Robust Adaptive Finite Element Schemes for Viscoelastic Solid Deformation: an investigative study

Simon Shaw & J. R. Whiteman

BICOM: *Brunel Institute of Computational Mathematics*
Department of Mathematics and Statistics

Brunel University,
UB8 3PH, England.

February 10, 1999

Final report for the US Army's European Research Office Seed Project: contract number N68171-97-M-5763, September 1997 – November 1998

Abstract

This is the final report detailing the seed project research which was funded by the U.S. Army through its European Research Office during the period September 1997 — November 1998.

The primary purpose of the research funding was to enable Prof. J.R. Whiteman and Dr. S. Shaw (BICOM, Brunel University, Uxbridge, England) to collaborate with Dr. A.R. Johnson (Army Research Laboratory, Vehicle Technology Center, NASA, Langley, VA, USA) toward developing a framework for the adaptive finite element solution of quasistatic viscoelasticity problems.

The major goal was to develop this framework in the context of:

- the practical utility of the **internal variable** formulation, as used by Dr. Johnson; and
- the theoretical utility of the **hereditary integral** formulation, as used at BICOM.

The first of these allows for practical software to be developed in a straightforward way from existing linear elasticity codes, while the second facilitates the derivation of mathematically rigorous *a posteriori* error estimates—the essential building block for **adaptive** finite element solvers.

During the project we proposed and developed a hybrid algorithm blending the best features of these two approaches. Also, we implemented our *a posteriori* error estimates to produce software capable of automatic spatial error control based on adaptive meshing. A prototype version of this software is now mounted on Dr. Johnson's workstation at NASA, Langley, and a short "manual" illustrating the configuration and use of this software is included later in this report. Full details of the work undertaken on the Seed Project are also contained in the following pages.

Contents

Chapter 1

Introduction

1.1 The report in a nutshell

This report is based on a research logbook (initiated on October 4, 1997) of our collaborative research with Dr. Arthur R. Johnson (at ARL, VTC, NASA, Langley) on Internal Variable methods applied to:

- structural viscoelasticity problems of **quasistatic** type with a view to their formulation and subsequent adaptive solution.

The research was sponsored by the United States Army through a European Research Office Seed Project (purchase order N68171-97-M-5763) and we would like to take this opportunity to gratefully acknowledge this support.

This first part of the document contains a short introduction to the constitutive models used in viscoelasticity theory in terms of **hereditary integrals** (Chapter 2) as well as **internal variables** (Chapters 2 and 3). We then indicate (Chapter 4) how the hereditary integrals lead to **memory** terms appearing in the governing equations, thus resulting in **Partial Differential Volterra** (or PDV) equation problems that need to solved in order to model the physics. This material is based on the paper [50]. We also include a small amount of detail on the phenomena of **Non-Fickian Polymer diffusion** and some recent attempts by applied mathematicians to construct mathematical models. In parallel with this Seed Project we at BICOM have also been actively engaged in developing prototype software to model these effects.*

The Volterra operator (i.e. the memory) can be removed from the differential equations by appealing to the viscoelastic internal variables, in the manner of Johnson and Tessler in [24] for example, and so we explore also (Chapter 5) the equations resulting from this approach.

By "mixing and matching" the various forms of the constitutive relationship with the governing equations, we illustrate in the next part of the report that it is possible to derive a variety of differential equation problems which can then be addressed numerically in

* *The computational modelling of problems of nonlinear diffusion in the contexts of controlled drug release technology and percutaneous drug absorbtion* (Project QR97/2, ongoing)

order to model the problems at hand (Chapter 6). This is important because, as we also illustrate, the numerical discretizations of various **equivalent** continuous problems can result in **differing** numerical solutions. Prior to this Seed Project Shaw and Whiteman had already spent a great deal of effort in deriving *a posteriori* error estimates for the Volterra formulation of the quasistatic problem (see [47, 48, 52, 57, 54, 55]), and this presents a natural entry point into the task of providing adaptive software based on error bounds which are computable in terms of the discrete solution. On the other hand, Johnson and colleagues have expended similar amounts of effort on practical solution software based on internal variable formulations [25, 24, 23, 26]. Since these will result in different numerical solutions the *a posteriori* estimates for the Volterra formulation cannot be used.

