the SLS model.

The shape of the plot in Figure 3.13 is similar to that in Figure 3.5, in that there is an instantaneous elastic response when the step strain is applied, followed by a gradual decrease in the stress over time. The similarity of the plots suggests that the SLS model models stress relaxation well.

The shape of the plot in Figure 3.14 is similar to that in Figure 3.4, in that there is an instantaneous elastic response when the step stress is applied, followed by

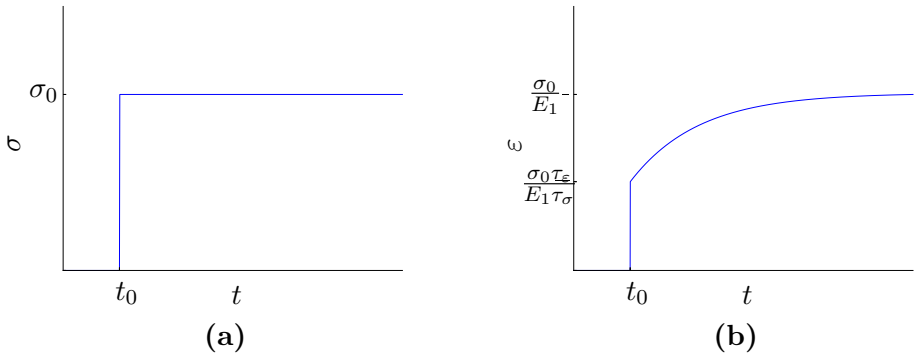


Figure 3.14: Plot of the step stress (a) and the creep function (b) for the SLS model.

a gradual, bounded increase in the strain over time. The similarity of the plots suggests that the SLS model models creep well.

Summary of the behavioral analysis

The outcome of the analysis of all three constitutive relationships are summarized in Table 3.1.

	Models:	
Model	Creep	Relaxation
Maxwell	No*	Yes
KV	Yes	No
SLS	Yes	Yes

Table 3.1: Summary of results from behavioral analysis of the viscoelastic models. * Does not model creep for viscoelastic solids.

Because the spinal cord is assumed to be a solid, the Maxwell model is not adequate for modelling the response of the spinal cord, and is thus discarded. Since it has been shown in numerous experiments that the spinal cord undergoes stress relaxation [12], the KV will not be adequate in modelling the response of the spinal cord, and is also discarded.

The SLS model models both stress relaxation and creep adequately, while modelling a viscoelastic solid. Therefore the chosen model for the viscoelastic response of the spinal cord is the SLS model.

3.4 SLS relationship on integral form

The constitutive relationship (3.15) is given in differential form. Following the methodology in [45], (3.15) can be rewritten in the integral form (3.11) by using an internal variable approach.

This approach is to consider the stresses and strains within each branch from its components separately. This means that in the lowermost branch (Figure 3.8) we must consider a stress and a strain due to the spring, and similarly a stress and a strain due to the dashpot. These stresses and strains are denoted by $\sigma_{S2}, \varepsilon_{S2}$ and $\sigma_{D2}, \varepsilon_{D2}$ respectively. In addition, we have the stress and strain from the uppermost branch, denoted by $\sigma_{S1}, \varepsilon_{S1}$. The springs both obey (3.7), while the dashpot obeys (3.12). So far, this is analogous to the derivation of the SLS model shown earlier in this chapter. However, the following calculations give rise to an equivalent, but different looking, formulation.

The total strain of the system is the same for both branches, but the total strain in the lowermost branch is given by the sum of the strains from the components, giving

$$\varepsilon = \varepsilon_{S2} + \varepsilon_{D2} \tag{3.39}$$

The stress over the lowermost branch is the same throughout, thus

$$\sigma_{D2} = \sigma_{S2}. \tag{3.40}$$

This gives

$$E_2\varepsilon_{S2} = \sigma_{D2} = \eta\dot{\varepsilon}_{D2} = \eta(\dot{\varepsilon} - \dot{\varepsilon}_{S2}). \tag{3.41}$$

Rearranging (3.41) gives the differential equation

$$\dot{\varepsilon}_{S2} + \frac{1}{\tau}\varepsilon_{S2} = \dot{\varepsilon} \tag{3.42}$$

where $\tau = \eta/E_2$, with solution

$$\varepsilon_{S2}(t) = e^{-t/\tau}\varepsilon(0) + \int_0^t e^{-(t-s)/\tau}\varepsilon_s(s)ds. \tag{3.43}$$

Introducing the *stress relaxation function*

$$D(t) = E_1 + E_2\, e^{-t/\tau}, \tag{3.44}$$

using the fact that the total stress of the system is given as the sum of the stresses from the branches, and integrating by parts gives

$$\sigma(t) = D(0)\varepsilon(t) - \int_0^t D_s(t-s)\varepsilon(s)ds, \tag{3.45}$$

where the subscript $_s$ denotes a derivative with respect to the variable s. This is our constitutive relationship on the form (3.11).

3.5 Extending the SLS model to 2D and 3D

The SLS model is developed using a combination of two springs and a dashpot. The use of these components only takes into account the extension/contraction along one axis; in the case shown in Figure 3.8, this is the horizontal axis. The extension of the model from 1D to 2D/3D is done analogously to the linear elasticity case, which is explained here. Elastic materials (in 1D, a spring) obey Hooke's law. In 1D, Hooke's law may be written as

$$\sigma = E\varepsilon,$$

where E is Young's modulus. This relationship may also be written as

$$\sigma = 2\mu\varepsilon, \tag{3.46}$$

where μ is one of the Lamé parameters, defined in (3.9) in terms of the Young's modulus and the Poisson ratio ν. The parameter μ is also known as the *shear modulus*. In 1D, $\nu = 0$ and $E = 2\mu$.

Note that while (3.46) is valid in 1D, 2D and 3D, it is missing a term compared to (3.8). This is because in 2D and 3D, effects of *compressibility* become important. In 1D, $\nu = 0$, because there are no other directions than along the given axis, and a material can thus not expand in directions perpendicular to the applied stress. Thus, (3.46) is enough. However, in 2D and 3D the so-called Poisson effect becomes important, and the term $\lambda \mathrm{tr}(\varepsilon)I$ is non-zero, resulting in (3.8).

Since the constitutive relationship for the SLS model is built up from the constitutive relationships of springs and dashpots, a term addressing the Poisson effect

must be included in order to make the model accurate for 2D and 3D. Redefining the functions $D(t)$ and $D_s(t)$ to instead be *operators* on $\varepsilon(t)$, where

$$D(t)\varepsilon(t) = 2\mu(t)\varepsilon(t) + \lambda(t)\mathrm{tr}(\varepsilon(t))I, \tag{3.47}$$
$$D_s(t-s)\varepsilon(s) = 2\mu_s(t-s)\varepsilon(s) + \lambda_s(t-s)\mathrm{tr}(\varepsilon(s))I, \tag{3.48}$$

the constitutive relationship (3.45) also applies in 2D and 3D. The functions $\mu(t)$ and $\lambda(t)$ are stress relaxation functions on the form (3.44),

$$\mu(t) = \mu_1 + \mu_2\, e^{-t/\tau},$$
$$\lambda(t) = \lambda_1 + \lambda_2\, e^{-t/\tau}.$$

Assuming that ν is constant in time, $\lambda(t)$ may be defined by

$$\lambda(t) = C\mu(t), \tag{3.49}$$

where C is a constant [44], and is chosen to reflect the value of ν. Letting $E_1 = 2\mu_1$ and $E_2 = 2\mu_2$ only the parameters E_1 and E_2 need to be determined.

3.6 Determining parameters for the models

The constitutive relationships (3.8) and (3.45) to be used in simulating the response of the spinal cord both require respective parameters. The relationship (3.45) requires two elasticity parameters, E_1 and E_2 representing the stiffness of the springs in the SLS model, and one viscous parameter η, representing the viscosity of the dashpot in the SLS model, see Figure 3.8. The relationship (3.8) requires the two Lamé parameters μ and λ. As mentioned in subsection 3.3.1, these parameters may be expressed in terms of Young's modulus and Poisson ratio, as shown in (3.9)-(3.10).

By finding parameter values that are used in the literature, appropriate values for representing the spinal cord may be determined. This section aims at giving an overview of the information available in literature on the spinal cord. This is a challenging task, considering the amount of material that is available. Overall, the range of values that appear in the literature for elastic or viscoelastic parameters of the spinal cord (and brain) is quite large, for example in the order of $1 - 1000$kPa in magnitude for the Young's modulus. In addition, there are very few parameter values that are directly applicable to the SLS model. Parameter values from the literature therefore need to be adapted to the SLS model.

The papers reviewed are concerned with material properties for both the spinal cord and the brain. This review is not in any way exhaustive. For detailed reviews

of material properties of the spinal cord, either from experimental data or used in computer simulations, see the reviews by for example Cheng et al. [12] or Banks et al. [5], results from which are included here.

3.6.1 Parameter summary for linear elasticity

Parameters found in the literature for the elastic properties of the spinal cord are summarized in Table 3.2.

Paper / article	Region	Model	Parameters
Hung et al. [25]	spinal cord	Experimental	$E = 0.26$ MPa
Ben-Hatira et al. [6]	spinal cord	Nonlinear elastic	$E = 1.4$ MPa $\nu = 0.499$
Ozawa et al. [37]	spinal cord	Experimental	$E = 16$ kPa
Smith and Humphrey [48]	brain	Poroelastic	$E = 5.0$kPa $\nu = 0.479$
Cheng et al. [12]	spinal cord	Review	$E = 0.0119 - 1.98$MPa
Clarke [13]	spinal cord	Review	$E = 0.012 - 1.37$MPa
Persson et al. [38]	spinal cord	Review (linear elastic)	$E = 0.26 - 1.3$MPa

Table 3.2: Parameter summary from literature, including values reported in review articles.

Note that there is quite a large range of values reported / used in the literature. For example, [50] reported a span of $5 - 1000$kPa for values of Young's modulus in literature concerning studies on syringomyelia. The reviews by Cheng et al. [12] and Clarke [13] report many of the same values from the same studies. Note also that the values reported in Persson et al. [38] are relatively high. It is suggested by Cheng et al. [12] and Clarke [13] that an increased strain rate may lead to a higher reported pseudo modulus (the slope of the linear part of the stress-strain curve), meaning that the values for Young's modulus in the order of magnitude 1MPa may be more relevant when dealing with traumatic spinal cord *injury*, and less relevant for the simulations in this thesis.

The study by Støverud et al. [50] modelling the SC as a poroelastic medium also used a linear elasticity model as a reference material. The parameter values used in the study were the Young's modulus reported by Ozawa et al. [37], $E = 16kPa$

and Poisson's ratio $\nu = 0.479$ used by Smith and Humphrey [48]. In order to compare results, these are the values we select for the linear elasticity model.

3.6.2 Parameter summary for viscoelasticity

Parameters found in the literature for viscoelastic properties of the spinal cord are summarized in Table 3.3.

Paper / article	Region	Model	Parameters
Miller [35]	brain	Hyperelastic linear viscoelastic	$E_1 = 1.05$ kPa $E_2 = 1.96$ kPa
Klatt et al. [28]	brain	SLS	$E_1 = 0.84$ kPa $E_2 = 2.03$ kPa $\eta = 6.7$ Pa s
Sack et al. [42]	brain	KV	$E = 1.56 \pm 0.07$ kPa $\eta = 3.4 \pm 0.2$
Green et al. [21]	brain	Experimental (KV)	$G' = 2.9$kPa ($E = 2.9$kPa) $G'' = 2.5$kPa ($\eta = 4.42$Pa s)

Table 3.3: Summary of parameter values found in the literature for viscoelastic models for the spinal cord

Note that the second entry in Table 3.3 are parameter values that have been developed specifically for the SLS model. This is very valuable, and the values found by Klatt et al. [28] are therefore used as the *default parameters* for the simulations of the spinal cord with the SLS model. More details of the data obtained by Klatt et al. [28] is shown in Table 3.4 and Table 3.5 below.

From Tables 3.4 and 3.5 it is clear that experimental results vary a good deal, sometimes also with the same test subject. Getting accurate parameter values is therefore a challenge.

The parameter values obtained by Klatt et al. [28] are from MRE experiments on the *brain*. However, since the spinal cord is made up of grey and white matter it is assumed that these values may be applied to the spinal cord. These values also provide a rough reference for the ratios between the parameters E_1, E_2 and η.

Note that the elastic parameter values appear to be generally much higher than the viscoelastic parameter values / properties. It is possible that this is because the parameter values from for example Klatt et al. [28] are developed from experiments on brain tissue, which may well be softer than spinal cord tissue. The difference

Parameter	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Mean*
η (Pa s)	6.3	5.6	6.5	7.2	6.4 (0.7)
E_1 (kPa)	1.08	1.29	0.72	0.91	1.00 (0.24)
E_2 (kPa)	2.27	2.42	1.82	2.26	2.19 (0.26)
χ (kPa)	0.05	0.17	0.05	0.12	0.08

Table 3.4: Volunteer 1, brain experiments: viscoelastic parameters and error of the fit χ according to the SLS model [28]. The parameters are derived from four independent experiments. The mean (with SD in brackets) for η, E_1 and E_2 was calculated from the parameter values over the four experiments, while $\chi*$ was calculated as the error to the fit using the mean viscoelastic parameters.

Parameter	Vol. 1	Vol. 2	Vol. 3	Vol. 4	Vol. 5	Mean**
$\eta*$ (Pa s)	6.4 (0.7)	7.5 (0.3)	7.6 (2.0)	5.2 (0.1)	6.7 (1.3)	6.7 (1.0)
E_1* (kPa)	1.00 (0.24)	0.67 (0.10)	0.65 (0.24)	1.00 (0.08)	0.87 (0.13)	0.84 (0.17)
E_2* (kPa)	2.19 (0.26)	1.85 (0.06)	2.03 (0.13)	2.10 (0.20)	1.96 (0.10)	2.03 (0.13)
$\chi*$ (kPa)	0.08	0.13	0.11	0.14	0.09	0.10

Table 3.5: Results of brain experiments: mean viscoelastic parameters and $\chi*$ of the brain of five volunteers (Vol. 1-5) according to the Zener (or SLS) model (SD in brackets). The asterisk indicates that the given quantities were averaged over four independent experiments. The interindividual mean and SD (**) of the viscoelastic parameters are shown in the last column, whereas $\chi**$ is calculated as $\chi*$ in Table 3.4, this time using the interindividual mean parameter values.

may also arise due to the differences in methodology. However, the values from Klatt et al. [28], Sack et al. [42] and Green et al. [21] are in quite good agreement. In addition, the values reported in a review by Bilston [9] for low strain rates are of similar magnitudes.

3.6.3 How should parameters be chosen?

The parameter values found in the literature for viscoelastic models such as the SLS model seem to be relatively consistent. However, values for Young's modulus vary greatly in magnitude. A question of how parameters should be chosen such that the results from the linear elasticity model and the SLS model may be compared arises.

Clearly, in order to compare the results in this thesis with the results using the poroelastic / purely elastic equations in [50], the same Young's modulus should be used, i.e. $E = 16\text{kPa}$. On the other hand, the values obtained by Klatt et al. [28] are directly applicable to the SLS model. Comparing the magnitudes of the displacements from a simple, preliminary simulation using $E = 16\text{kPa}, \nu = 0.479$ for the linear elasticity case, and $E_1 = 0.84\text{kPa}, E_2 = 2.03\text{kPa}, \eta = 6.7\text{Pas}$ for the linear viscoelasticity case with the SLS model, results in the maximum displacement in a single point in the mesh to have values listed in Table 3.6.

Model	Max. displacement
Linear elasticity ($E = 16\text{kPa}$)	$4.65\times10^{-4}\text{cm}$
SLS ($E_1 = 0.84\text{kPa}, E_2 = 2.03\text{kPa}, \eta = 6.7\text{Pas}$)	$1.18 \times 10^{-2}\text{cm}$

Table 3.6: Comparing magnitudes for displacement in a single point in the solution with linear elasticity and SLS model respectively. Note that the same point was used for both models.

The results in Table 3.6 show maximum displacements in a chosen point that are two orders of magnitude different. Thus, it is unlikely that it is of interest to compare displacement magnitude results of simulations run with these parameters.

In order to obtain results that are comparable, tests are done to find out for what value of E the solutions are closest in magnitude when $E_1 = 0.84\text{kPa}, E_2 = 2.03\text{kPa}, \eta = 6.7\text{Pas}$, and vice versa. The results are summarized in Table 3.7. All the parameters listed in Table 3.7 will be used in simulations.

Parameter values	
Linear Elasticity	SLS model
$E = 0.65\text{kPa}*, \nu = 0.479$	$E_1 = 0.84\text{kPa}, E_2 = 2.03\text{kPa}, \eta = 6.7\text{Pa s}$
$E = 16\text{kPa}, \nu = 0.479$	$E_1 = 21\text{kPa}*, E_2 = 52.5\text{kPa}*, \eta = 0.17\text{kPa s}*$

Table 3.7: Parameter values giving corresponding displacements.
*: Calculated value.

Another interesting parameter to investigate is η – what happens when η is doubled? Therefore, simulations will also be run with the default values, but

where η has a higher value. A summary of the parameters to be used in the simulations is given in Table 6.10 in section 6.2.

Chapter 4

Numerical methods for PDEs

Numerical methods are widely used to solve many different types of mathematical problems, such as partial differential equations (PDEs).

This chapter aims at creating an overview of the methods that will be used to solve the elasticity and viscoelasticity problems numerically. A review of the solution of partial differential equations using the finite element method is provided, followed by a brief introduction to numerical integration. Sources of error in the numerical methods are identified and discussed. Finally, methods for solving the large systems that arise from the discretization of the elasticity and viscoelasticity problems are discussed.

4.1 The finite element method

The finite element method is a general method for solving differential equations, particularly PDEs, numerically. Since partial differential equations can be very complicated and thus difficult to solve analytically, numerical methods may be useful alternatives; the finite element method is one such method. This section contains a short introduction to the finite element method. For illustration purposes, an example equation is used. For a more thorough review of the finite element method, see for example [27, 30].

Consider Poisson's equation, find $u = u(x)$ such that

$$-\Delta u = f, \quad x \in \Omega, \tag{4.1a}$$

$$u = u_D, \quad x \in \Gamma_D, \tag{4.1b}$$

$$-\partial_n u = g, \quad x \in \Gamma_N. \tag{4.1c}$$

Here, Δ is the Laplace operator, $\Delta = \nabla^2$. As before, Γ_D denotes the Dirichlet boundary, where the solution u has a prescribed value, while Γ_N denotes the Neumann boundary. The notation ∂_n represents the normal derivative, given by

$$\partial_n u = \nabla u \cdot n$$

.