Once the implications of this had been fully appreciated (i.e. a duplication of effort in order to derive *a posteriori* estimates for internal variable formulations), one of our foremost goals therefore came to be to derive a "hybrid" formulation of the problem that would have the same numerical solution as the discrete Volterra equations, but could be implemented algorithmically in the same manner as an internal variable method. Our proposed algorithm is detailed in Chapter 7, along with some numerical tests to illustrate that the numerical scheme does indeed behave as it should (in terms of convergence to the continuous solution as the discretization is indefinitely refined).

We move on to adaptivity in Chapter 8. We first recapitulate the *a posteriori* error estimate as given in [55] and then use it to implement adaptive meshing in the linear elasticity context. Once the technique has been detailed the extension to the time dependent linear quasistatic viscoelasticity case is straightforward, and we include several examples to demonstrate adaptive space-discretization-error control by selective mesh refinement.

As noted in [47] (and seemingly also implied by others in, for example, [28, 2]) the robust control of the time discretization error by varying the time steps is a difficult issue for this quasistatic problem, and this is due to there being no time derivative in the governing equation. As a result we feel that guaranteed temporal error control can be achieved only in a more abstract way by measuring the error in a negative norm. Details of our work in this direction are in [53, 56], the first of which gives numerical results for a very simple case which, nevertheless, demonstrates that the resulting error control is effective. The extension of these results to the space-time problem introduced and considered below is non-trivial and, although we are progressing, no results are yet available.

In Chapter 9 we give a brief "manual" describing how to obtain, configure and run the software that was used to produce the numerical results in this report. The report closes with Chapter 10, In which we suggest further research work that would follow naturally from this Seed Project, and then we give a complete bibliography.

The remainder of this chapter is devoted first to a brief discussion of the generic PDV equations that arise when modelling viscoelastic effects, and then to establishing a context and the basic notation for the entire report.

1.2 The context

In their simplest form viscoelastic materials exhibit behaviour characteristic of both classical Hookean solids and Newtonian fluids. The resulting effects are important when the material is deforming under an applied load. This load could, for example, be due to externally applied forces; internal deformation caused by a diffusing penetrant; or, constrained thermal expansion caused by temperature gradients. See for example [30, 7, 38]. Moreover, the material somehow keeps a record of its response history and, for this reason, viscoelastic materials are said to possess **memory**. This memory is manifest in the constitutive relationship between the stress and strain tensors, $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$, and as a result mathematical models of viscoelastic behaviour take the form of partial differential Volterra (PDV) equation problems. The canonical forms of these equations are: the **elliptic Volterra** problem,

$$Au(t) = f(t) + \int_0^t B(t,s)u(s)\,ds; \tag{1.1}$$

the **parabolic Volterra** problem,

$$u'(t) + Au(t) = f(t) + \int_0^t B(t,s)u(s)\,ds; \tag{1.2}$$

and, the **hyperbolic Volterra** problem,

$$u''(t) + Au(t) = f(t) + \int_0^t B(t,s)u(s)\,ds. \tag{1.3}$$

These are supplied with initial and/or boundary data as appropriate, and the dependence on the space variable $\boldsymbol{x}$ is suppressed. In these problems we use A and $B(t,s)$ to represent partial differential operators (acting only in the space variables) where, for example, we could have

$$A := -\nabla^2 \qquad \text{and} \qquad B(t,s) := -\nabla \cdot \phi(t,s)\nabla,$$

although for (**1**.1) and (**1**.3) the appropriate form for A is the linear elasticity operator—with $B(t,s)$ "similar".

The purpose of the first chapters is to illustrate how the memory terms arise in these equations and also to summarize the various PDV equations used when modelling problems of **quasistatic** and **dynamic** viscoelasticity, and **non-Fickian** diffusion in polymers. We also indicate some of the numerical analysis work that has been carried out for these problems (but we do not claim to be exhaustive, for a fuller account see [49]).