The method is as follows: the equation (4.1a) is multiplied with a *test function* v and integrated over the domain, giving

$$\int_\Omega -\Delta u \cdot v \; dx = \int_\Omega f \cdot v \; dx \tag{4.2}$$

The test function is chosen such that it vanishes at the Dirichlet boundary, $v|_{\Gamma_D} = 0$. The left hand side is integrated by parts in order to reduce the double derivative to a single derivative:

$$\int_\Omega -\Delta u \cdot v \; d\Omega = \int_\Omega \nabla u \cdot \nabla v \; d\Omega - \int_{\partial\Omega} \partial_n u \cdot v d\partial\Omega \tag{4.3}$$

Inserting (4.3) into (4.2), and splitting the boundary into Γ_D and Γ_N give

$$\int_\Omega \nabla u \cdot \nabla v \; d\Omega - \int_{\Gamma_D} \partial_n u \cdot v \; d\Gamma - \int_{\Gamma_N} \partial_n u \cdot v \; d\Gamma = \int_\Omega f \cdot v \; d\Omega. \tag{4.4}$$

Since the test function v is required to vanish on the Dirichlet boundary, the second term on the left hand side of (4.4) also vanishes. Inserting the Neumann boundary condition (4.1c) and defining

$$V = \{v \in H^1(\Omega) : v = u_D \text{ on } \Gamma_D\}, \tag{4.5}$$

$$\hat{V} = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}, \tag{4.6}$$

the *variational formulation* of (4.1) is given by:

Find $u \in V$ such that

$$\int_\Omega \nabla u \cdot \nabla v \; d\Omega = \int_\Omega f \cdot v \; d\Omega - \int_{\Gamma_N} g \cdot v \; d\Gamma, \;\; \forall v \in \hat{V}. \tag{4.7}$$

The space V is called the *trial space* and the space $\hat{V}$ is called the *test space*. As such the unknown function u is known as a *trial function*, while, as mentioned, v is known as a *test function*.

Defining for this example

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \; d\Omega,$$
$$L(v) = \int_\Omega f \cdot v \; d\Omega - \int_{\Gamma_N} g \cdot v \; d\Gamma,$$

the variational form can also be written on a more abstract form:

Find $u \in V$ such that

$$a(u,v) = L(v), \;\; \forall v \in \hat{V}. \tag{4.8}$$

Here, $a(\cdot,\cdot)$ is called a *bilinear form* and $L(\cdot)$ is called a *linear form*. This *abstract formalism* will be used extensively in the discretization of the elasticity problems. However, in order to see the details of the finite element method, the explicit versions of the variational form will be used in the remainder of this example.

The problem (4.1) may be discretized by restricting the variational formulation (4.7) to the discrete function spaces V^h and $\hat{V}^h$. To obtain these function spaces, the domain Ω is partitioned into a set of cells, $\mathcal{T}$, such that $T|_{T\in\mathcal{T}} = \Omega$. The cells may not overlap. Together, the cells make up the *mesh*.

On each cell, a local function space $\mathcal{V}$ is defined. Each cell also has a set of degrees of freedom, called *nodes*. The number of nodes on the cell depend on the function space $\mathcal{V}$. The function space $\mathcal{V}$ can be represented by a set of *basis functions*. The cell, together with the function space $\mathcal{V}$ and the nodes make up a *finite element*.

Cells are typically triangles in 2D and tetrahedra in 3D. An example of a finite element is the Lagrange element of degree 1, also known as the P1 element. In 2D it is a triangle, with three nodes, as illustrated in Figure 4.1.

On this triangle, the local function space $\mathcal{V} = \mathcal{P}_1$ is defined, that is, the space of all polynomials of degree 1. For the P1 element, the basis functions for $\mathcal{V}$ must be continuous with the basis functions of neighbouring elements in the nodes. Combining all the cells, the global function space for the mesh is made up of the local function spaces, and is in this example thus *piecewise linear*. The finite element approximation, with the use of P1 elements, will then be a piecewise linear approximation. For a thorough review of finite elements and finite element function spaces, see for example [27].

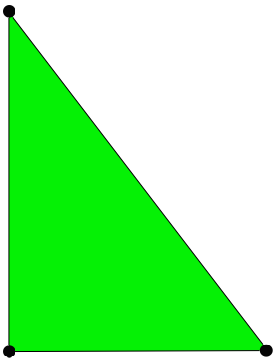


Figure 4.1: The Lagrange finite element of degree 1 in 2D. Nodes are displayed as black dots.

In general, it is assumed that there is a basis $\{\phi_j\}_{j=1}^N$ for V^h and a basis $\{\hat{\phi}_i\}_{i=1}^N$ for $\hat{V}^h$, made up from combining the local function spaces for each cell in the mesh.

The discrete finite element formulation is given by:

find $u^h \in V^h \subset V$ such that

$$\int_\Omega \nabla u^h \cdot \nabla v \; d\Omega = \int_\Omega f \cdot v \; d\Omega - \int_{\Gamma_N} g \cdot v \; d\Gamma, \;\; \forall v \in \hat{V}^h \subset \hat{V}, \tag{4.9}$$

Using abstract formalism:

Find $u^h \in V^h$ such that

$$a(u^h, v) = L(v), \;\; \forall v \in \hat{V}^h, \tag{4.10}$$

where $a(\cdot,\cdot)$ and $L(\cdot)$ are defined as before.

The finite element solution u^h may be approximated as a linear combination of the basis functions of V^h,

$$u^h = u^h(x) = \sum_{j=1}^{N} U_j \phi_j(x). \tag{4.11}$$

Here, $U \in \mathbb{R}^N$ is an unknown vector of degrees of freedom to be computed. Once U is computed, the solution u^h can be found via (4.11). Inserting (4.11) into (4.9) and replacing v with $\hat{\phi}_i(x)$ gives

$$\int_\Omega \nabla \left(\sum_{j=1}^{N} U_j \phi_j \right) \cdot \nabla \hat{\phi}_i \, d\Omega = \int_\Omega f \cdot \hat{\phi}_i \, d\Omega - \int_{\Gamma_N} g \cdot \hat{\phi}_i \, d\Gamma, \;\; i = 1, ..., N$$

By linearity, the sum $\sum_{j=1}^{N} U_j$ can be moved outside the integral, giving

$$\sum_{j=1}^{N} U_j \int_\Omega \nabla \phi_j \cdot \nabla \hat{\phi}_i \, d\Omega = \int_\Omega f \cdot \hat{\phi}_i \, d\Omega - \int_{\Gamma_N} g \cdot \hat{\phi}_i \, d\Gamma, \;\; i = 1, ..., N$$

Define the matrix $A = \{A_{ij}\}_{i,j=1}^{N}$ by

$$A_{ij} = \int_\Omega \nabla \phi_j \cdot \nabla \hat{\phi}_i \, d\Omega,$$

and the vector $b = \{b_i\}_{i=1}^{N}$ by

$$b_i = \int_\Omega f \cdot \hat{\phi}_i \, d\Omega - \int_{\Gamma_N} g \cdot \hat{\phi}_i \, d\Gamma.$$

To compute U, and thus to find the solution u^h, is then a matter of solving

$$AU = b. \tag{4.12}$$

Thus, the finite element method ultimately results in a linear system of equations. Methods for solving such systems are introduced in section 4.4.

4.2 Numerical integration

As with differential equations, there exist many complex integrals that are challenging, if not impossible, to solve analytically. This motivates the use of numerical methods for calculating integrals. There exists a large amount of different methods for this – in this section the basic idea of numerical integration is explained, and the chosen method, known as the *trapezoidal rule* is introduced. The following review of numerical integration is adapted from the textbook by Dahlquist & Björck [15].

Numerical integration involves the approximation of a definite integral,

$$I[f] = \int_a^b f(t)dx. \tag{4.13}$$

Here, $f = f(t)$ is a given function and $[a, b]$ is a finite interval. A numerical integration rule, or *quadrature rule*, typically takes the form

$$I[f] \approx \sum_{i=1}^{n} w_i f(t_i), \tag{4.14}$$

where the function $f(t)$ is evaluated in the discrete points/nodes $t_1 < t_2 < \cdots < t_n$ in $[a, b]$, and $w_1, w_2, ..., w_n$ are *weights*. There are many examples of quadrature rules – in this section interpolatory quadrature formulas will be considered. Specifically, only rules where the nodes $t_i,\ i = 1 : N$ are equally spaced will be reviewed. These are called *Newton-Cotes* formulas.

Interpolatory quadrature rules approximate the integral in question by

$$\int_a^b f(t)dx \approx \int_a^b p(t)dx, \tag{4.15}$$

where $p(t)$ is the unique polynomial of degree $n-1$ interpolating the function $f(t)$ at the distinct points $t_1 < t_2 < \cdots < t_n$. Classical examples of such quadrature rules include the trapezoidal, midpoint and Simpson's rules. For simplicity the trapezoidal rule is used for numerical integration in this thesis. An overview of how the trapezoidal rule is derived is given below, and the error in the trapezoidal rule is stated.

4.2.1 The trapezoidal rule

The trapezoidal rule is based on *linear* interpolation of $f(t)$ in two nodes, $t_1 = a$ and $t_2 = b$. Thus $f(t)$ is approximated by

$$p(t) = f(a) + (t - a)\frac{f(b) - f(a)}{b - a}.$$

The integral represents the area under the graph of the function, thus the integral is approximated by

$$\int_a^b f(t)dt \approx \frac{(b-a)}{2}(f(a)+f(b)), \tag{4.16}$$

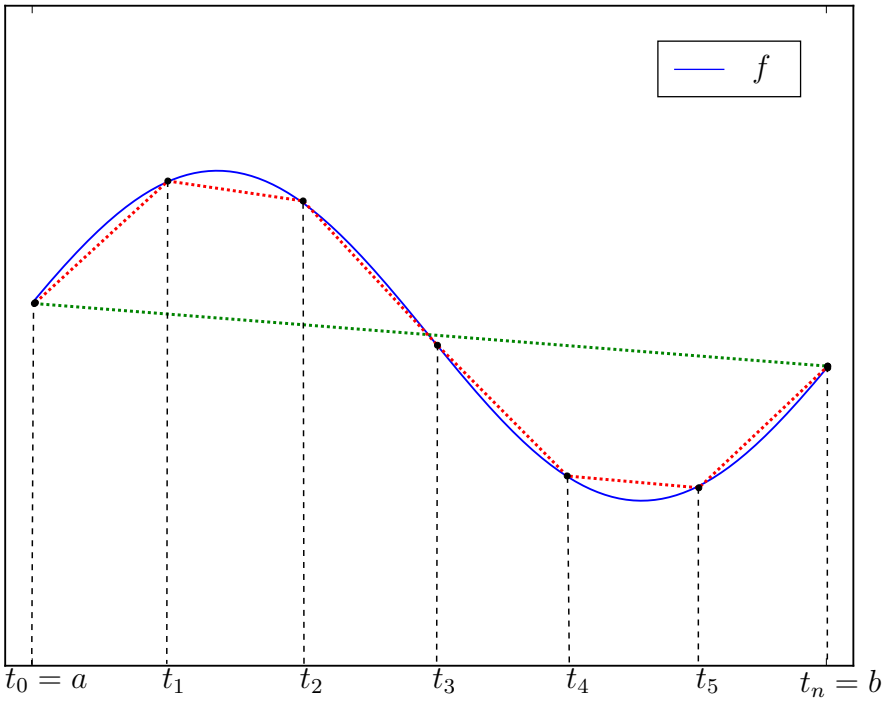


Figure 4.2: Illustration of the trapezoidal rule with one interval $n = 1$ (green) and six intervals $n = 6$ (red).

Unless the function f is linear, a linear interpolation in two points is generally not very accurate. A simple way to improve the accuracy of the approximation is to divide the interval $[a, b]$ into n subintervals $[t_i, t_{i+1}]$ where

$$t_0 = a, \quad t_i = t_0 + i\Delta t, \quad t_n = b.$$

See Figure 4.2 for an example. It is assumed that $f(t_i)$ is known. Here, $\Delta t = (b-a)/n$ is the (time) *step length*. The trapezoidal approximation is then used on each of the subintervals, so

$$\int_{t_i}^{t_{i+1}} f(t)dt \approx \frac{\Delta t}{2}(f(t_i) + f(t_{i+1})).$$

Summing the contributions from all the subintervals gives rise to the *composite trapezoidal rule*,

$$\int_a^b f(t)dt = \frac{\Delta t}{2}(f_0 + f_n) + \Delta t \sum_{i=1}^{n-1} f_i + R_T, \tag{4.17}$$

where $f_i = f(x_i)$. R_T is the global truncation error in the trapezoidal rule, given by

$$R_T = -\frac{1}{12}(b-a)\Delta t^2 f''(\xi), \;\; \xi \in [a,b] \tag{4.18}$$

From the expression for the truncation error, it is clear that the error depends on Δt^2. This means that decreasing Δt by a factor 2 should result in a decrease in the error by a factor of $2^2 = 4$.

4.3 Error in the numerical solution

The numerical solution is an *approximation* to the exact solution of the problem, and as an approximation, the numerical solution contains some error. This section aims at addressing the sources of such errors.

There are two overall approximations made in the numerical scheme (5.16) that is developed in the next chapter, namely the finite element approximation, and approximation of the temporal integral in (3.45) using numerical quadrature. These two approximations are the main source of error.

The error in the numerical solution is defined as the difference between the true solution u and the numerical approximation u^h, such that

$$e_h = u - u^h.$$

To obtain a single number that represents the size of the error in the numerical solution, the $L2$-norm of the error may be used,

$$||e_h||_{L^2(\Omega)} = ||u - u^h||_{L^2(\Omega)}, \tag{4.19}$$

where the $L2$-norm is defined by

$$||v||_{L^2(\Omega)} = \left(\int_\Omega |v|^2 \, d\Omega \right)^{\frac{1}{2}} \tag{4.20}$$

The $L2$-norm is useful because it gives information about the average size of a function [30], thus providing a general overview of how the error behaves. Two different types of error estimates are *a priori* estimates which provide an estimate of the error in terms of the unknown solution u, and *a posteriori* estimates which provide an error estimate in terms of the finite element approximation u^h. A priori and a posteriori error estimates for the linear elasticity problem are well defined and may be found in the text by Larson and Bengzon [30]. A priori and a posteriori error estimates for the linear viscoelasticity problem using the SLS model may be found in the papers by Shaw and Whiteman [47] and Shaw and Whiteman [46] respectively. The error analysis in this thesis will focus on the actual error values given by (4.19) when computing u^h. These errors will be analysed when verifying the implementation of the numerical schemes.

The finite element method provides an approximation in space, and is thus dependent on the *resolution* of the mesh on which the problem is solved. This means that the solution should get better and better (i.e. the error should get smaller) when the mesh is refined. We use the variable h to denote the largest cell diameter in the mesh, or more precisely twice the largest circumradius in the mesh. As h gets smaller, so should the error. Specifically, when using the $L2$-norm, we expect the error to converge at a rate of 2 when using elements of degree 1, and at a rate of 3 when using elements of degree 2 [27]. To exemplify what this means, if we double the mesh resolution, i.e. decrease h by a factor of 2, a convergence rate of 1 corresponds to the error also decreasing by a factor 2; a convergence rate of 2 corresponds to the error decreasing by a factor 4; a convergence rate of 3 corresponds to the error decreasing by a factor 8, and so on. The convergence rate of the error can be calculated by the formula

$$r = \frac{\ln(\frac{E_i}{E_{i-1}})}{\ln(\frac{h_i}{h_{i-1}})}, \tag{4.21}$$

where i is the experiment number. Therefore, to calculate convergence rates, the errors from at least two experiments with different values of h are required.

As mentioned, the second approximation that appears in the numerical scheme (5.16) is the use of numerical quadrature to approximate the time integral in the viscoelastic constitutive relationship. Note that this error source is not relevant for the linear elasticity case. Again, we are most interested in the actual error values

from the numerical scheme, rather than a theoretical analysis. The errornorm (4.19) is used to measure the size of the error. In this case, as Δt becomes smaller, it is expected that the numerical solution will become more accurate. The expected convergence rate for the trapezoidal rule is 2, seeing as the truncation error goes as Δt^2 (see section 4.2). Therefore we expect a convergence rate of 2 in time for the error in the numerical solution for the linear viscoelasticity case.

In order to test whether an implementation displays the expected convergence rates in the errornorm, an *exact* solution is needed. Because finding an exact solution to the linear elasticity and linear viscoelasticity problems can be challenging, a different approach is used.

4.3.1 The method of manufactured solutions

The chosen method is called *manufactured solutions*, in which a solution u_e, any solution, is chosen. The terms in the governing equations (3.4) are calculated from this solution. Specifically, the constitutive relationship, either (3.8) or (3.45), is used to calculate σ from the chosen solution, u. Then the Neumann boundary term, g, is calculated from its definition, $\sigma \cdot n = g$, and the source term, f, is calculated from the governing equation

$$-\nabla \cdot \sigma = f$$

A test is performed to check whether the implementation returns the chosen solution when the calculated values are used.

The error in the solution u^h returned by the implemented solver when compared to the manufactured solution u_e is recorded for several values of h, and, in the linear viscoelasticity case, for several values of Δt.

4.4 Solving linear systems of equations

As mentioned in section 4.1, using the finite element method on PDEs results in a system of equations that must be solved. In the context of the linear elasticity and linear viscoelasticity problems, these systems are *linear*. In general, a linear system

$$Ax = b \tag{4.22}$$

is solved by computing the inverse of the matrix A, and multiplying it with the system such that

$$A^{-1}Ax = A^{-1}b$$
$$\Rightarrow x = A^{-1}b$$

This may, however, be problematic in the context of the finite element method. First of all, the matrix A is typically *sparse* [27], containing $\mathcal{O}(N)$ non-zero entries. The inverse of A will not be sparse, which means that calculating

$$x = A^{-1}b$$

requires $\mathcal{O}(N^3)$ operations. This is very costly when the system becomes large, and motivates the use of faster methods to solve the system.

4.4.1 Direct methods for linear systems

A classical method for solving linear systems is *Gaussian elimination*, in which the matrix A is turned into an upper triangular matrix through row operations, see Figure 4.3.

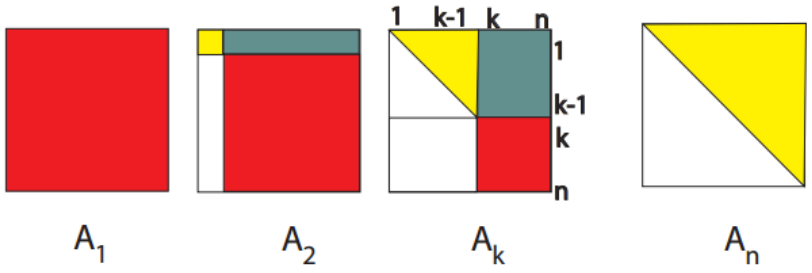


Figure 4.3: Schematic of Gaussian elimination [33].

The system is then solved using a backwards algorithm, starting at the bottom and working up the system. It can be shown that the Gaussian elimination of a matrix A leads to an LU *factorization* of A, so $A = LU$ where L is a lower triangular matrix and U is an upper triangular matrix [33]. Assuming that A is an $N \times N$ matrix, Gaussian elimination may require up to $\mathcal{O}(N^3)$ arithmetic operations and up to $\mathcal{O}(N^2)$ storage spaces. When the system becomes large this may become problematic.

4.4.2 Iterative methods for linear systems

Iterative methods are much more efficient in both arithmetic operations and storage requirements, in that both are $\mathcal{O}(N)$ [32]. Iterative methods begin with an initial guess x_0 to the solution x. Then a sequence of approximations $\{x_k\}$ are computed, hopefully in such a way that x_k converges towards x [11]. One family of iterative methods are *Krylov* methods.

Definition 4.1 (Krylov subspace method (adapted from [41]))**.** A Krylov subspace method is a method that seeks an approximate solution x_m from an affine subspace $x_0 + \mathcal{K}_m$ of dimension m by imposing the condition

$$b - Ax_m \perp \mathcal{L}_m,$$

where $\mathcal{L}_m$ is another subspace of dimension m.

Here, $\mathcal{K}_m$ is the Krylov subspace

$$\mathcal{K}_m(A, r_0) = span\{r_0, Ar_0, A^2r_0, ..., A^{m-1}r_0\},$$

where $r_0 = b - Ax_0$. Different versions of Krylov methods arise from different choices of $\mathcal{L}_m$ and different choices of *preconditioners*.

One such Krylov subspace method is the generalized minimal residual method (GMRES), which works when solving *non-symmetric* systems [11] and will be used in the simulations of the response of the spinal cord. For a detailed review of iterative methods, Krylov methods and GMRES, the reader is referred to [11, 41].

GMRES may require a *preconditioner*. A preconditioner modifies the linear system $Ax = b$ in such a way that it becomes easier to solve by an iterative solver, but does not change the system mathematically. Thus a preconditioner B may be applied to the linear system such that

$$BAx = Bb.$$

According to [32], an incomplete LU factorization (ILU) preconditioner, or an algebraic multigrid (AMG) preconditioner may be suitable for use with GMRES, thus both will be tested in the implementation. For details on ILU and AMG preconditioners, see for example the texts by Saad [41] and Stüben [51] respectively.

Chapter 5

Discretization of the elasticity and viscoelasticity equations

The previous chapter contained a review of the tools needed to solve the elasticity and viscoelasticity problems numerically. In this chapter those tools are applied, and the variational formulation for the general elasticity problem is developed. This is followed by a consideration of the constitutive relationships and how their respective variational formulations appear. In particular, the linear viscoelasticity problem is addressed in detail due to the temporal integral in the SLS model.

A computationally efficient scheme for computing the temporal integral is presented and tested. Boundary conditions are discussed, and further constraints are added on the boundary to make the implementation stable. Finally, a review of the implementations of the numerical schemes is given.

5.1 Continuous variational formulations

This section deals with developing variational formulations for the linear elasticity and linear viscoelasticity problems. First the general variational formulation for the governing equation (3.4a) is obtained, followed by considerations of the constitutive relationships, showing how they factor into the variational formulation.

5.1.1 Variational formulation – Governing equations

Multiplying (3.4a) with a test function v and integrating over the domain gives:

$$\int_\Omega -(\nabla \cdot \sigma) \cdot v \; d\Omega = \int_\Omega f \cdot v \; d\Omega, \tag{5.1}$$

where $\sigma = \sigma(u(x,t))$. Integrating by parts and inserting the boundary condition (3.4c) gives

$$\int_\Omega \sigma : \nabla v \; d\Omega - \int_{\Gamma_N} g \cdot v \; d\Gamma = \int_\Omega f \cdot v \; d\Omega \tag{5.2}$$

The general variational form for the governing equations thus becomes:

Find $u \in V$ such that

$$a(u,v) = L(v), \quad \forall \, v \in \hat{V}, \tag{5.3}$$

where

$$a(u,v) = \int_\Omega \sigma : \nabla v \; d\Omega, \tag{5.4}$$

$$L(v) = \int_\Omega f \cdot v \; d\Omega + \int_{\Gamma_N} g \cdot v \; d\Gamma. \tag{5.5}$$

The spaces V and $\hat{V}$ are defined as in (4.5) and (4.6) respectively.

In the following sections, the variational formulation (5.3) is developed in more detail, giving specific variational formulations for the two different constitutive relationships we are working with.

5.1.2 Variational formulation – linear elasticity

First, the variational formulation for the linear elasticity problem is developed. Recall that the constitutive (or stress-strain) relationship for linear elasticity for an isotropic material is given by

$$\sigma = 2\mu\varepsilon + \lambda \mathrm{tr}(\varepsilon) I.$$

This is inserted into (5.3), giving

$$\underbrace{\int_\Omega 2\mu\varepsilon : \nabla v \; d\Omega + \int_\Omega \lambda \mathrm{tr}(\varepsilon) I : \nabla v \; d\Omega}_{a(u,v)} = L(v), \;\; \forall \, v \in \hat{V}. \tag{5.6}$$

This is really no different from the variational formulation for the governing equations. Moving on to linear viscoelasticity and the SLS model, more work is required to develop a variational formulation.

5.1.3 Variational formulation – linear viscoelasticity

Recall that the constitutive relationship for the SLS model, is given by

$$\sigma(t) = D(0)\varepsilon(t) - \int_0^t D_s(t-s)\varepsilon(s) \; ds,$$

where $\varepsilon(t) = \varepsilon(u(x,t))$. Inserting this into (5.3) results in

$$\underbrace{\int_\Omega D(0)\varepsilon(t) : \nabla v \; d\Omega - \int_\Omega \int_0^t D_s(t-s)\varepsilon(s) \; ds : \nabla v \; d\Omega}_{a(u,v)} = L(v) \;\; \forall \, v \in \hat{V}. \tag{5.7}$$

The bilinear form $a(u,v)$ is now split into two parts, $b(u,v)$ and $c(u,v)$, allowing the temporal integral to be dealt with separately. Defining

$$b(t,t;u,v) = \int_\Omega D(0)\varepsilon(t) : \nabla v \; d\Omega, \tag{5.8a}$$

$$c(t,s;u,v) = \int_\Omega D_s(t-s)\varepsilon(s) : \nabla v \; d\Omega, \tag{5.8b}$$

the variational formulation for the linear viscoelasticity problem becomes: Find $u \in V$ such that

$$b(t,t;u,v) - \int_0^t c(t,s;u,v) \; ds = L(v), \;\; \forall \, v \in \hat{V}. \tag{5.9}$$

5.2 Discrete variational formulations

The problems (5.6) and (5.9) will now be discretized both in time and in space. We introduce u^h as the discrete finite element approximation to u, and consider the discrete time points $t_i,\ i = 0, ..., N$. Uniformly spaced points in time are used, thus $t_i = i\Delta t$ where $\Delta t = t_i - t_{i-1}$. The discrete solution at time level i is denoted by u_i^h.

5.2.1 Linear elasticity – discrete variational formulation

The discrete variational formulation for the linear elasticity problem is found simply by inserting u_i^h in place of u in the continuous variational formulation (5.3). Note that σ_i^h denotes $\sigma(u_i^h) = \sigma(u^h(t_i))$.

For each i, find $u_i^h \in V^h \subset V$ such that

$$a(u_i^h, v) = L(v), \quad \forall\, v \in \hat{V}^h, \tag{5.10}$$

where

$$a(u_i^h, v) = \int_\Omega \sigma_i^h : \nabla v \; d\Omega \tag{5.11a}$$

$$L(v) = \int_\Omega f_i \cdot v \; d\Omega + \int_{\Gamma_N} g_i \cdot v \; d\Gamma \tag{5.11b}$$

Seeing as the governing equations are quasi-static, the solution itself will not be time dependent. It is the values of f_i and g_i (which can both vary in time) that can affect the solution u_i^h. Thus, if f and g are constant in time, then the solution u_i^h will also be constant, while if f and g vary in time, the solution u_i^h will also vary in time.

5.2.2 Linear viscoelasticity – discrete variational formulation

As with the linear elasticity case above, the discrete variational formulation for the linear viscoelasticity case may be found by inserting u_i^h into (5.9):

Find $u_i^h \in V^h$ such that

$$b(t_i, t_i; u_i^h, v) - \int_0^{t_i} c(t_i, s; u^h(s), v)\ ds = L(v),\ \ \forall\, v \in \hat{V}^h, \tag{5.12}$$

where

$$b(t_i, t_i; u_i^h, v) = \int_\Omega D(0)\varepsilon_i^h : \nabla v\ d\Omega, \tag{5.13a}$$

$$c(t_i, s; u^h(s), v) = \int_\Omega D_s(t_i - s)\varepsilon^h(s) : \nabla v\ d\Omega \tag{5.13b}$$

Here, (5.12) is still continuous in time. To discretize it completely, the temporal integral may be approximated by a numerical quadrature rule, giving a discrete equation on the form

$$b(t_i, t_i; u_i^h, v) - \sum_{j=0}^{i} \bar{\omega}_{ij} c(t_i, t_j; u_j^h, v) = L(v), \tag{5.14}$$

where $\bar{\omega}_{ij}$ is a weight arising from a weighted quadrature rule. The *trapezoidal rule* is a simple example of such a quadrature rule (see section 4.2). Applying this rule to (5.14) gives

$$b(t_i, t_i; u_i^h, v) - \frac{\Delta t}{2} \sum_{j=1}^{i} [c(t_i, t_j; u_j^h, v) + c(t_i, t_{j-1}; u_{j-1}^h, v)] = L(v) \tag{5.15}$$

Rewriting and simplifying gives the problem: For each i, find $u_i^h \in V_h$ such that

$$b(t_i, t_i; u_i^h, v) - \frac{\Delta t}{2} c(t_i, t_i; u_i^h, v) = L(v) + \frac{\Delta t}{2} c(t_i, t_0; u_0^h, v) + \Delta t \sum_{j=1}^{i-1} c(t_i, t_j; u_j^h, v) \tag{5.16}$$

for all $v \in \hat{V}_h$.

5.3 An efficient discrete scheme for viscoelasticity

When i becomes large, the sum-term on the right hand side of (5.16) will be a long sum that needs computing. Since the sum must be recomputed for every time step, this can become very time consuming. Fortunately, the sum term in (5.16) can be rewritten in terms of the sum from the previous time step and the solution from the previous time step. Note that the considerations here are for the 1D model. The extension to 2D/3D is trivial because the stress relaxation functions $\mu(t)$ and $\lambda(t)$ are on the same form as $D(t)$, and thus both include the factor $e^{-t/\tau}$.

Recall that for the SLS model in 1D, we have

$$D(t) = E_1 + E_2\, e^{-t/\tau}.$$

In the constitutive relationship (3.45), the derivative of D with respect to s, $D_s(t-s)$, is also required. Here, s is the variable being used for the time integration. The expression for this derivative is

$$D_s(t-s) = \frac{E_2}{\tau}\, e^{-(t-s)/\tau}. \tag{5.17}$$

Due to the nature of the exponential function, the sum term from (5.16) can be rewritten as a function of the sum from the previous time step.

The details are as follows:

$$\begin{aligned}\sum_{j=1}^{i-1} c(t_i, t_j; u_j^h, v) &= \sum_{j=1}^{i-1} \int_\Omega D_s(t_i - t_j)\varepsilon_j : \nabla v \; d\Omega \\ &= \sum_{j=1}^{i-1} \int_\Omega \frac{E2}{\tau}\, e^{-(t_i-t_j)/\tau}\varepsilon_j : \nabla v \; d\Omega. \end{aligned} \tag{5.18}$$

Here, the expression (5.17) for $D_s(t-s)$ has been inserted into the sum term on the right hand side of (5.16). Since

$$e^a\, e^b = \, e^{a+b},$$

the factor $e^{-(t_i-t_{i-1})/\tau}$ can be taken outside the sum in (5.18), giving

$$\sum_{j=1}^{i-1} c(t_i, t_j; u_j^h, v) = e^{-(t_i - t_{i-1})/\tau} \sum_{j=1}^{i-2} \int_\Omega \frac{E2}{\tau} e^{-(t_{i-1} - t_j)/\tau} \varepsilon_j : \nabla v \, d\Omega + \int_\Omega \frac{E2}{\tau} e^{-(t_i - t_{i-1})/\tau} \varepsilon_{i-1} : \nabla v \, d\Omega.$$

The sum term on the right hand side here is equal to the original sum term in (5.18), in the previous time step t_{i-1}. Changing this back to the abstract form results in

$$\sum_{j=1}^{i-1} c(t_i, t_j; u_j^h, v) = e^{-(t_i - t_{i-1})/\tau} \sum_{j=1}^{i-2} c(t_{i-1}, t_j; u_j^h, v) + c(t_i, t_{i-1}; u_{i-1}^h, v). \quad (5.19)$$

The sum for the new time step can thus be calculated using only the sum calculated in the previous time step, and the solution from the previous time step.

5.3.1 Analysis of the efficient scheme

It is expected that the use of the new, efficient sum (5.19) will give considerable speedup in the calulations of (5.16). A simple analysis will explain why. When using the regular trapezoidal rule, the integral is approximated by a sum, and for each new time level the sum must be calculated. Accordingly, at time level t_i, $\mathcal{O}(i)$ operations are required. To calculate the solution for N time levels requires $1 + 2 + \ldots + N = \mathcal{O}(N^2)$ operations.

In comparison, using the sum (5.19) means that at each time step, all that is required is *one multiplication* and *one addition*. Thus to calculate the solution for N time levels requires $2N = \mathcal{O}(N)$ operations in total. When N becomes large, this will make a huge difference. For example, imagine we have $N = 10^5$. With the efficient sum (5.19), $\mathcal{O}(10^5)$ operations are required, while for the trapezoidal sum, $\mathcal{O}(10^{10})$ (!) operations are required.

In addition to the actual computations being much faster, we can also save memory. When using the trapezoidal rule, every solution from previous time levels must be kept available in order to calculate a solution at a new time level. This will require $\mathcal{O}(N)$ storage locations in the memory, where N again denotes the total number of time levels. For large N this may cause problems. With the new sum (5.19), only the sum calculated at the previous time step, and the solution for the previous time step need to be kept available, reducing the memory requirements drastically.

5.4 Boundary conditions

Thus far, the Dirichlet BC has been treated as a general BC $u = u_D$, for simplicity. This is because the Dirichlet BC can be *strongly* enforced, in that the linear system that arises from the variational formulations (5.10) or (5.16) can be altered to incorporate the Dirichlet BCs. As per the discussion in section 3.2 on the Dirichlet BCs, $u_z = 0$ seems to be a natural choice.

However, using only the constraint $u_z = 0$ on the boundaries Γ_1 and Γ_2, as suggested in section 3.2 means that the spinal cord is free to move in any direction except for along the z-axis. The constraint $u_z = 0$ is not sufficient for obtaining even qualitatively correct solutions, except for in very special cases when the mesh is symmetric. More constraints are required, as the geometry is free to be translated and/or rotated.

Checking eigenvalues of the matrix A

An investigation into the eigenvalues of the matrix A for the linear elasticity problem is a useful tool in determining what happens with the geometry under different constraints / boundary conditions. Any eigenvalue with value zero (or close to zero) indicates an unconstrained degree of freedom in the solution. The eigenvalues are calculated for different combinations of BCs. The results of the eigenvalue tests are summarized in Table 5.1.

1. *No essential (Dirichlet) BCs at all:*

   With no Dirichlet BCs, the matrix A has 6 eigenvalues with value ($\approx$) zero. From plots of the corresponding eigenvectors, it appears that all the degrees of freedom are rotational.

2. $u = 0$ *at* Γ_1 *and* Γ_2*:*

   Using the original BC og holding $u = 0$ on the Γ_1 and Γ_2 results in all non-zero eigenvalues. Thus this constraint is sufficient for the all the degrees of freedom to be restrained. However, this is hardly a realistic BC as the spinal cord segment would not be allowed to compress / dilate on Γ_1 and Γ_2.

3. $u_z = 0$ *at* Γ_1 *and* Γ_2*:*

   Holding the $z-$component of u fast on Γ_1 and Γ_2 results in the matrix A having 3 eigenvectors with value ($\approx$) zero. Two of the corresponding eigenvectors appear to be rotational, while one appears to be translational.

4. *Weakly enforcing a no-rotation condition* ($u \cdot e_\theta = 0$) *at* Γ_1 *and* Γ_2*:*

Imposing a no-rotation condition on Γ_1 and Γ_2 (using Nitsche's method, details follow below) results in the matrix A having one eigenvalue with value ($\approx$) zero. The eigenvector appears to be translational in the z-direction.

5. *$u = 0$ at a point in Γ_1 and Γ_2:*

   Setting $u = 0$ at a point in Γ_1 and Γ_2 rather than on the whole of Γ_1 and Γ_2 is a condition which tethers the cord in place while still allowing it to contract/dilate along its entire length. Imposing this condition results in the matrix A having one eigenvalue with value ($\approx$) zero. The corresponding eigenvector appears to be rotational.

6. *Combining $u_z = 0$ at Γ_1 and Γ_2 with $u = 0$ at a point in Γ_1 and Γ_2:*

   The results are similar to BC 5 (above).

7. *Combining $u \cdot e_\theta = 0$ with $u_z = 0$ at Γ_1 and Γ_2:*

   Combining these two BC's results in the matrix A having all non-zero entries. It seems that this is the minimum condition we need.

BC	No. of $\approx$ zero eigenvalues		
	Rotation	Translation	Total
1	6	0	6
2	0	0	0
3	2	1	3
4	0	1	1
5	1	0	1
6	1	0	1
7	0	0	0

Table 5.1: Summary of findings when checking the eigenvalues of the matrix A.

5.4.1 Weakly enforcing a no-rotation BC using Nitsche's method

In order to make sure that the spinal cord is tethered and unable to rotate, we impose the following Diriclet BCs:

$$u_z = 0, \quad x \in \Gamma_{1,2}, \tag{5.20}$$

$$u \cdot e_\theta = 0, \quad x \in \Gamma_{1,2}, \tag{5.21}$$

$$u = 0, \quad x \in \Gamma_{3,4}, \tag{5.22}$$

where, as before, the boundaries Γ_1 and Γ_2 represent the entire top and bottom surface of the cord geometry respectively, while Γ_3 and Γ_4 are introduced to represent a point in the top and bottom surface respectively, see Figure 5.1.

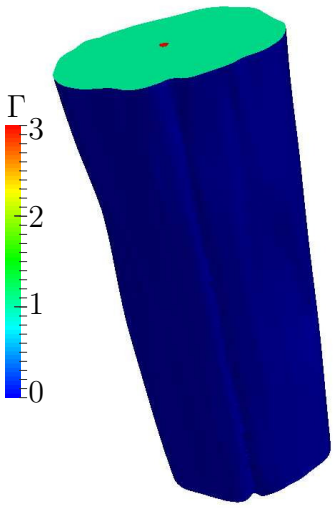


Figure 5.1: Colour schematic of the different boundaries.