Throughout, the positive real number T will denote a final time and we use $\mathcal{J} := [0,T]$ and $\mathcal{I} := (0,T]$ to denote time intervals. Also, for $n = 1, 2$ or 3 we consider $\Omega \subset \mathbb{R}^n$ to be an open bounded domain with boundary $\partial\Omega$. Furthermore, we consider $\partial\Omega$ in the form

$$\partial\Omega := \overline{\Gamma_D \cup \Gamma_N} \qquad \text{with} \qquad \Gamma_D \cap \Gamma_N = \emptyset,$$

where the closed set $\Gamma_D \subseteq \partial\Omega$ is called the **Dirichlet boundary** and is of positive measure so that

$$\int_{\Gamma_D} d\Gamma > 0.$$

We call the (possibly empty) open set $\Gamma_N \subset \partial\Omega$ the **Neumann boundary**. The reason for this terminology is the obvious one where we refer to the type of boundary condition specified on these subsets. We indicate vector-valued quantities with boldface so that, for example, we use $\boldsymbol{x} := (x_i)_{i=1}^n$ to indicate a point in $\mathbb{R}^n$. Tensors are indicated by a further underlining: $\underline{\boldsymbol{\sigma}} = (\sigma_{ij})_{i,j=1}^n$.

Suppose that the interior of a compressible viscoelastic body $\mathcal{G}$ occupies Ω and that its surface coincides with $\partial\Omega$. If at a time t this body is subjected to a system of body forces $\boldsymbol{f} := (f_i(\boldsymbol{x},t))_{i=1}^n$, for $\boldsymbol{x} \in \Omega$, and surface tractions $\boldsymbol{g} := (g_i(\boldsymbol{x},t))_{i=1}^n$, for $\boldsymbol{x} \in \Gamma_N$, then the body $\mathcal{G}$ will deform from its equilibrium configuration. A material particle originally at the point $\boldsymbol{x}$ will move to the new time dependent location $\boldsymbol{x} + \boldsymbol{u}(\boldsymbol{x},t)$ where $\boldsymbol{u} := (u_i)_{i=1}^n$ denotes the displacement vector. In the linear theory these displacements define the symmetric strain tensor $\underline{\boldsymbol{\varepsilon}} := (\varepsilon_{ij})_{i,j=1}^n$ by the relationships:

$$\varepsilon_{ij}(\boldsymbol{u}) := \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \qquad (\mathbf{1}.4)$$

In addition to this strain field there will also be induced in $\mathcal{G}$ a stress field described by the symmetric stress tensor $\underline{\boldsymbol{\sigma}} := (\sigma_{ij})_{i,j=1}^n$. This stress field rationalizes the internal force field which is set up within $\mathcal{G}$ to resist the external forces $\boldsymbol{f}$ and $\boldsymbol{g}$.

The stress field can be related to $\boldsymbol{u}$, $\boldsymbol{f}$ and $\boldsymbol{g}$ by Newton's second law of motion (see later in equation (**4**.1)) and so it is of interest to derive a **constitutive relationship** linking $\underline{\boldsymbol{\sigma}}$ and $\boldsymbol{u}$, or in practice, linking the tensors $\underline{\boldsymbol{\sigma}}$ and $\underline{\boldsymbol{\varepsilon}}$.

Part I

Basic Theory

Chapter 2

Constitutive relationships

2.1 Hereditary constitutive relationships

In classical linear elasticity theory the constitutive relationship between stress and strain is provided by Hooke's law:

$$\sigma_{ij} = D_{ijkl}\varepsilon_{kl} \qquad \text{or} \qquad \underline{\boldsymbol{\sigma}} = \underline{\boldsymbol{D}}\,\underline{\boldsymbol{\varepsilon}},$$

where $\underline{\boldsymbol{D}}$ is a positive-definite fourth-order tensor of elastic coefficients satisfying the symmetries

$$D_{ijkl} = D_{jikl}, \qquad D_{ijkl} = D_{ijlk}, \qquad \text{and} \qquad D_{ijkl} = D_{klij}.$$

The first two of these are implied by the symmetry of $\underline{\boldsymbol{\sigma}}$ and $\underline{\boldsymbol{\varepsilon}}$ while the third follows from energy considerations. However, in viscoelasticity the third of these only applies when the material is isotropic, see [30, Equations (1.10) and (2.62)].