The problem to be solved is now

$$-\nabla \cdot \sigma = f, \text{ in } \Omega \tag{5.23a}$$
$$u_z = 0, \text{ on } \Gamma_{1,2}, \tag{5.23b}$$
$$u \cdot e_\theta = u_0 \cdot e_\theta = 0, \text{ on } \Gamma_{1,2}, \tag{5.23c}$$
$$u = 0, \text{ on } \Gamma_{3,4}, \tag{5.23d}$$
$$\sigma \cdot n = g, \text{ on } \Gamma_N, \tag{5.23e}$$

Using the method outlined in [50] we define the vector e_θ as a unit vector for rotation in the xy-plane ,

$$e_\theta = \begin{pmatrix} \frac{-(y_0+y)}{r} \\ \frac{x_0+x}{r} \\ 0 \end{pmatrix} \tag{5.24}$$

where $r = \sqrt{(x_0+x)^2 + (y_0+y)^2}$. The values x_0 and y_0 depend on the geometry of the mesh.

The BCs in (5.23) are now handled one by one. First of all, the BC (5.23e) is the same as before, and will appear in the variational formulation, exactly as before. Next, the BCs (5.23b) and (5.23d) can be enforced *strongly* and thus no changes need to be made to the variational forms on account of these. The BC (5.23c), however, can not be enforced strongly and must instead be enforced weakly. This is done via Nitsche's method. An overview of the use of Nitsche's method on general problems can be found in [26]. The following result adapted from [26] for weakly enforcing a Dirichlet BC will be used:

Theorem 5.1. *Nitsche's method applied to the Dirichlet problem*

$$\begin{aligned} -\Delta u &= f \ \ in\ \Omega \\ u &= u_0 \ \ on\ \Gamma \end{aligned}$$

results in the variational form

$$\begin{aligned} &(\nabla u_h, \nabla v)_\Omega - \left\langle \frac{\partial u_h}{\partial n}, v \right\rangle_\Gamma - \left\langle u_h, \frac{\partial v}{\partial n} \right\rangle_\Gamma + \sum_{E \in \mathcal{G}_h} \frac{1}{\gamma h_E} \langle u_h, v \rangle_E \\ &= (f, v)_\Omega - \left\langle u_0, \frac{\partial v}{\partial n} \right\rangle_\Gamma + \sum_{E \in \mathcal{G}_h} \frac{1}{\gamma h_E} \langle u_0, v \rangle_E \end{aligned}$$

(Recall the variational form (4.7) for the Poisson equation where the Dirichlet BC is implemented strongly.)

Adapting Theorem 5.1 to the variational formulation (5.3) results in the problem:

Find $u \in V$ such that

$$\begin{aligned} &\int_\Omega \sigma(u) : \nabla v \ d\Omega + \frac{\gamma}{h_E} \int_{\Gamma_{1,2}} (u \cdot e_\theta)(v \cdot e_\theta) \ d\Gamma \\ &\quad - \int_{\Gamma_{1,2}} (\sigma(u) \cdot n \cdot e_\theta)(v \cdot e_\theta) \ d\Gamma - \int_{\Gamma_{1,2}} (\sigma(v) \cdot n \cdot e_\theta)(u \cdot e_\theta) \ d\Gamma \\ &= \int_\Omega f \cdot v \ d\Omega + \int_{\Gamma_N} g \cdot v \ d\Gamma + \frac{\gamma}{h_E} \int_{\Gamma_{1,2}} (u \cdot e_\theta)(v \cdot e_\theta) \ d\Gamma \\ &\quad - \int_{\Gamma_{1,2}} (\sigma(v) \cdot n \cdot e_\theta), (u_0 \cdot e_\theta) \ d\Gamma, \ \ \forall\, v \in \hat{V}, \end{aligned} \tag{5.25}$$

where $\gamma > 0$ is a stabilization parameter. Deriving the specific variational formulations for the two constitutive relationships follows the exact same procedure as in section 5.2, and the details are not shown here.

5.5 Implementation

This section contains a review of the implementation of the linear elasticity problem and the linear viscoelasticity problem with the SLS model. An overview of the language used and the code structure are presented, followed by a detailed review of some of the important functions that are implemented. Finally, details of usage of the implementation are provided. The entire code for the implementation is published on Bitbucket,
`https://bitbucket.org/nkylstad/master_thesis_src`.

5.5.1 Programming language and software

The code is, in its entirety, written in Python. Because implementing the large framework of the finite element method would be a thesis unto itself, and because there exist implementations of this already, finite element software compatible with Python was used. The chosen software was FEniCS, "a collection of free software with an extensive list of features for automated, efficient solution of differential equations" [1]. FEniCS uses a tool called UFL (Unified Form Language) which allows the user to write the code almost exactly like mathematical formula. See Appendix B for an example on the use of FEniCS.

The use of the finite element method typically results in a matrix equation on the form

$$AU = b,$$

see section 4.1 for an example. FEniCS calls on external libraries to solve these linear (or sometimes nonlinear) algebra problems. The linear algebra backend used in the implementation is PETSc. The direct and iterative (Krylov) solvers mentioned in section 4.4 are implemented in PETSc.

5.5.2 The code structure

As mentioned above, the code is written entirely in Python. Seeing as the linear elasticity problem and the linear viscoelasticity problem share some attributes,

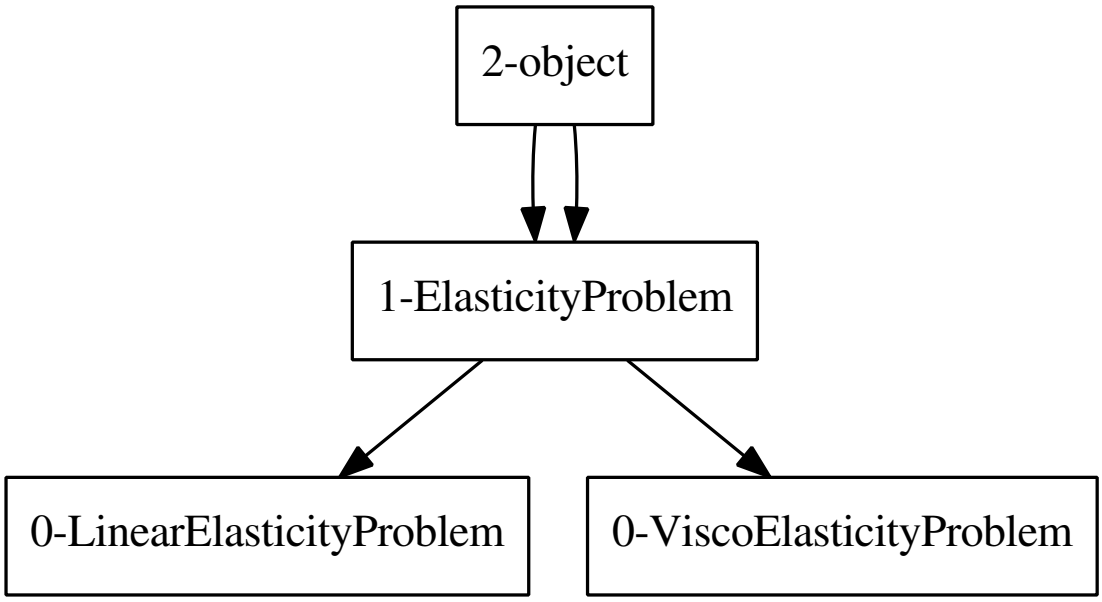


Figure 5.2: Class hierarchy for implementation

such as BCs, an object oriented approach was chosen. The class hierarchy used can be seen in Figure 5.2.

Superclass – `ElasticityProblem`

The superclass `ElasticityProblem` contains all the methods that the linear elasticity and linear viscoelasticity problems have in common, such as setting up the BCs, and the `solver`-method. Seeing as the `solver`-method is central in obtaining a solution, key parts are shown below.

```
def solver(self, i, a=None, L=None, matrix=False,
           boundary_parts=None, bcs=None):

    # Assemble system:
    if matrix:
        self.calculate_vector_b(i)
        A = self.A
        b = self.b
    else:
        A = assemble(a, exterior_facet_domains=boundary_parts)
        b = assemble(L, exterior_facet_domains=boundary_parts)

    # Apply boundary conditions:
    for bc in bcs:
        bc.apply(A, b)

    # Solve system:
```

```
    u = Function(self.V)
    if self.direct:
        # Use direct (LU) solver
        solve(A, u.vector(), b)
    else:
        # Use iterative (Krylov) solver
        solve(A, u.vector(), b, "gmres", "amg")

    return u
```

Code 5.1 : `ElasticityProblem.solver`

The `solver`-method obtains the stiffness matrix A and the RHS-vector b from methods implemented in the subclasses. The BCs are then applied to the system, and the system is solved using a PETSc *linear algebra solver*. Note that the direct LU solver is the default with PETSc. Specifying `(..., "gmres", "ilu")` or `(..."gmres", "amg")` tells PETSc to use the Krylov method GMRES with the ILU and AMG preconditioners respectively (see section 4.4).

In order to make the subclasses as similar as possible, they both implement the following methods:

- `form_a`
- `calculate_vector_b`

Details of the implementation for the two subclasses are shown below.

Subclass – `LinearElasticityProblem`

The subclass `LinearElasticityProblem` contains the methods and variables that only apply to the linear elasticity problem. Methods of note are

- `sigma` – calculates σ using the constitutive relationship (3.8):

```
def sigma(self,u, i=None):
  eps = self.eps
  mu, lmbda = self.mu, self.lmbda
  sigma = 2*mu*eps(u)+ lmbda*tr(eps(u))*Identity(self.dim)
  return sigma
```

Code 5.2 : `LinearElasticityProblem.sigma`

- `form_a` – calculates the matrix A from the bilinear form $a(u, v)$ in (5.6):

```
def form_a(self, i=0):
  self.a = inner(self.sigma(u), grad(v))*dx
  if theta_proj is not None:
    self.a = self.a \
```

```
      + gamma/h*inner(u, theta_proj)*inner(v, theta_proj)*(
   ds(1)+ds(2)) \
      - inner(self.sigma(v)*n, theta_proj)*inner(u,
   theta_proj)*(ds(1)+ds(2)) \
      - inner(self.sigma(u)*n, theta_proj)*inner(v,
   theta_proj)*(ds(1)+ds(2))

  # Assemble matrix
  self.A = assemble(self.a, exterior_facet_domains=self.bp)
```

Code 5.3 : `LinearElasticityProblem.form_a`

and

- `calculate_vector_b` – calculates the vector b from the linear form $L(v)$ in (5.6):

```
def calculate_vector_b(self, i=0, f=None, g=None):
  self.L = dot(f,v)*dx + dot(-g*n,v)*ds
  if theta_proj is not None:
    self.L = self.L\
      + gamma/h*inner(no_rotation, theta_proj)*inner(v,
   theta_proj)*ds(1)\
      + gamma/h*inner(no_rotation, theta_proj)*inner(v,
   theta_proj)*ds(2)\
      -inner(no_rotation, theta_proj)*inner(self.sigma(v)*n,
    theta_proj)*ds(1) \
      -inner(no_rotation, theta_proj)*inner(self.sigma(v)*n,
    theta_proj)*ds(2)

  # Assemble RHS vector
  self.b = assemble(self.L, exterior_facet_domains=self.bp)
```

Code 5.4 : `LinearElasticityProblem.calculate_vector_b`

Subclass – `ViscoElasticityProblem`

The subclass `ViscoElasticityProblem` contains the methods and variables that only apply to the linear viscoelasticity problem. Methods to note are

- `sigma_LHS` – calculates the part of σ, given by (3.45), containing the unknown displacement u_i at time level i only, given by

$$\sigma_i^{LHS} = D(0)\varepsilon(t_i) - \frac{\Delta t}{2} D_s(0)\varepsilon(t_i) \tag{5.26}$$

```
def sigma_LHS(self, u, i):
    lhs = mu(0)*eps(u) + lmbda(0)*div(u)*I
    if i > 0:
```

```
        lhs = lhs - Constant(0.5*dt)*mu_s(0)*eps(u) -
            Constant(0.5*dt)*lmbda_s(0)*div(u)*I
    return lhs
```

Code 5.5 : `ViscoElasticityProblem.sigma_LHS`

- `sigma_RHS` –calculates the part of σ, given by (3.45), containing the known displacements from the previous time levels, given by

$$\sigma_i^{RHS} = \frac{\Delta t}{2} D_s(t_i - t_0)\varepsilon(t_0) + \Delta t \sum_{j=1}^{i-1} D_s(t_i - t_j)\varepsilon(t_j) \tag{5.27}$$

  The known part of *sigma* (from previous time steps) in the Nitsche boundary term

$$\int_0^t \int_{\Gamma_{1,2}} (\sigma(u(s)) \cdot n \cdot e_\theta)(v \cdot e_\theta)\, d\Gamma\, ds,$$

  are also calculated in this method.

```
def sigma_RHS(self, i):
    tmp_first = assemble(inner(Constant(0.5*dt)*(mu_s(i*dt)*
        eps(u0) + lmbda_s(i*dt)*div(u0)*self.I), grad(v))*dx,
                                exterior_facet_domains=bp)
    rhs = tmp_first.copy()
    if theta_proj is not None:
        tmp_first_n = assemble(inner(Constant(-0.5*dt)
          *(mu_s(i*dt)*eps(u0) + lmbda_s(i*dt)*div(u0)*self.
              I)*n, theta_proj)
          *inner(v, theta_proj)*(ds(1)+ds(2)),
              exterior_facet_domains=bp)
        rhs += tmp_first_n

    if i > 1:
        if i > 2:
            self.prev_sum *= exp(-dt/tau)
            tmp_sum = self.prev_sum.copy()
            tmp_sum *= dt
            rhs += tmp_sum
            if theta_proj is not None:
                self.prev_sum_n *= exp(-dt/tau)
                tmp_sum_n = self.prev_sum_n.copy()
                tmp_sum_n *= -dt
                rhs += tmp_sum_n

        end_term = assemble(inner((mu_s(dt)*eps(up) +
            lmbda_s(dt)*div(up)*self.I), grad(v))*dx,
            exterior_facet_domains=bp)
```

```
        tmp_end = end_term.copy()
        tmp_end *= dt
        rhs += tmp_end
        self.prev_sum += end_term
        if theta_proj is not None:
            end_term_n = assemble(inner((mu_s(dt)*eps(up) +
                lmbda_s(dt)*div(up)*self.I)*n, theta_proj)
                       *inner(v, theta_proj)*(ds(1)+ds(2)),
                           exterior_facet_domains=bp)
            tmp_end_n = end_term_n.copy()
            tmp_end_n *= -dt
            rhs += tmp_end_n
            self.prev_sum_n += end_term_n
    return rhs
```

Code 5.6 : `ViscoElasticityProblem.sigma_RHS`

Other methods of note are the methods

- `form_a` – similar to the corresponding method in `LinearElasticityProblem`, but uses the `sigma_LHS` instead, calculating (5.26).

```
def form_a(self, i=0):
  self.a = inner(sigma_LHS(u,i), grad(v))*dx
    if theta_proj is not None:
        h, gamma = self.h, self.gamma
        self.a = self.a \
            + Constant(gamma)/h*inner(u, theta_proj)*inner(v
                , theta_proj)*(ds(1)+ds(2)) \
            - inner((self.sigma_vD*eps(v))*n, theta_proj)*
                inner(u, theta_proj)*(ds(1)+ds(2)) \
            - inner((self.sigma_vL*div(v)*I)*n, theta_proj)*
                inner(u, theta_proj)*(ds(1)+ds(2))\
            - inner(sigma_LHS(u,i)*n, theta_proj)*inner(v,
                theta_proj)*(ds(1)+ds(2))
```

Code 5.7 : `ViscoElasticityProblem.form_a`

- `calculate_vector_b` – similar to the corresponding method in `LinearElasticityProblem`, and in addition uses the `sigma_RHS`-method, calculating (5.27).

```
def calculate_vector_b(self, i=0, f=None, g=None):
    # L(v):
    self.myform = dot(f,v)*dx + dot(-g*n,v)*ds

    if self.no_rotation is not None:
        if self.theta_proj is not None:
            self.myform += Constant(gamma)/h*inner(self.
                no_rotation, self.theta_proj)*inner(v, self.
                theta_proj)*(ds(1)+ds(2)) \
                        - inner(self.no_rotation, self.
                            theta_proj) \
```

```
                        *inner((self.sigma_vD*self.eps(v) +
                            self.sigma_vL*div(v)*self.I)*n,
                            self.theta_proj)*(ds(1)+ds(2))

    # Putting together b vector:
    self.b = assemble(self.myform, exterior_facet_domains=bp
        )
    if i > 0:
         self.b += sigma_RHS(i)
```

Code 5.8 : `ViscoElasticityProblem.calculate_vector_b`

The methods implemented in the classes `ElasticityProblem`, `LinearElasticityProblem` and `ViscoElasticityProblem` are used to run simulations. A Python script takes in arguments from the command line, treats these arguments, and calls on a function `compute`. This function is described in detail below.

The function `compute` sets up the problem to be solved. A parameter `prob_type` determines if the problem should be a linear elasticity problem or a linear viscoelasticity problem. Parameter values relevant to the model used are sent in and stored.

```
def compute(prob_type, Nx, pval, degree, dim, pres_file, dt=0.1,
            makeplot=False, T=0.0, rectangle=False, cord=False,
            cordval=None, save_sol=False, direct=False,
            simple=False, path=None, eigentester=False):

    # Set up problem
    if prob_type == 'viscoElast':
        problem = ViscoElasticityProblem(degree, dim, dt, T,
                          direct=direct)
        E1 = 0.84E3; E2 = 2.03E3; eta = 6.7
        C = 22.8/2
        problem.set_parameters(C, E1=E1, E2=E2, eta=eta)
    elif prob_type == 'linElast':
        problem = LinearElasticityProblem(degree, dim, dt, T,
                          direct=direct)
        E = 16e3
        problem.set_parameters(E=E, nu=0.479)
    else:
        print "Error. Please enter valid problem type."

    # Set up mesh, function space, functions etc.
    problem.setup(Nx, mesh=None, rectangle=rectangle, cord=cord,
       cordval=cordval)

    # Set up path for saving solutions and save all parameter
       values
    # used to file.
    if save_sol:
        problem.set_save_file(path)
        problem.save_parameters(Nx, pval, degree, dim, dt,
           makeplot,
                      T, cordval, direct, simple)

    Nt = problem.Nt
```

Code 5.9 : `compute_solutions.py (1)`

The mesh is set up using `problem.setup`, and the different parameters this function takes determine what geometry the mesh will have.