One way of deriving a constitutive relationship for viscoelastic materials is to assume that a Boltzmann superposition of stress increments can be applied where these stress increments are related by Hooke's law to corresponding strain increments. For example, suppose that $\mathcal{G}$ is quiescent for $t < 0$ so that $\underline{\boldsymbol{\varepsilon}}(t) \equiv \underline{\boldsymbol{0}}$ for $t < 0$, and that at $t = 0$ the body undergoes a strain $\underline{\boldsymbol{\varepsilon}}(0)$. Then for $t \geq 0$ the resulting stress is assumed to be given by

$$\underline{\boldsymbol{\sigma}}_0(t) = \underline{\boldsymbol{D}}(t)\underline{\boldsymbol{\varepsilon}}(0),$$

where a time dependence has been introduced into the Hooke's tensor $\underline{\boldsymbol{D}}$. Physically we expect $\underline{\boldsymbol{D}}$ to be a smooth monotone decreasing function of t since it is unrealistic to expect $\underline{\boldsymbol{\sigma}}$ to grow over time for the fixed strain $\underline{\boldsymbol{\varepsilon}}(0)$. (Where would the strain energy come from?) In fact experiments on polymers show that $\underline{\boldsymbol{D}}$ does in fact decrease and this phenomena is known as **stress relaxation**.

Now, let Δt be a small time interval and set $t_i := i\Delta t$. We approximate the strain evolution by the step function

$$\widetilde{\underline{\boldsymbol{\varepsilon}}}(t) := \underline{\boldsymbol{\varepsilon}}(t_i) \qquad \text{in } [t_i, t_{i+1}) \text{ for } i = 0, 1, 2, \ldots,$$

and then each strain increment,

$$\Delta\underline{\boldsymbol{\varepsilon}}(t_{i+1}) := \underline{\boldsymbol{\varepsilon}}(t_{i+1}) - \underline{\boldsymbol{\varepsilon}}(t_i),$$

induces a stress increment according to Hooke's law:

$$\Delta\underline{\boldsymbol{\sigma}}_j(t_i) := \underline{\boldsymbol{D}}(t_i - t_j)\Delta\underline{\boldsymbol{\varepsilon}}(t_j) \qquad \text{for } 1 \le j \le i.$$

Notice that each of these stress increments will also relax according to the time dependence of $\underline{\boldsymbol{D}}$. The total stress at time t_i is now given by superposition:

$$\begin{aligned}\underline{\boldsymbol{\sigma}}(t_i) &:= \underline{\boldsymbol{\sigma}}_0(t_i) + \sum_{j=1}^{i} \Delta\underline{\boldsymbol{\sigma}}_j(t_i), \\ &= \underline{\boldsymbol{D}}(t_i)\underline{\boldsymbol{\varepsilon}}(0) + \sum_{j=1}^{i} \underline{\boldsymbol{D}}(t_i - t_j)\Delta\underline{\boldsymbol{\varepsilon}}(t_j),\end{aligned}$$

and by taking an appropriate limit we get the hereditary constitutive law as

$$\underline{\boldsymbol{\sigma}}(\boldsymbol{x}, t) = \underline{\boldsymbol{D}}(t)\underline{\boldsymbol{\varepsilon}}(\boldsymbol{u}(\boldsymbol{x}, 0)) + \int_0^t \underline{\boldsymbol{D}}(t - s)\underline{\boldsymbol{\varepsilon}}(\boldsymbol{u}'(\boldsymbol{x}, s))\, ds. \tag{2.1}$$

Since we are assuming that $\underline{\boldsymbol{D}}(t)$ is smooth we can arrive at an alternate form by partial integration,

$$\underline{\boldsymbol{\sigma}}(\boldsymbol{x}, t) = \underline{\boldsymbol{D}}(0)\underline{\boldsymbol{\varepsilon}}(\boldsymbol{u}(\boldsymbol{x}, t)) - \int_0^t \underline{\boldsymbol{D}}_s(t - s)\underline{\boldsymbol{\varepsilon}}(\boldsymbol{u}(\boldsymbol{x}, s))\, ds, \tag{2.2}$$

where the subscript s indicates partial differentiation with respect to the **history variable** s. Either of these may be used as the constitutive relationship, and each demonstrates clearly the role of memory in viscoelastic modelling.

To get a feel for the form of the time dependence of the stress relaxation tensor $\underline{\boldsymbol{D}}$ we now describe a perhaps more intuitive method for deriving these constitutive relationships.

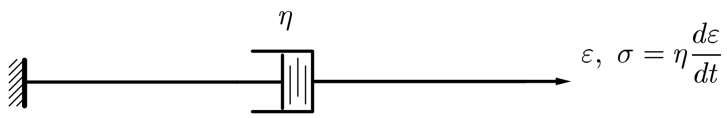
2.2 Spring and dashpot models

We start with the physical observation that viscoelastic materials display the characteristics of both elastic solids and viscous fluids. The kinetics of these type of substances are modelled respectively by the spring and the dashpot. In these models the stress carried by the spring is proportional to the strain in the spring and is given by Hooke's law: $\sigma = E\varepsilon$. The stress carried in the dashpot is proportional to the strain rate and is given by Newton's law of viscosity: $\sigma = \eta\varepsilon'$.

One then models a viscoelastic material by considering a notional system of springs and dashpots with independent stiffness and viscosity parameters. There are essentially two ways to connect a spring to a dashpot: in **series** and in **parallel**. These are the building blocks and are named the "Maxwell" and "Voigt" models.

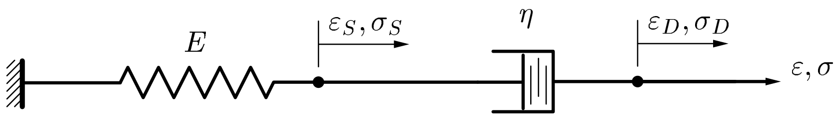
Figure 2.1: A Hookean (linear) spring: $\sigma = E\varepsilon$; E is the spring stiffness

Figure 2.2: A Newtonian (linear) dashpot: $\sigma = \eta \dfrac{d\varepsilon}{dt}$; η is the viscosity



The Maxwell model

Figure 2.3: The Maxwell model



The Maxwell model is a series connection of a spring and dashpot. In this model ε_S and σ_S denote the strain and stress in the spring alone, and ε_D, σ_D denote those in the dashpot alone. The total stress is given by $\sigma = \sigma_S = \sigma_D$ and the total strain by $\varepsilon = \varepsilon_S + \varepsilon_D$. Differentiating and using Hooke's and Newton's laws yield

$$\frac{d\varepsilon}{dt} = \frac{1}{E}\frac{d\sigma_S}{dt} + \frac{\sigma_D}{\eta} \quad \Longrightarrow \quad \frac{d\sigma}{dt} + \frac{\sigma}{\tau} = E\frac{d\varepsilon}{dt}, \tag{\textbf{2}.3}$$

where $\tau := \eta/E$ is the so-called **relaxation time**. Using $\sigma(0) = E\varepsilon(0)$ this ODE is easily solved to give

$$\sigma(t) = Ee^{-t/\tau}\varepsilon(0) + E\int_0^t e^{-(t-s)/\tau}\varepsilon'(s)\,ds,$$

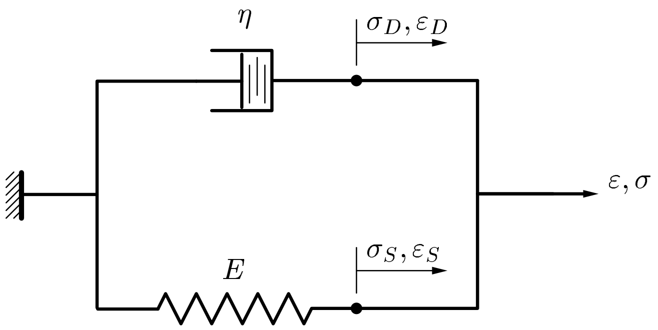
and this is essentially (**2**.1) with the scalar analogue of $\underline{\boldsymbol{D}}$ given by $D(t) = Ee^{-t/\tau}$.