The Neumann BC, acting as the applied pressure, comes in two forms – *simple* - a constant value p over the entire geometry for $t \in [0, T/2]$, followed by $p = 0$ for $t \in (T/2, T]$. This form of the BC is for testing, to check if the solution behaves as exprected. The other form comes from reading pressure data, measured from a patient, from file. This pressure data is obtained from Støverud et al. [50].

```
    # Set up boundaries, Dirichlet BCs and NO-ROTATION condition.
    problem.set_bc()
    gamma = 2.0e3
    problem.set_no_rotation(gamma)

    # Set up body force
    f = Constant([0.0]*dim)

    # Set up boundary pressure
    if simple:
        p = np.zeros(Nt+1)
        p[:Nt/2.] = pval
        gval = Expression("p", p=float(p[0]))
    else:
        v_wave = 200.0 # cm/s
        tt, pres_heartbeat = problem.read_pressure_file(pres_file
            )
        shift_val = 0.0
        pres_spline = splrep(tt, pres_heartbeat-shift_val)
        gval = problem.applied_pres(v_wave, pres_spline, dim)

    problem.set_g(gval)
    problem.set_f(f)
```

Code 5.10 : `compute_solutions.py (2)`

A `for`-loop is used to solve for `Nt` time levels. For the viscoelasticity problem, the bilinear form $(a(u, v))$ is slightly different for the initial time level, at $t = 0$.

```
    # Special case for the first time step for viscoElast.
    # For linElast, this will just calculate the first time step
    # normally.
    problem.form_a(0)
    U0 = problem.solver(0, matrix=True)
    problem.update(U0, save_sol=save_sol)

    # Set up time loop for computation
    progress = Progress("Time-stepping...")

    if prob_type == 'viscoElast':
        # Update bilinear form to include all terms
        problem.form_a(1)

    for i in range(1, Nt+1):
        # Check progress
        print i, " of ", Nt-1
        progress.update(i*dt/T)

        # Update parameters
        if simple:
            problem.update_g(float(p[i]), pressure=True)
        else:
            problem.update_g(t[i], time=True, simple=simple)

        # Solve
        U = problem.solver(i, matrix=True)

        # Update solution, save the solution to file if save_sol
           is True.
        problem.update(U, i=i, save_sol=save_sol)
```

Code 5.11 : `compute_solutions.py (3)`

The solution at each time level may be saved to file for analysis at a later time.

Chapter 6

Results

The simulations documented in this chapter are separated into two categories:

(i) Verification of the implementations. This category uses a constant applied pressure to display properties of the solutions. Errors in the results from these simulation are analysed using the method of manufactured solutions.

(ii) Measuring the response of the spinal cord under realistic conditions. The simulation scenario is described in section 3.2. The response using both the linear elasticity and SLS models is documented.

As such, this chapter is divided into two sections. The first section aims at verifying that the implementations give the expected results. The second section aims at documenting the response of the spinal cord, and comparing results from the viscoelastic and purely elastic models. Note that the parameter choices discussed in section 3.6 are summarized at the beginning of this section.

6.1 Verification of the implementations

Before any actual simulations can be run, the implementation of the schemes must be tested to make sure everything is working correctly. A simple preliminary test is a visual one – do plots of the solution behave as expected? Following this visual test is a more mathematical test, checking whether the implementations give the expected solutions under given constraints. The testing process is automated with the use of scripts to run the tests. The tests are run on a unit square (2D) or unit cube (3D) geometry for simplicity.

6.1.1 Linear Elasticity

This section covers the testing and verification of the implementation for the linear elasticity case. The parameters used in the verification tests are listed in Table 6.1.

Parameter	Value
E	0.65kPa
ν	0.479
T	1.0s
Δt	0.005
p_0	2.0
mesh	Unit Square
h	8.84×10^{-2}*
degree	1*

Table 6.1: Parameters used in verification tests for the linear elasticity implementation.
* Unless otherwise stated

Visual test

A simple pressure simulation is done on a unit square geometry. The Neumann boundary condition, $\sigma \cdot n = g = -pn$, where p is a scalar, simulates the applied pressure.

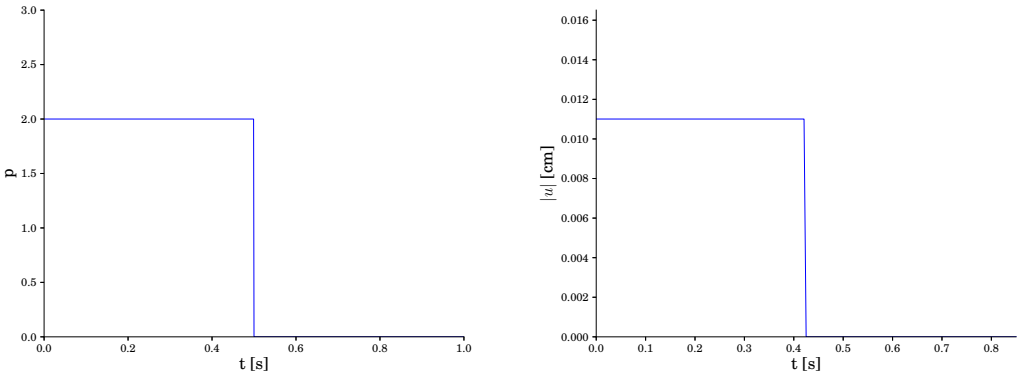
The pressure p is made to vary as

$$p = \begin{cases} p_0 & \text{if } 0, \leq t \leq 0.5, \\ 0, & \text{if } t > 0.5, \end{cases} \tag{6.1}$$

where p_0 is a constant. The pressure variation, as well as the displacement u in the point $(0.2, 0.5)$ in the mesh over time from the simulation are shown in Figure 6.1.

A quick look at Figure 6.1 is reassuring – the chosen point in the unit square reaches equilibrium instantaneously and remains at this equilibrium until the pressure is removed.

With confirmation that the implementation appears to be working , more rigorous tests are performed in order to determine that the solver actually delivers the correct results. To do this, a more mathematical test is required, with quantifiable results.



(a) Applied pressure **(b)** Displacement

Figure 6.1: (a) Pressure variation over time, and (b) Resulting displacement over time in a chosen point in the mesh from simple pressure simulation using linear elasticity solver on unit square geometry (2D). Parameters used are listed in Table 6.1.

Test using the method of manufactured solutions

See subsection 4.3.1 for an overview of the method of manufactured solutions.

The chosen solution u_e for testing the implementation is

$$u_e = \begin{pmatrix} (1-y)\, e^{px} \\ (1-y)\, e^{px} \end{pmatrix}. \tag{6.2}$$

The software *Mathematica* was used to calculate σ, g and f from (6.2), and the solution from the solver was tested against the exact solution (6.2). The $L2$-norm of the error,

$$||e||_{L^2} = ||u_e - u||_{L^2},$$

was calculated for different mesh resolutions. The results for a test in 2D are shown in Table 6.2.

Performing a similar test in 3D gives the results shown in Table 6.3.

The convergence rates of the errors are calculated using (4.21).

In the 2D case there is a clear second order convergence for elements of degree 1 and a clear third order convergence for elements of degree 2, which is exactly as expected. In the 3D case, the error displays tendencies towards second order convergence for elements of degree 1 and third order convergence for elements of degree 2, and it is assumed that the convergence in 3D is adequate. This assumption is acceptable because the implementation is nearly independent of the

	Degree 1		Degree 2	
h	e	rate	e	rate
2.50E-01	1.53E-01	–	2.33E-03	–
1.25E-01	4.46E-02	1.78	2.78E-04	3.07
6.25E-02	1.17E-02	1.93	3.41E-05	3.03
3.12E-02	2.97E-03	1.98	4.22E-06	3.01
1.56E-02	7.46E-04	1.99	5.26E-07	3.01
7.81E-03	1.87E-04	2.00	6.56E-08	3.00

Table 6.2: Errors in the numerical solution u for linear elasticity, when compared to a manufactured exact solution u_e, for elements of degrees 1 and 2 in 2D using direct LU solver. h represents the mesh resolution, and is defined as the maximum cell diameter in the mesh.

	Degree 1		Degree 2	
h	e	rate	e	rate
5.00E-01	5.47E-01	–	3.51E-02	–
2.50E-01	1.75E-01	1.64	2.84E-03	3.63
1.25E-01	5.05E-02	1.79	2.71E-04	3.39
8.33E-02	1.35E-02	1.91	2.99E-05	3.18
7.14E-02	8.72E-03	1.95	1.50E-05	3.09

Table 6.3: Errors in the numerical solution u for linear elasticity, when compared to a manufactured exact solution u_e, for elements of degrees 1 and 2 in 3D using direct LU solver. h represents the mesh resolution, and is defined as the maximum cell diameter in the mesh.

problem's dimension. Thus it would seem that the implementation of the linear elasticity problem gives the correct results for elements of degree 1 and 2, both in 2D and 3D.

6.1.2 Linear Viscoelasticity

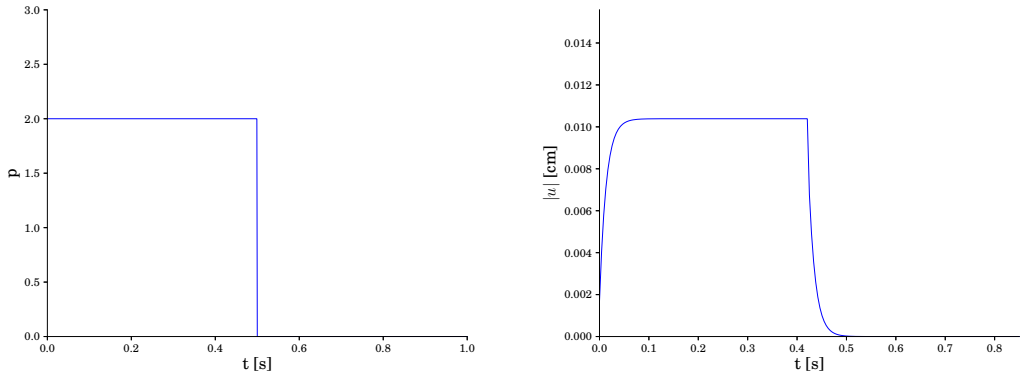
This section covers the testing and verification of the implementation of the linear viscoelasticity problem, using the SLS model. The parameter values that are used in the testing of the viscoelasticity solver are summarized in Table 6.4.

Parameter	Value
E_1	0.84kPa
E_1	2.1kPa
η	6.7Pas
C	22.8*
T	1.0s
Δt	0.005
p_0	2.0Pa/100
mesh	unit square
h	8.84×10^{-2}*
degree	1*

Table 6.4: Parameters used in verification tests for the linear viscoelasticity implementation.
* Unless otherwise stated

Visual test

As with the linear elasticity problem, a look at some plots to check if they look sensible may be useful. The solution is expected to display a viscoelastic creep effect when a constant pressure is applied. A simulation is run with the pressure variation given by (6.1). Again, the pressure variation as well as the magnitude of the displacement of the point $(0.2, 0.5)$ in the mesh over time, are shown in Figure 6.2.



(a) Applied pressure **(b)** Displacement

Figure 6.2: (a) Pressure variation over time, and (b) Resulting displacement over time in a chosen point in the mesh from simple pressure simulation using linear viscoelasticity solver on unit square geometry (2D). Parameters: $T = 1.0$s, $\Delta t = 0.005$, $E_1 = 0.84$kPa, $E_2 = 2.1$kPa and $\eta = 6.7$Pa s.

The shape of the plot looks excellent; under a constant applied pressure, an arbitrary point in the mesh displays an instantaneous elastic response followed by a gradual, bounded increase in displacement. Since the displacement is directly proportional to the strain, the chosen point thus displays the viscoelastic property of creep (see Figure 3.4). This suggests that the implementation of the viscoelasticity problem is working. Note that the viscoelastic range is short, as the point appears to have fully crept by approximately 0.05s.

The Method of Manufactured Solutions

See subsection 4.3.1 for an overview of the method of manufactured solutions.

The chosen solution u_e for testing the implementation is given by (6.2).

The software *Mathematica* was used to calculate σ, g and f from (6.2), and the solution u from the solver was tested against the exact solution (6.2). The $L2$-norm of the error,

$$||e||_{L^2} = ||u_e - u||_{L^2},$$

was calculated for different mesh resolutions. The results for a test in 2D for elements of degree 1 are shown in Table 6.5, while results for a test in 2D for elements of degree 2 are shown in Table 6.6. Note that $C = 0$ was used in the test with the manufactured solution.

$\Delta t \setminus h$	*3.54E-01*	*1.77E-01*	*8.84E-02*	*4.42E-02*	*2.21E-02*	*1.10E-02*
2.00E-02	**4.19E-03**	3.96E-03	5.64E-03	6.09E-03	**6.21E-03**	**6.23E-03**
1.00E-02	7.90E-03	**1.18E-03**	9.64E-04	1.38E-03	**1.49E-03**	**1.51E-03**
5.00E-03	8.91E-03	2.12E-03	**3.10E-04**	2.40E-04	**3.43E-04**	**3.68E-04**
2.50E-03	9.16E-03	2.37E-03	5.42E-04	**7.87E-05**	**5.98E-05**	**8.37E-05**
1.25E-03	9.22E-03	2.44E-03	6.07E-04	1.37E-04	**1.98E-05**	-*
1.00E-04	9.24E-03	2.46E-03	6.29E-04	1.58E-04	**3.95E-05**	-*
5.00E-05	**9.24E-03**	**2.46E-03**	**6.29E-04**	**1.58E-04**	**3.96E-05**	-*

Table 6.5: Errors in the numerical solution for linear viscoelasticity (SLS model), when compared to a manufactured exact solution, for elements of degree 1 in 2D using direct LU solver. h represents the mesh resolution, and is defined as the maximum cell diameter in the mesh. Δt represents the time step size.
* Note that these error values are missing due to the large time requirements for the computations.

The expected convergence rate in space for the error is 2 for elements of degree 1, and 3 for elements of degree 2. The expected convergence rate in time is 2. See section 4.3 for a review of the errors in the numerical solution.

$\Delta t \setminus h$	*3.54E-01*	*1.77E-01*	*8.84E-02*	*4.42E-02*	*2.21E-02*
2.00E-02	**6.25E-03**	6.25E-03	6.25E-03	6.25E-03	**6.25E-03**
1.00E-02	1.53E-03	**1.53E-03**	1.53E-03	1.53E-03	**1.53E-03**
5.00E-03	3.85E-04	3.80E-04	**3.79E-04**	3.79E-04	**3.79E-04**
2.50E-03	1.06E-04	9.51E-05	9.47E-05	**9.47E-05**	**9.47E-05**
1.25E-03	4.61E-05	2.45E-05	2.37E-05	2.37E-05	**2.37E-05**
1.00E-04	3.69E-05	5.13E-06	6.89E-07	1.75E-07	**1.52E-07**
5.00E-05	**3.69E-05**	**5.12E-06**	**6.70E-07**	**9.37E-08**	**3.94E-08**

Table 6.6: Errors in the numerical solution for linear viscoelasticity (SLS model), when compared to a manufactured exact solution, for elements of degree 2 in 2D using direct LU solver. h represents the mesh resolution, and is defined as the maximum cell diameter in the mesh. Δt represents the time step size.

There are some clear tendencies in the errors. We begin by looking at Table 6.5. In the bottom row, where Δt is very small, there is a clear second order convergence in space, just as in the linear elasticity case. Clearly, the Δt error is sufficiently small to display the behaviour of the spatial error. There is also a tendency towards a second order convergence on the diagonal. In the far right column, where h is small, there is a second order convergence in time to begin with. It is possible that an even finer mesh in space is required to fully capture the behaviour of the error in time; however, due to the very long computing times this would require, it is not prioritized, and it is assumed that the error has adequate convergence in time.

In Table 6.6, there is a second order convergence on the diagonal. There is also a clear second order convergence in the rightmost column, as well as tendencies towards a third order convergence in the bottom row, corresponding to the third order convergence we saw in the linear elasticity problem for elements of degree 2.

Overall it appears that the error in the solution for the linear viscoelasticity problem converges much as expected, and the implementation is assumed to be verified.

6.1.3 Testing the efficient scheme for the trapezoidal sum

To test whether the scheme (5.19) is more efficient than actually calculating the entire sum

$$\sum_{j=1}^{i-1} c(t_i, t_j; u_j, v),$$

a test is set up with a simple integral, on the same form as the history integral from the constitutive relationship (3.45), with known solution:

$$\int_0^t e^{-(t-s)} \sin(s)\ ds \tag{6.3}$$

We test the how well the new method converges towards the exact solution of (6.3) comparing the error in the new, possibly more efficient sum to the error in the trapezoidal sum. Several experiments are performed, with smaller and smaller Δt, meaning that the sum to be calculated becomes longer and longer. The results of this test are found in Table 6.7.

The time taken to calculate the longest sum ($\Delta t = 7.81E-04$) for the two methods is also compared, to see if there is any real computational speedup. The results from this test are shown in Table 6.8.

	Trapezoidal		Efficient		
Δt	e_h	rate	e_h	rate	Difference
1.00E-01	7.938E-04	–	7.938E-04	–	0.0
5.00E-02	1.735E-04	2.19	1.735E-04	2.19	2.07E-17
2.50E-02	4.067E-05	2.09	4.067E-05	2.09	4.76E-18
1.25E-02	9.852E-06	2.05	9.852E-06	2.05	4.67E-17
6.25E-03	2.425E-06	2.02	2.425E-06	2.02	8.34E-17
3.13E-03	6.015E-07	2.01	6.015E-07	2.01	3.55E-18
1.56E-03	1.498E-07	2.01	1.498E-07	2.01	9.79E-16
7.81E-04	3.738E-08	2.00	3.738E-08	2.00	3.95E-16

Table 6.7: Comparing errors in the solution for trapezoidal sum and efficient sum (5.19).

	Time taken (s)
Trapezoidal	1.294
Efficient	0.017
Speedup	74.6

Table 6.8: Comparing time taken to obtain solution when using trapezoidal sum and efficient sum (5.19).

From Table 6.7, it is clear that the error in both methods have the expected convergence rate of 2 (see section 4.3). Note that the errors in both methods are

very close to each other, and in fact appear identical in Table 6.7. This is expected, as the only difference in the efficient sum (5.19) from the regular trapezoidal sum is that it exploits a property of the exponential function. Therefore the two sums should be mathematically equal, and yield the same results.

The results in Table 6.8 show that there is a significant speed-up (a factor $\approx$ 70) when using (5.19) to calculate the integral as compared to using the regular trapezoidal sum.

The code for these tests can be found in Appendix A.1.

6.1.4 Testing the solver on large systems

As stated, there are $\approx 1,150,000$ cells in the mesh, resulting in a very large system to be solved. As mentioned in section 4.4, iterative solvers perform better than direct solvers for large linear systems. The performance of iterative solvers vs. a direct solver for the linear system that arises from (5.10) is investigated. Let the system size be $N \times N$. The solvers are tested on larger and larger systems to see how this is handled. The results are displayed in Table 6.9.

SYSTEM SIZE	**Direct (LU)****	**GMRES, ILU**		**GMRES, AMG**	
N	Time (s)	Time (s)	Iterations	Time (s)	Iterations
2523	0.199	0.020	44	0.087	10
11661	1.233	0.179	98	0.504	11
21063	3.964	0.399	120	1.178	11
40593	11.96	1.277	208	2.581	11
58692	31.41	1.943	213	4.418	11
95754	69.15	3.480	238	7.910	12
128076	–*	6.694	325	12.408	13
605958	–*	–*	>10000	98.113	26

Table 6.9: Results from testing three different solvers on larger and larger linear systems.
* The solver failed.
** Number of iterations are not relevant for a direct solver.

From the results in Table 6.9, it is clear that the direct solver can not compete with the iterative solvers. It seems that the (GMRES, ILU) combination solves the system faster for smaller systems than the (GMRES, AMG) combination. However the number of iterations increases quite drastically with system size for (GMRES, ILU), while it remains relatively constant for (GMRES, AMG). This problem becomes visible for the largest system tested – this is the size of the

system when elements of degree 1 are used with a mesh of an actual spinal cord – when the (GMRES, ILU) combination failed to converge in 10000 iterations. This is clearly a huge drawback, and the simulations henceforth will be run using an AMG preconditioner.

6.2 Simulation results

This section contains the results of the simulations using a spinal cord segment geometry, as outlined in section 3.2. Six different simulations are run; four using the SLS model and two using the linear elasticity model, all with different parameter choices. A summary of all the different model parameters that are used is shown in Table 6.10.

	Constitutive relationship	E_1(Pa)	E_2(Pa)	η(Pa s)	C	E(Pa)	ν
Model 1	SLS	0.84×10^3	2.03×10^3	6.7	22.8	–	0.479∗
Model 2	SLS	0.21×10^5	0.53×10^5	1.7×10^2	22.8	–	0.479∗
Model 3	SLS	0.84×10^3	2.03×10^3	13.4	22.8	–	0.479∗
Model 4	SLS	0.84×10^3	2.03×10^3	6.7	0	–	0.0∗
Model 5	Lin. elast	–	–	–	–	1.6×10^4	0.479
Model 6	Lin. elast	–	–	–	–	6.5×10^2	0.479

Table 6.10: Summary of the parameters to be used in simulations.
*Defined implcitly through C.

The other (default) parameters used in the simulations are listed in Table 6.11.

Parameter	Value
degree	1
T	0.85s
Δt	0.005
γ	2000*
x_0	1.5cm
y_0	0.7cm

Table 6.11: Other (default) parameters used in simulations.
* For Model 2, $\gamma = 1 \times 10^4$ was required.

In addition to the simulations defined by Models 1-6 and Table 6.11, simulations using Models 1 and 6 are run over four cycles ($T = 3.4$s).

This section is divided into two parts. The first part contains only simulation results from using the SLS model, while the second contains comparisons between results from the SLS model and the linear elasticity model. The results are displayed in such a way that two simulations are compared at a time. The comparisons consist of plots of the displacement patterns and magnitudes over the spinal cord geometry, as well as line plots, displaying the magnitude of the displacement of selected points in the geometry over time. Three points are chosen for these line plots; one point towards the top of the geometry, one point near the middle and one point near the bottom. The coordinates for the points are given in Table 6.12.

	Point coordinates		
	x	y	z
x_T (top)	2.0	0.56	3.3
x_M (middle)	2.0	0.56	1.7
x_B (bottom)	2.0	0.56	0.1

Table 6.12: Coordinates for points in the mesh where displacement magnitude is recorded over time.

In addition, one simulation is run using *Model 1* with $T = 3.4$s. A plot of the magnitude of the displacements in the points specified in Table 6.12 is included.

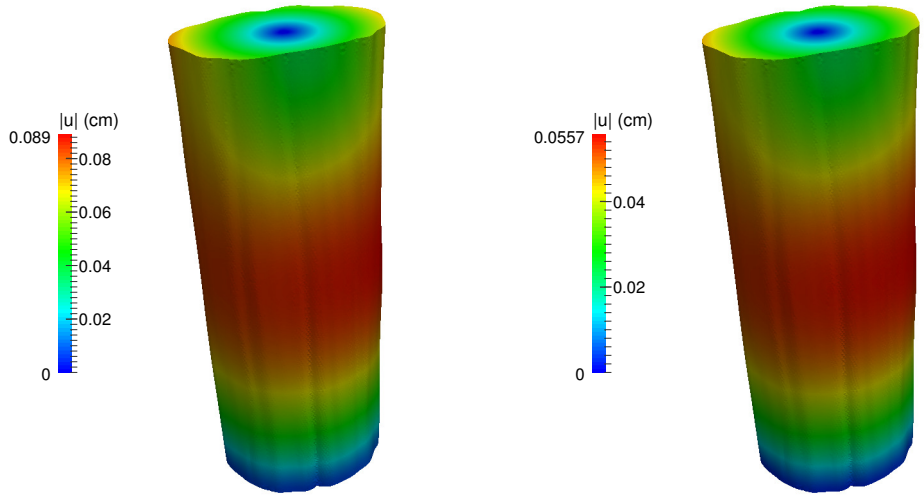
6.2.1 Viscoelastic response of the spinal cord

A comparison of the results of Models 1 and 3 can be seen in Figure 6.3, where η has been doubled from Model 1 to Model 3, while the remaining parameters are the same. The comparison in Figure 6.3 suggests that changing η does not have a large effect on the displacement pattern. Qualitatively the plots look similar. However, it should be noted that the magnitude of the displacements decreased noticeably by just doubling η.

Recording the magnitude of the displacement in the points specified by Table 6.12 for Models 1 and 3 result in the plots shown in Figure 6.4.

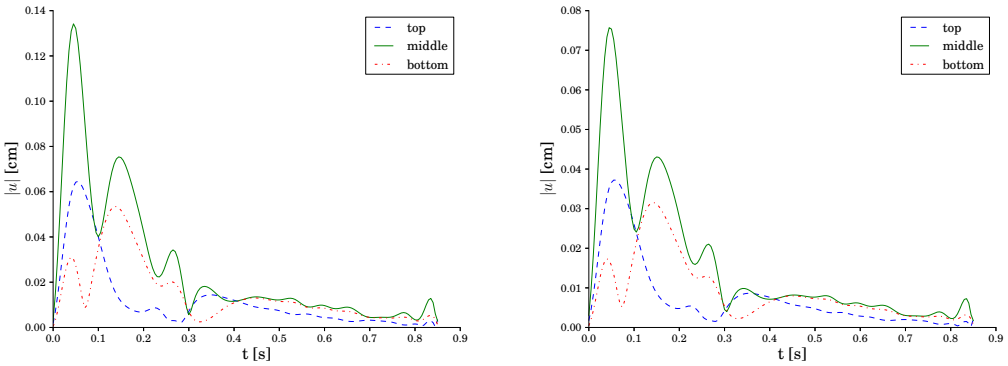
The plots in Figure 6.4 have very similar shapes; at first glance they appear identical. The magnitude of the displacement is the largest for the middle point x_M, while the displacements for the top and bottom points (x_T, x_B) have similar magnitudes, with peak displacements approximately half the value of the peak displacement of x_M.

Model 3 (with the higher value of η) has a smaller displacement magnitude in all the points than Model 1. This is to be expected considering the plots in Figure 6.3. An



(a) Model 1 **(b)** Model 3

Figure 6.3: Visual comparison of Model 1 and Model 3 at $t = 0.075$s. The displacement patterns are similar for Models 1 and 3, while the magnitudes of the displacement differ slightly.



(a) Model 1 **(b)** Model 3

Figure 6.4: Displacement magnitude over time for chosen points in the geometry for Model 1 and Model 3. The two plots show the similar qualitative behavior, but differ in magnitude.

observation can be made about the ratios between the instantaneous displacement at $t = 0$ and the peak displacement for Models 1 and 3. The magnitude of the displacement is largest in the middle point, thus this point is used for comparison. Since the values of E_1 and E_2 are the same for Models 1 and 3, the initial displacement at $t = 0$ is the same. Denoting u_0 as the instantaneous displacement at $t = 0$ and u_{max} as the peak displacement in $x_M s$, the ratios u_{max}/u_0 are shown

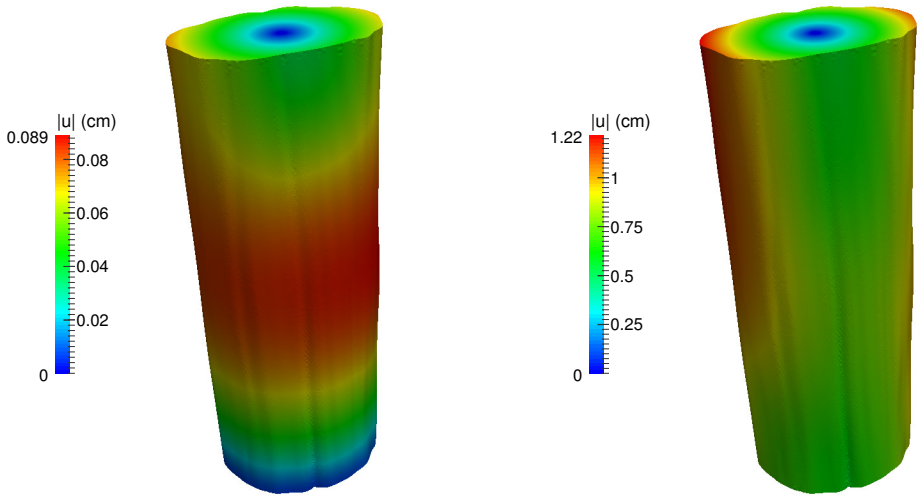
	u_{max}	u_0	$\frac{u_{max}}{u_0}$
Model 1	3.08×10^{-3}	1.34×10^{-1}	43.6
Model 3	3.08×10^{-3}	7.57×10^{-2}	24.6

Table 6.13: Comparing peak displacement of a point in the middle of the geometry over time for Model 1 and Model 3.

in Table 6.13.

From Table 6.13, it is clear that the parameter η has a pronounced effect on the peak displacement, as the ratio $\frac{u_{max}}{u_0}$ of Model 1 is nearly twice that of Model 3. As a result of this, the displacement magnitude over time appears slightly damped for Model 3 in comparison with Model 1. Both Model 1 and Model 3 reach peak displacement at $t = 0.045$s.

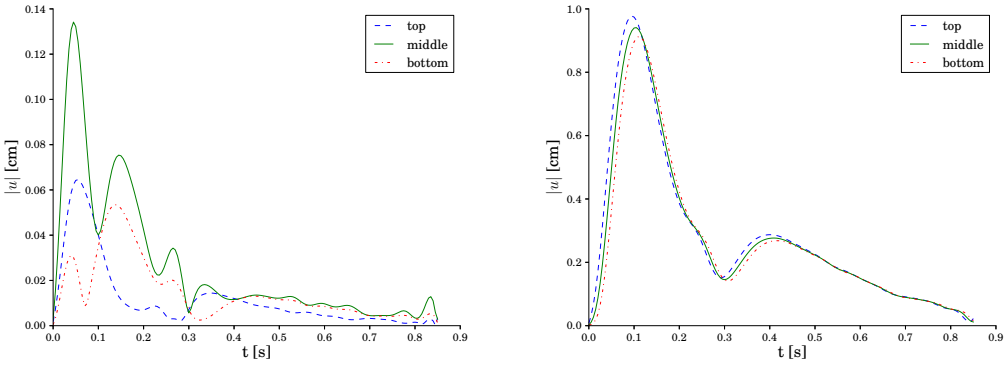
As explained in section 3.5, in order to get an accurate model in 2D and 3D, the simple model developed from a combination of springs and dashpots is expanded to account for the effect of compressibility. However, it is interesting to see if the effects of compressibility are of any significance. To that end, a comparison between Model 1 and Model 4 ($C = 0$) is shown in Figure 6.5.



(a) Model 1 **(b)** Model 4

Figure 6.5: Visual comparison of Model 1 and Model 4 at $t = 0.075$s. The displacement patterns are drastically different for Models 1 and 4, as are the magnitudes of the displacement.

From the comparison in Figure 6.5, it is clear that the effect of compressibility is very important, as the plots look very different. The maximum displacements at



(a) Model 1 **(b)** Model 4

Figure 6.6: Displacement magnitude over time for chosen points in the geometry for Model 1 and Model 4. The two plots differ both in qualitative behavior and in magnitude.

the time level depicted are focused mainly around the centre of the geometry for Model 1, while the maximum displacements for Model 4 are focused around the sides of the geometry.

Recording the magnitude of the displacement in the points specified by Table 6.12 for Models 1 and 4 result in the plots shown in Figure 6.4.

For Model 1, as mentioned, the displacements in the points x_T (top) and x_B (bottom) are smaller than the displacement in the point x_M (middle). This is not the case for Model 4. The displacements in the points x_T, x_M and x_B are similar in magnitude, and the plots for the three points are similar in shape.

The magnitude of the displacements is one order of magnitude larger for Model 4 than for Model 1. A displacement in the order of 1cm is extremely large considering that the length of the spinal cord segment is approximately 3.4cm, and approximately 1.5cm at the widest. Clearly the compressibility effect is significant.

Following a point in the geometry for Model 1 over time for several cycles results in the plot shown in Figure 6.7.

Qualitatively, the displacement appears the same for each of the cycles. Denote the peak displacement in each cycle for the points x_T, x_M and x_B by $u_{T,max}, u_{M,max}$ and $u_{B,max}$ respectively. The peak displacements for the three points for each cycle are shown in Table 6.14.

There is a definite, but small difference in the values of the peak displacements over the four cycles in all three points, in the order of 0.1μm. The peaks for the point x_M (the point with the largest displacement) are equidistant, and occur

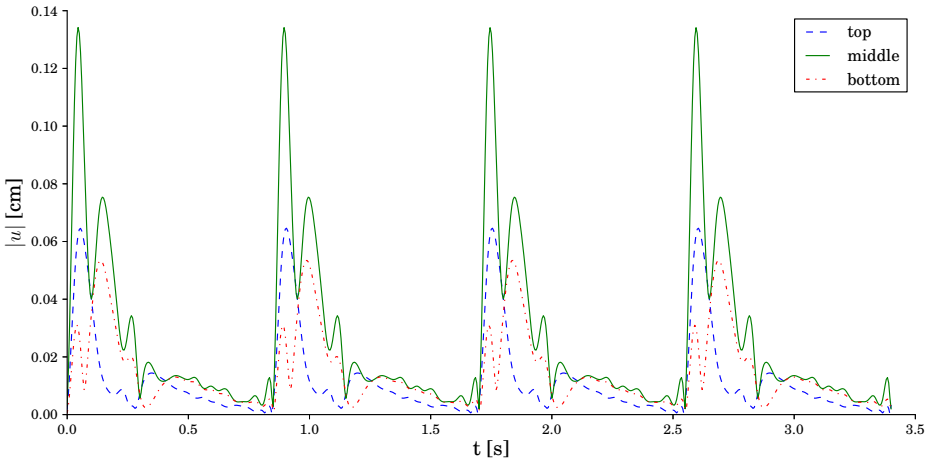


Figure 6.7: Displacement in the points specified in Table 6.12 over four cycles ($T = 3.4$s).

Cycle	$u_{T,max}$(cm)	$u_{M,max}$(cm)	$u_{B,max}$(cm)
1	0.064560	0.134180	0.0534309
2	0.064612	0.134151	0.0534311
3	0.064609	0.134119	0.0534312
4	0.064609	0.134119	0.0534312

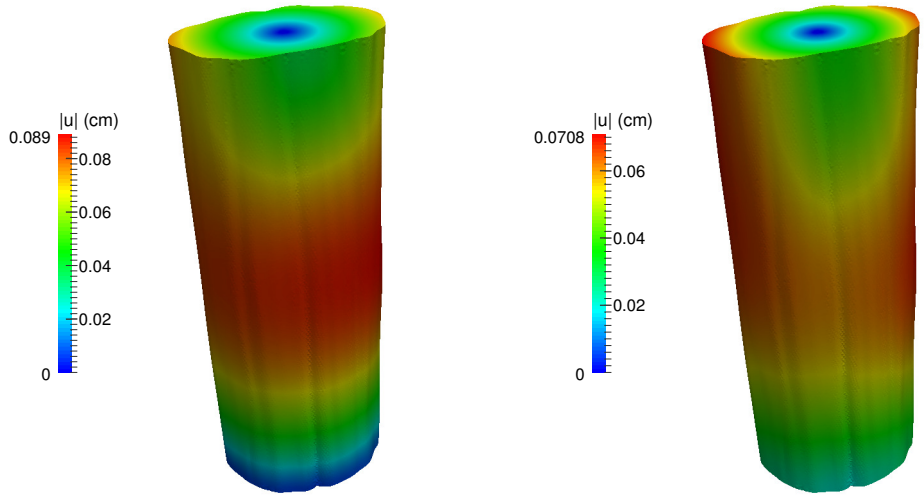
Table 6.14: Peak displacements for the points x_T, x_M and x_B for each cycle, using Model 1 with $T = 3.4$. The difference in the peak displacements over four cycles is in the order of 1×10^{-7}m for each of the points.

0.045s into each cycle.

6.2.2 Comparing the viscoelastic model with the purely elastic model

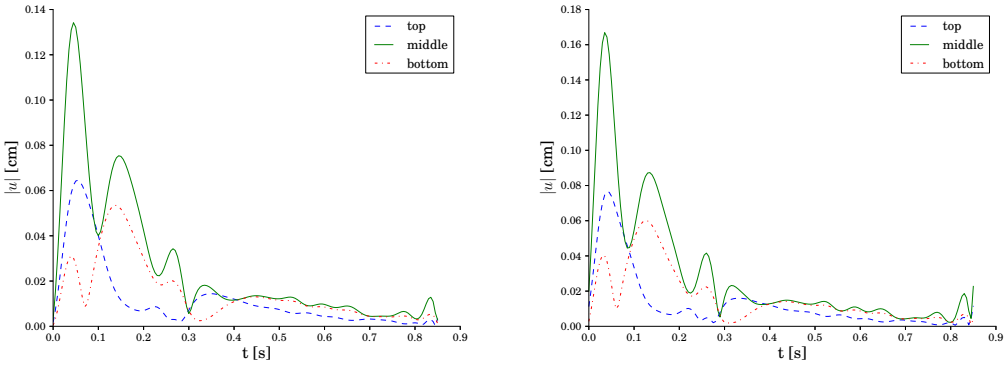
A comparison of the results of Models 1 and 6 can be seen in Figure 6.8. The parameters for the Model 6 have been computed in order to allow the displacement magnitudes to be compared.

From the comparison in Figure 6.8, the displacement patterns for Models 1 and 6 appear relatively similar. The only clear difference at this time level is that the largest displacements are focused slightly more along the sides of the geometry for Model 6, and more in the centre of the geometry for Model 1. Viewing the plots for the whole time series shows why the patterns are slightly different; Model 1



(a) Model 1 **(b)** Model 6

Figure 6.8: Visual comparison of Model 1 and Model 6 at $t = 0.075$s.



(a) Model 1 **(b)** Model 6

Figure 6.9: Displacement magnitude over time for a chosen point in the geometry for Model 1 and Model 6. The two curves show the similar qualitative behavior, but differ in magnitude.

has a slight lag in response when compared to Model 6. This can also be seen with plots of the points x_T, x_M and x_B over time for Models 1 and 6, as shown in Figure 6.9.

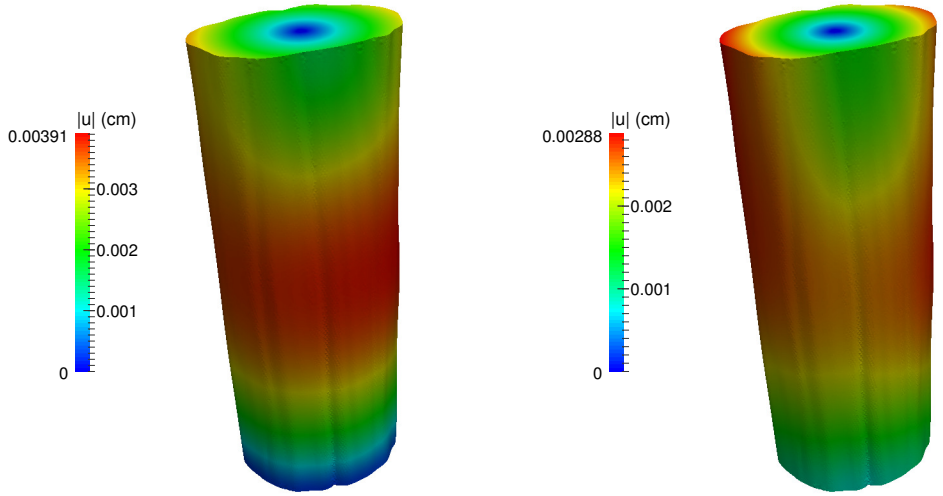
The suggested lag of Model 1 in comparison to Model 6 may be quantified by testing when the point x_T, x_M and x_B reach their peaks for the respective models. The results of such a test are shown in Table 6.15.

There is a clear lag of $10 - 15$ms overall for Model 1 when compared to Model 6.

	Reaches peak after (s)	
Point	Model 1	Model 6
x_T	0.055	0.04
x_M	0.045	0.035
x_B	0.14	0.125

Table 6.15: Time taken to reach peak displacement for the chosen points in Model 1 and Model 6.

A comparison of the results of Models 2 and 4 can be seen in Figure 6.10. The parameters for the Model 2 have been computed in order to allow the displacement magnitudes to be compared.

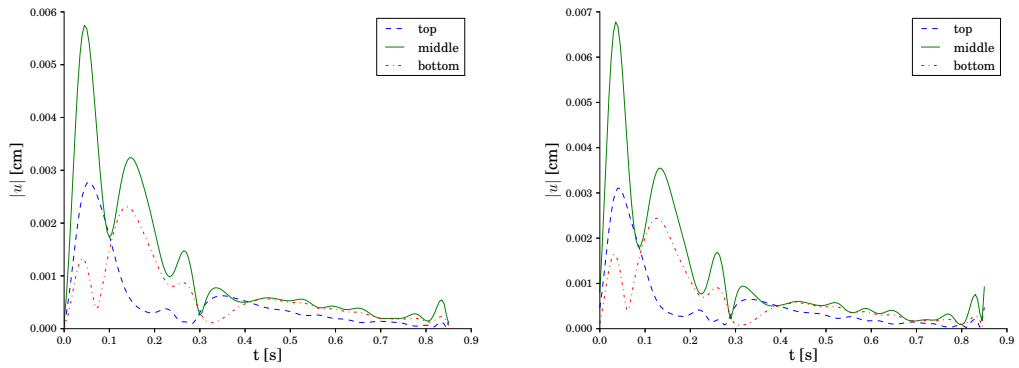


(a) Model 2 **(b)** Model 5

Figure 6.10: Visual comparison of Model 2 and Model 5 at $t = 0.075$s.

The results shown Figure 6.10 in are very similar to the results in Figure 6.8. The results in Figure 6.8a and Figure 6.10a show very similar displacement patterns; the same goes for the results in Figure 6.8b and Figure 6.10b. At the given time level, the largest displacements in Figure 6.10 (Models 2 and 5) are one order of magnitude smaller than the the largest displacements in Figure 6.8 (Models 1 and 6).

The results from following the chosen points over time for Models 2 and 5 are displayed in Figure 6.11. The plots display behaviour that is similar in shape to the plots in Figure 6.9, in that the plots in Figure 6.11b lag somewhat behind the plots in Figure 6.11a. The magnitude of the displacements shown in Figure 6.11



(a) Model 2 **(b)** Model 5

Figure 6.11: Displacement magnitude over time for a chosen point in the geometry for Model 2 and Model 5. The two curves show the similar qualitative behavior, but differ in magnitude.

are two orders of magnitude smaller than those shown in Figure 6.9.

Running a simulation for Model 6 over several cycles ($T = 3.4$s) results in a plot similar in shape to Figure 6.7. Measurements of the displacement magnitude of the peak at each cycle for the points x_T, x_M and x_B are shown Table 6.16.

Cycle	$u_{T,max}$(cm)	$u_{M,max}$(cm)	$u_{B,max}$(cm)
1	0.0765298	0.166881	0.0600790
2	0.0765298	0.166881	0.0600790
3	0.0765298	0.166881	0.0600790
4	0.0765298	0.166881	0.0600790

Table 6.16: Peak displacements for the points x_T, x_M and x_B for each cycle, using Model 6 with $T = 3.4$. There is no difference in the magnitude of the peak displacement for each cycle.

The results in Table 6.16 show that there is no difference in the magnitude of the displacement at each peak for the points x_T, x_M and x_B for Model 6. The peaks for the point x_B (the point with the largest displacement) occur equidistantly, at 0.035s into the cycle.

Chapter 7

Discussion

Simulations were run using six different models; four models used a viscoelastic constitutive relationship, and two models used a linear elastic constitutive relationship. The different models are summarized in Table 6.10. The greatest difference in magnitude in the displacement results was between Models 1 and 5. The greatest difference in displacement patterns and behaviour was between Models 1 and 4. The results are discussed in detail below.

The results showed displacements in the order of 1mm for the SLS model with default parameter values (Model 1), and displacements in the order of 70μm for the linear elasticity model with default parameter values (Model 5). Thus, with the default parameters for both models, Model 1 resulted in displacement magnitudes approximately 15 times larger than Model 5.

Doubling the viscosity parameter η from the SLS model with default values (Model 3) resulted in a decrease in displacement magnitude by a factor of approximately 1.6 when compared to Model 1. The ratio of the peak displacement to the initial displacement was approximately 1.8 times higher for Model 1 than for Model 3.

Neglecting the effects of compressibility (Model 4) resulted in displacement magnitudes one order of magnitude larger than when these effects were included (Model 1).

Running a simulation for several cycles with Model 1 showed that the peak displacement differed slightly (in the order of $10^{-7} - 10^{-8}$m) in the first three cycles; the difference between the peak displacements in cycles 3 and 4 was not visible. A similar test using linear elasticity (Model 6) did not show any difference between the peak displacements in the four cycles. The change in peak displacement became smaller and smaller for each cycle with Model 1. It is therefore anticipated that a simulation over more than four cycles will display the same changes in the peak displacement over the first three / four cycles, but that

the change in peak displacement will diminish with each cycle.

The behaviour of the results using the SLS model with default and calculated parameters (Models 1,2) displayed a lag of approximately 10ms when compared to the corresponding linear elasticity models (Models 5,6). This can be seen in Figure 6.9.

Parameter values

Changing parameter values in the models had some effect on the results. Scaling up/down the parameter values, for example from Model 1 to Model 2 or from Model 5 to Model 6 had a clear effect on the magnitude of the displacements, as can be seen in Figures 6.8-6.11.

Doubling η caused the displacement magnitude to decrease by a factor of ≈ 1.6 which is quite substantial. The displacement patterns in Figure 6.3 appear to be the same. However, viewing the plots for selected time steps over the time interval showed that while the displacement patterns were very similar, they were not identical at every time step. This is supported by the plots in Figure 6.4, where the plot in Figure 6.4b displays a more damped behaviour than the plot in Figure 6.4a. The choice of η was based on the MRE study by Klatt et al. [28]. The values obtained in that study are supported by values obtained in studies by Green et al. [21] and Sack et al. [42], although Green et al. [21] obtained values that were slightly higher. Reviews by Bilston [9] and Clarke [13] report values of the same magnitude, as well as higher values.

The choice of η, as well as the elastic parameters E_1 and E_2 are clearly important to the magnitude of the displacement for the viscoelastic model.

The parameter ν appears in the linear elasticity model, and also appears implicitly in the viscoelastic model. This parameter provides information about a materials *compressibility*. As mentioned, $\nu = 0$ means a material is completely compressible, while $\nu = 0.5$ means a material is completely incompressible. The effect of compressibility was investigated using Models 1 and 4, where Model 1 used the parameter value $C = 22.8$ to implicitly set $\nu \approx 0.479$, while Model 4 used $C = 0$ to set $\nu = 0$. The results, displayed in Figure 6.5 and Figure 6.6 suggest that the solution is sensitive with respect to the value of ν, and that the effects of compressibility must be included in the model to get believable results.

Overall, selecting appropriate parameter values, both for the SLS model and for the linear elasticity model is challenging due to the large number of different values reported in the literature. In Model 6, the Young's modulus was calculated in order to give similar displacement magnitudes for Models 1 and 6. The value for Young's modulus used in Model 6 is in the order of 1kPa, and is slightly below

the range found in the literature, see section 3.6. Thus it seems likely that a higher Young's modulus is more accurate for modelling the spinal cord.

Model 5 uses the same parameter values as Støverud et al. [50], with $E = 16$kPa. This seems to be a more realistic value considering what is reported in the literature, see subsection 3.6.1. However, using Model 5 results in displacements that are at least one order of magnitude smaller than with Model 1. The calculated parameter values used in Model 2 result in similar displacement magnitudes as Model 5, but these values are slightly above the range of values reported by for example Bilston [9] and Clarke [13]. Higher values typically arise from testing with higher strain rates and strains.

Considering the pronounced effect the parameter values had on the displacement magnitude in the results, as well as having some effect on the displacement patterns, there is a clear need for an established methodology for obtaining parameter values from observations This may allow for the development of standardized parameter values for the spinal cord.

Viscoelastic effect

The results show a small but clear effect of using a linear viscoelastic model, as compared to a linear elastic model, when simulating the response of the spinal cord under pressure. Plots of selected points in the geometry over time, as shown in for example Figure 6.9 and Figure 6.11 display a lag of approximately 10ms in the viscoelastic displacement when compared to the linearly elastic displacement.

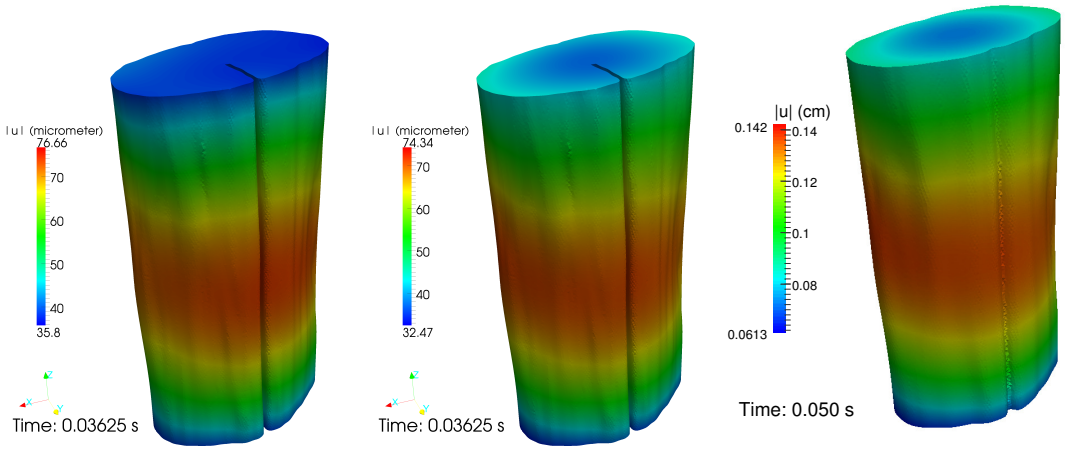
Running simulation using Model 1 over four cycles ($T = 3.4$s) and recording the magnitude of the displacements in the points specified in Table 6.12 results in the plot displayed in Figure 6.7. A measurement of the peak displacement in the points for each cycle, as listed in Table 6.14, shows that the peak displacement does vary for the first three cycles, by approximately 0.1μm. This does not occur in a similar simulation, using Model 6 and $T = 3.4$s, see Table 6.16.

However, it is unclear whether the viscoelastic effect is important in the context of syrinx formations. Seeing as the simulations using the SLS model display behaviour very similar to that of the simulations using the linear elasticity model, it is unlikely that the effect is significant.

Comparing viscoelastic reponse to poroelastic response

Using results obtained by Støverud et al. [50], we may compare the results using the SLS model to results using a poroelastic model. The comparison is shown in Figure 7.1. Note that the top and bottom 0.5cm have been discarded for the result

from Model 1, so that the geometries are comparable. Note also that the plot for Model 1 is at a different time level than the two other plots. This is because the time level for the plots in Figure 7.1a and Figure 7.1b is chosen because the displacement is at it's maximum at this time level. However, as shown above, Model 1 (and the other viscoelastic models) reaches peak displacement at a later time level.



(a) Poroelastic [50] **(b)** Linear elastic [50] **(c)** SLS, Model 1

Figure 7.1: Comparison of results from Støverud et al. [50] with viscoelastic results using Model 1. Note that the top and bottom 0.5cm have been cut from the geometry to display the same geometry as Støverud et al. [50].

From the visual comparison, there displacement pattern for Model 1 appears similar to the patterns obtained by Støverud et al. [50] using poroelastic and linear elastic models. The biggest difference between the results is the magnitude of the displacement. Note that the magnitude of the displacement for Model 5 is similar to that shown in Figure 7.1b at the corresponding time step.

The main difference between the viscoelastic and the poroelastic models is that the viscoelastic model models the spinal cord a *solid*, while the poroelastic model also takes into consideration fluid flow within the spinal cord.

Limitations

The most significant limitation in modelling the spinal cord mathematically is the lack of standardized parameter values. Because the range in values is large in the literature, choosing appropriate parameter values is challenging.

Because the geometry modelled is only a small part of the spinal cord, selecting appropriate boundary conditions is also a challenge. It is possible that restricting

the geometry in the axial direction at the top and bottom boundaries is not entirely accurate. The Neumann boundary condition simulating the applied pressure is also a simplification. The walls of the subarachnoid space, where CSF flows, may be elastic and dampen the pressure wave. Other effects like an obstruction in the SAS may also have an effect on the pressure wave. This limitation also applies to the poroelastic model, as stated by Støverud et al. [50].

Modelling the spinal cord as an elastic solid allows for understanding of how the spinal cord deforms under pressure. However, the models do not take into account the fluid flow within the cord. The fluid that is present in a syrinx must come from somewhere, and a model that includes fluid flow within the cord, such as a poroelastic model, may give a better idea of how a syrinx forms than a model that considers the spinal cord as a solid.

Chapter 8

Conclusions

This thesis has developed two mathematical models for simulating the response of the spinal cord under pressure induced by CSF flow. The simulations were done on a mesh created from the geometry of an actual spinal cord segment from a sheep. The applied pressure was based on pressure data from a patient with the Chiari I malformation. The mathematical models were discretized using the finite element method and the discretizations were implemented in Python using FEniCS[1]. The implementations were verified using visual tests and the method of manufactured solutions. The use of an efficient scheme when calculating the integral in the viscoelastic constitutive relationship (3.45) resulted in a computational speed comparable to the calculations of the linear elasticity constitutive relationship (3.8).

Simulations using the spinal cord geometry were run with both the SLS model and the linear elasticity model, and different model parameters were tested. Simulations over several cycles were also run. The results from using the SLS model with different parameters were compared, and results from using the SLS model were compared with results from using the linear elasticity model.

There was a small but distinct effect in using a viscoelastic model as compared to a purely elastic model, in that the behaviour of the viscoelastic model lagged behind that of the linear elastic model by approximately 10ms. There was also a small variation in the peak displacements when running a simulation over several cycles using the viscoelastic model in the order of $10^{-7} - 10^{-8}$; this did not occur when using the linear elastic model. However, it is unclear whether this has any effect in the context of syrinx formation.

Apart from the lag discussed above, the viscoelastic model displayed results that were qualitatively very similar to the results from the linear elasticity model. Although the magnitudes of the displacements were approximately 15 times larger for the viscoelastic model than for the linear elastic model with default parameters,

the difficulty in selecting appropriate parameter values may be the cause of this.

Due to the fact that the viscoelastic model exhibited behaviour very similar to that of the linear elastic model, it is unlikely that using a linear viscoelastic model will further understanding of syrinx formation. In that respect, a poro-elastic model, which takes into account fluid flow within the spinal cord, may be of more interest in further studies into the underlying mechanisms of syringomyelia.

An interesting extension to this thesis would be to obtain MRE data as well as patient specific geometries from a set of healthy and unhealthy patients. This would allow for simulations using parameter values that are specific to a given patient.

As using a linear viscoelasticity model did not produce results that create new insight into the mechanics of syringomyelia, the use of a non-linear model may be of interest.

Another interesting extension would be to couple a simulation of CSF flow, using computational fluid dynamics, to an anatomically realistic geometry. Doing this would allow for example the effect of an obstruction in the SAS on a patient-specific spinal cord geometry to be investigated.

Appendix A

Code

A.1 Testing numerical integration

```
"""
Testing two different methods of approximating an integral in
                                    time:
 - Trapezoidal sum
 - Efficient sum based on the trapezoidal sum

 This script calculates solutions for a given integral and
                                     compares them tol
 the exact solution for both methods.
"""

from matplotlib.pyplot import *
from trapeziodal_test import Trap_Tester, Efficient_Trap_Tester
import numpy as np
import math as m
import sys, time

def compute(n, rule):
  a = 0; b = 1;
  test = rule(a, b, n)
  t = test.t
  dt = test.dt
  sol = np.zeros(n)
  # trapn = 0.5*dt*test.Ds(0)*test.eps(t[-1])
  t0 = time.time()
  sol[0] = test.sum(0)
  for i in range(1,n):
    sol[i] = test.sum(i) + test.end_term(i)
  t1 = time.time()
  t_comp = t1 - t0
  error = test.exact(t) - sol
```

```
  E = m.sqrt(dt*sum(error**2))
  return E, t_comp

def convergence(compute, rule):
    h = []  # dt
    E = []  # errors
    nlist = [10, 20, 40, 80, 160, 320, 640, 1280]
    N = 8
    for n in nlist:
        h.append(1./n)
        Eval, t = compute(n, rule)
        E.append(Eval)


    print "Finished computing h and E."
    print E

    # Convergence rates
    from math import log as ln #(log is a dolfin name too)
    print "Convergence rate:"
    print "h=%10.2E, E=%12.10E, r=--" % (h[0], E[0])
    for i in range(1, len(E)):
        r = ln(E[i]/E[i-1])/ln(h[i]/h[i-1])
        print "h=%10.2E, E=%12.10E, r=%.4f" % (h[i], E[i], r)
    # Return time taken to compute last (longest) sum
    return t

def plot_stuff(rule):
  n = 101
  if rule == "trap":
    E, sol, t, exact = compute_trapezoidal(n)
  elif rule == "efficient":
    E, sol, t, exact = compute_efficient(n)

  plot(t, exact, '-', t, sol, '--')
  legend(['Exact', 'Numerical (%s)' % rule])
  show()


if __name__ == '__main__':
  print "--------- Regular trapezoidal rule -----------"
  t0 = convergence(compute, Trap_Tester)
  # plot_stuff("trap")
  print "----------------------------------------------"
  print "-------- Efficient trapezoidal rule ----------"
  t1 = convergence(compute, Efficient_Trap_Tester)
  # plot_stuff("efficient")
  print "Time taken for regular trapezoidal rule: ", t0
  print "Time taken for efficient trapezoidal rule: ", t1
  print "Speedup: ", float(t0)/(t1)

```

```
  h = []
  E_T = []  # errors
  E_E = []
  nlist = [10, 20, 40, 80, 160, 320, 640, 1280]
  N = 8
  for n in nlist:
      h.append(1./n)
      Eval, t = compute(n, Trap_Tester)
      E_T.append(Eval)
      Eval, t = compute(n, Efficient_Trap_Tester)
      E_E.append(Eval)

  print "Finished computing h and E."
  print "Differences (T, E):"
  for i in range(len(E_E)):
      print "h=%6.2g, E_T=%10g, E_E=%10g, diff=%g" % (h[i], E_T[i
                                          ], E_E[i], E_T[i]-E_E[i])
```

/home/nina/Dropbox/bitbucket/master_thesis_src/src/integral_test.py

```
""" Implement the trapezoidal rule and test it for a given
                                  function """
"""
    def trapezoidal(self, i):
        Ds, u = self.Ds, self.u
        t, dt = self.t, self.dt
        eps = self.eps
        s = 0*eps(self.u[0])
        for j in range(1, i):
            s += Ds(t[i]-t[j])*eps(u[j]) + Ds(t[i]-t[j-1])*eps(u[
                                                    j-1])
        return 0.5*dt*s
"""
import numpy as np
import math as m
import matplotlib.pyplot as plt

class Problem:
  def exact(self, t):
    return 0.5*(np.exp(-t) + np.sin(t) - np.cos(t))

  def Ds(self, t):
    return m.exp(-(t))

  def eps(self, u):
    return m.sin(u)

class Trap_Tester(Problem):

  def __init__(self, a, b, n):
    self.t = np.linspace(a, b, n)
    self.dt = self.t[1] - self.t[0]
```

```
32    def sum(self, i):
        t, dt = self.t, self.dt
34      Ds, eps = self.Ds, self.eps
        s = 0.5*Ds(t[i] - t[0])*eps(t[0])
36      for j in range(1, i):
          s += Ds(t[i] - t[j])*eps(t[j])
38      return dt*s

40    def end_term(self, i):
        t, dt = self.t, self.dt
42      Ds, eps = self.Ds, self.eps
        return 0.5*dt*Ds(0)*eps(t[i])
44
  class Efficient_Trap_Tester(Problem):
46
    def __init__(self, a, b, n):
48      self.t = np.linspace(a, b, n)
        self. dt = self.t[1] - self.t[0]
50      self.tot_sum = 0
        self.temp_sum = 0
52
    def sum(self, i):
54      t, dt = self.t, self.dt
        Ds, eps = self.Ds, self.eps
56      if i == 0:
          self.temp_sum = Ds(t[i]-t[0])*eps(t[0])
58        self.tot_sum = 0.5*dt*self.temp_sum
        else:
60        self.temp_sum = np.exp(-(t[i]-t[i-1]))*self.temp_sum \
                        + Ds(t[i]-t[i-1])*eps(t[i-1])
62        self.tot_sum = dt*self.temp_sum
        return self.tot_sum
64
    def end_term(self, i):
66      t, dt = self.t, self.dt
        Ds, eps = self.Ds, self.eps
68      return 0.5*dt*Ds(0)*eps(t[i])
```

/home/nina/Dropbox/bitbucket/master_thesis_src/src/trapeziodal_test.py

Appendix B

The FEniCS Software – usage

This chapter shows an example of the syntax one may use with the FEniCS software. The example is based on the Poisson problem discretized in section 4.1, thus the discrete problem to be implemented is

Find $u^h \in V^h$ such that

$$a(u^h, v) = L(v), \quad \forall v \in \hat{V}^h, \tag{B.1}$$

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \; d\Omega,$$
$$L(v) = \int_\Omega f \cdot v \; d\Omega - \int_{\Gamma_N} g \cdot v \; d\Gamma.$$

The domain Ω will in this example be the unit square. The functions f, g are chosen to be

$$f = 10\, e^{-((x-0.5)^2+(y-0.5)^2)/0.02} \tag{B.2}$$
$$g = \sin(5x) \tag{B.3}$$

The Dirichlet boundary Γ_D is chosen to be the vertical boundaries of the unit square, with boundary condition $u = 0$. The Neumann boundary is chosen to be the horizontal boundaries.

Implementation

Note: the following example is taken from the Poisson demo in the FEniCS documentation, `http://fenicsproject.org/documentation/dolfin/1.3.0/python/demo/documented/poisson/python/documentation.html`.

First, the dolfin module is imported:

```
from dolfin import *
```

Code B.1 :

We begin by defining a mesh of the domain and a finite element function space V relative to this mesh. As the unit square is a very standard domain, we can use a built-in mesh provided by the class UnitSquareMesh. In order to create a mesh consisting of 32 x 32 squares with each square divided into two triangles, we do as follows

```
# Create mesh and define function space
mesh = UnitSquareMesh(32,32)
V = FunctionSpace(mesh, "CG", 1)
```

Code B.2 :

The second argument to FunctionSpace is the finite element family, while the third argument specifies the polynomial degree. Thus, in this case, our space V consists of first-order, continuous Lagrange finite element functions (or in order words, continuous piecewise linear polynomials).

Next, we want to consider the Dirichlet boundary condition. A simple Python function, returning a boolean, can be used to define the subdomain for the Dirichlet boundary condition (Γ_D). The function should return True for those points inside the subdomain and False for the points outside. In our case, we want to say that the points (x, y) such that $x = 0$ or $x = 1$ are inside on the inside of Γ_D. (Note that because of rounding-off errors, it is often wise to instead specify $x < \epsilon$ or $x > 1 - \epsilon$ where ϵ is a small number (such as machine precision).)

```
# Define Dirichlet boundary (x = 0 or x = 1)
def boundary(x):
    return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EPS
```

Code B.3 :

Now, the Dirichlet boundary condition can be created using the class DirichletBC. A DirichletBC takes three arguments: the function space the boundary condition applies to, the value of the boundary condition, and the part of the boundary on which the condition applies. In our example, the function space is V, the value of the boundary condition (0.0) can represented using a Constant and the Dirichlet

boundary is defined immediately above. The definition of the Dirichlet boundary condition then looks as follows:

```
# Define boundary condition
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundary)
```

Code B.4 :

Next, we want to express the variational problem. First, we need to specify the trial function u and the test function v, both living in the function space V. We do this by defining a TrialFunction and a TestFunction on the previously defined FunctionSpace V.

Further, the source f and the boundary normal derivative g are involved in the variational forms, and hence we must specify these. Both f and g are given by simple mathematical formulas, and can be easily declared using the Expression class. Note that the strings defining f and g use C++ syntax since, for efficiency, DOLFIN will generate and compile C++ code for these expressions at run-time.

With these ingredients, we can write down the bilinear form a and the linear form L (using UFL operators). In summary, this reads

```
# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)
    ) / 0.02)")
g = Expression("sin(5*x[0])")
a = inner(grad(u), grad(v))*dx
L = f*v*dx - g*v*ds
```

Code B.5 :

Now, we have specified the variational forms and can consider the solution of the variational problem. First, we need to define a Function u to represent the solution. (Upon initialization, it is simply set to the zero function.) A Function represents a function living in a finite element function space. Next, we can call the solve function with the arguments $a == L$, u and bc as follows:

```
# Compute solution
u = Function(V)
solve(a == L, u, bc)
```

Code B.6 :

settings for solving a variational problem have been used. However, the solution process can be controlled in much more detail if desired.

A Function can be manipulated in various ways, in particular, it can be plotted and saved to file. Here, we output the solution to a VTK file (using the suffix .pvd) for later visualization and also plot it using the plot command:

```
# Save solution in VTK format
file = File("poisson.pvd")
file << u

# Plot solution
plot(u, interactive=True)
```

Code B.7 :

Bibliography

[1] FEniCS Project. URL `http://fenicsproject.org/`. Accessed: 2014-04-04.

[2] What is Spinal Cord Injury. URL `http://www.spinal-injury.net/what-is-a-spinal-cord-injury.htm`. Accessed: 23.05.2014.

[3] Milton Abramowitz and Irene A. Stegun, editors. *Hanbook of Mathematical Functions: with Formulas, Graphs and Mathematical Tables*. Dover Books on Mathematics. Dover Publications, 9 edition, 1965.

[4] G.B. Arfken, H.J. Weber, and F.E. Harris. *Mathematical Methods for Physicists: A Comprehensive Guide*, chapter 1, pages 75–76. Elsevier, 2012. ISBN 9780123846549.

[5] H. T. Banks, Shuhua Hu, and Zackary R. Kenz. A Brief Review of Elasticity and Viscoelasticity. Technical report, Center for Research and Scientific Computation, and Department of Mathematics, North Carolina State University, 2010.

[6] Fafa Ben-Hatira, Kaouthar Saidane, and Abdelfatah Mrabet. A finite element modeling of the human lumbar unit including the spinal cord. *Journal of Biomedical Science and Engineering*, 52:146–152, 2012.

[7] C. D. Bertram. A numerical investigation of waves propagating in the spinal cord and subarachnoid space in the presence of a syrinx. *Journal of Fluids and Structures*, 25(7):1189–1205, 2009.

[8] C. D. Bertram, A. R. Brodbelt, and M. A. Stoodley. The Origins of Syringomyelia: Numerical Models of Fluid/Structure Interactions in the Spinal Cord. *Journal of Biomechanical Engineerng*, 127(7), 2005.

[9] Lynne E Bilston. Brain Tissue Mechanical Properties. In Karol Miller, editor, *Biomechanics of the Brain*, pages 69–89. Springer, 2011.

[10] Ronald Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill, 3 edition, 1999.

[11] Are Magnus Bruaset. *A survey of preconditioned iterative methods*. Pitman Research Notes in Mathematics. Longman Scientific & Technical, 1995.

[12] Shakoon Cheng, Elizabeth C. Clarke, and Lynne E Bilston. Rheological properties of the tissues of the central nervous system: A review. *Medical Engineering & Physics*, 30:1318–1337, 2008.

[13] Elizabeth C. Clarke. Spinal Cord Mechanical Properties. In Lynne E. Bilston, editor, *Neural Tissue Biomechanics*. Springer, 2011.

[14] Alan R Crossman and David Neary. *Neuroanatomy*. Elsevier, 2010.

[15] Germund Dahlquist and Åke Björck. *Numerical Methods in Scientific Computing*, volume 1. SIAM, 2008.

[16] Robert Dautray and Jaques-Louis Lione. *Mathematical Analysis and Numerical Methods for Science and Technology*, volume 1. Springer-Verlag, 1990.

[17] ND RN Diane M. Mueller and MD John J. Oro'. Prospective Analysis of Presenting Symptoms Among 265 Patients With Radiographic Evidence of Chiari Malformation Type I With or Without Syringomyelia. *Journal of the American Academy of Nurse Practitioners*, 16(3), 2004.

[18] Maria T. Dolar, Victor M. Haughton, Bermans J. Iskandar, and Mark Quigley. Effect of craniocervical decompression on peak CSF velocities in symptomatic patients with Chiari I malformation. *Americal Journal of Neuroradiology*, 25 (10):142–145, 2004.

[19] N. S. J. Elliot, C. D. Bertram, B. A. Martin, and A. R. Brodbelt. Syringomyelia: A review of the biomechanics. *Journal of Fluids and Structures*, 40:1–24, 2013.

[20] Henry Gray. *Anatomy of the human body*. Lea & Febiger; Bartleby.com, 1918, 2000.

[21] Michael A Green, Lynne E Bilston, and Ralph Sinkus. In vivo brain viscoelastic properties measured by magnetic resonance elastography. *NMR in Biomedicine*, 21(7):755–764, 2008. URL `http://onlinelibrary.wiley.com/doi/10.1002/nbm.1254/full`.

[22] J. A. Grotenhuis. Chiari Malformation. In *Practical Handbook of Neurosurgery*, pages 1246–1258. Springer, 2009.

[23] Victor Haughton, Frank Korosec, Joshua E. Medow, Maria T. Dolar, and Bermans Iskandar. Peak Systolic and Diastolic CSF Velocity in the Foramen Magnum in Adult Patients with Chiari I Malformations and in Normal Control Participants. *Americal Journal of Neuroradiology*, 24(24):169–176, 2003.

[24] S. Hentschel, K.-A. Mardal, A.E. Løvgren, S. Linge, and V. Haughton. Characterization of Cyclic CSF Flow in the Foramen Magnum and Upper

Cervical Spinal Canal with MR Flow Imaging and Computational Fluid Dynamics. *Americal Journal of Neuroradiology*, 31:997–1002, 2010.

[25] Tin-Kan Hung, Guan-Liang Chung, Hsin-Sun Lin, Frank R. Walter, and Leon Bunfgin. Stress-strain relationship of the spinal cord of anesthetized cats. *Journal of Biomechanics*, 14(4):269–276, 1981.

[26] Mika Juntunen and Rolf Stenberg. Nitsche's Method for general boundary conditions. *Mathematics of Computation*, 78(267):1353–1374, 2009.

[27] RobertC. Kirby and Anders Logg. The finite element method. In Anders Logg, Kent-Andre Mardal, and Garth Wells, editors, *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*, pages 77–94. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-23098-1. doi: 10.1007/978-3-642-23099-8_2. URL `http://dx.doi.org/10.1007/978-3-642-23099-8_2`.

[28] Dieter Klatt, Uwe Hamhaber, Patrick Asbach, Jürgen Braun, and Ingolf Sack. Noninvasive assessment of the rheological behavior of human organs using multifrequency MR elastography: a study of brain and liver viscoelasticity. *Physics in medicine and biology*, 52(24):7281, 2007. URL `http://iopscience.iop.org/0031-9155/52/24/006`.

[29] J. Klekamp. Syringomyelia. In *Practical Handbook of Neurosurgery*, pages 1260–1276. Springer, 2009.

[30] Mats G. Larson and Fredrik Bengzon. *The Finite Element Method*. Number 10 in Texts in Computational Science and Engineering. Springer, 2013.

[31] Peter Linz. *Analytical and Numerical Methods for Volterra Equations*. Studies in Applied Mathematics. SIAM, 1985.

[32] Anders Logg and Kent-Andre Mardal. Lectures on the Finite Element Method, 2014.

[33] Tom Lyche. Lecture Notes for MAT-INF 4130, 2013.

[34] Paul C. Matthews. *Vector Calculus*, chapter 5. Springer Undergraduate Mathematics Series. Springer, 2005.

[35] Karol Miller. Constitutive model of brain tissue suitable for finite element analysis of surgical procedures. *Journal of Biomechanics*, 32(5):531–537, May 1999. ISSN 0021-9290. URL `http://www.sciencedirect.com/science/article/B6T82-40BGNGJ-B/2/3a0661e0a906be58125b7e01c66a68da`.

[36] Ronan O'Rahilly, Fabiola Müller, Stanley Carpenter, and Rand Swenson. *Basic Human Anatomy*. Dartmouth Medial School, 2008.

[37] Hiroshi Ozawa, Takeo Matsumoto, Toshiro Ohashi, Masaaki Sato, and Shoichi Kokubun. Mechanical properties and function of the spinal pia mater. *Journal of Neurosurgery (Spine)*, 1:122–127, 2004.

[38] Cecilia Persson, Jon L. Summers, and Richard M. Hall. Modelling of Spinal Cord Biomechanics: In Vitro and Computational Approaches. In Lynne E. Bilston, editor, *Neural Tissue Biomechanics*. Springer, 2011.

[39] Mark F. Quigley, Bermans Iskandar, Michelle A. Quigley, Mark Nicosia, and Victor Haughton. Cerebrospinal fluid flow in foramen magnum: temporal and spatial patterns at MR imaging in volunteers and in patients with Chiari I malformation. *Radiology*, 232(10):229–236, 2004.

[40] A. Roldan, O. Wieben, V. Haughton, T. Osswald, and N. Chesler. Characterization of CSF Hydrodynamics in the Presence and Absence of Tonsillar Ectopia by Means of Computational Flow Analysis. *Americal Journal of Neuroradiology*, 30(5):941–6, 2009.

[41] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[42] Ingolf Sack, Bernd Beierbach, Uwe Hamhaber, Dieter Klatt, and Jürgen Braun. Non-invasive measurement of brain viscoelasticity using magnetic resonance elastography. *NMR in Biomedicine*, 21(3):265–271, 2008. URL `http://onlinelibrary.wiley.com/doi/10.1002/nbm.1189/abstract`.

[43] N. Shaffer, B. Martin, and F. Loth. Cerebrospinal fluid hydrodynamics in type I Chiari malformation. *Neurosurgical Research*, 33(3), 2011.

[44] Simon Shaw. *Finite Element and Discrete Time Methods for Continuum Problems with Memory and Applications to Viscoelasticity*. PhD thesis, Brunel University, 1993.

[45] Simon Shaw and J. R. Whiteman. Computational Modelling of problems with memory with emphasis on viscoelastic solid deformation, 1998.

[46] Simon Shaw and J. R. Whiteman. Numerical solution of linear quasistatic heredetary viscoelasticity problems I: A posteriori estimates. Technical report, BICOM, 1999.

[47] Simon Shaw and J. R. Whiteman. Numerical solution of linear quasistatic heredetary viscoelasticity problems I: A priori eatimates. Technical report, BICOM, 1999.

[48] Joshua H. Smith and Joseph A.C. Humphrey. Interstitial transport and transvascular fluid exchange during infusion into brain and tumor tissue. *Microvascular Research*, 73:58–73, 2007.

[49] Karen H. Støverud, Hans Petter Langtangen, Victor Haughton, and Kent-

Andre Mardal. CSF pressure and velocity in obstructions of the subarachnoid spaces. *Neuroradiol J.*, 26(24), 2013.

[50] Karen H. Støverud, Martin Alnæs, Hans Petter Langtangen, Victor Haughton, and Kent-André Mardal. Effect of pia mater, central canal, and geometry on wave propagation and fluid movement in the cervical spinal cord. *Manuscript submitted for publication*, 2014.

[51] K. Stüben. An Introduction to Algebraic Multigrid. In U. Trottenberg, C. W. Oosterlee, and A. Schüller, editors, *Multigrid*. Academic Press, 2001.

[52] R. Shane Tubbs and W. Jerry Oakes, editors. *The Chiari Malformations*. Springer New York, 2013.