The Voigt model

Connecting the spring and dashpot in parallel yields the Voigt model. This time $\varepsilon_S = \varepsilon_D = \varepsilon$ and equilibrium demands that $\sigma = \sigma_S + \sigma_D$, hence

$$\eta\frac{d\varepsilon}{dt} + E\varepsilon = \sigma \quad \Longrightarrow \quad \frac{d\varepsilon}{dt} + \frac{\varepsilon}{\tau} = \frac{\sigma}{\eta}.$$

Figure 2.4: The Voigt model



This gives the constitutive law in hereditary form as

$$\varepsilon(t) = e^{-t/\tau}\varepsilon(0) + \frac{1}{\eta}\int_0^t e^{-(t-s)/\tau}\sigma(s)\,ds.$$

The Maxwell solid

In his internal variable formulation A.R. Johnson, in for example [24], uses these basic building blocks in the Maxwell solid. Here E_0 and E_1 are spring stiffnesses and σ^*, ε^* are internal stress and strain variables. This time $\sigma^* = E_1\varepsilon^*$, $\varepsilon_D = \varepsilon - \varepsilon^*$ and $\sigma_S = E_0\varepsilon_S$. Also $\sigma^* = \sigma_D$ and this gives

$$E_1\varepsilon^* = \eta\frac{d}{dt}(\varepsilon - \varepsilon^*) \quad \Longrightarrow \quad \frac{d\varepsilon^*}{dt} + \frac{\varepsilon^*}{\tau} = \frac{d\varepsilon}{dt},$$

where now $\tau := \eta/E_1$. Solving this we get

$$\varepsilon^*(t) = e^{-t/\tau}\varepsilon(0) + \int_0^t e^{-(t-s)/\tau}\varepsilon'(s)\,ds = \varepsilon(t) - \frac{1}{\tau}\int_0^t e^{-(t-s)/\tau}\varepsilon(s)\,ds. \qquad (\mathbf{2}.4)$$

where $\varepsilon(0)$ could be any constant but arises here from the initial condition $\varepsilon^*(0) = \varepsilon(0)$.

Now, defining the **stress relaxation function**

$$D(t) := E_0 + E_1 e^{-t/\tau}$$

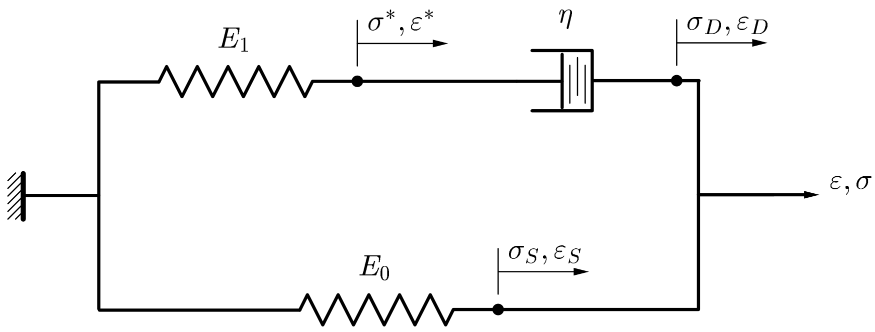
as the scalar analogue to the tensor $\underline{\boldsymbol{D}}(t)$ in (**2**.1) and (**2**.2), and using this in (**2**.4) along with the relation

$$\sigma = \sigma_S + \sigma^* = E_0\varepsilon + E_1\varepsilon^* \qquad (\text{since } \varepsilon_S = \varepsilon),$$

gives

$$\begin{aligned} \sigma(t) &= E_0\varepsilon(t) + E_1 e^{-t/\tau}\varepsilon(0) + \int_0^t E_1 e^{-(t-s)/\tau}\varepsilon'(s)\,ds, \\ &= D(0)\varepsilon(t) - \int_0^t D_s(t-s)\varepsilon(s)\,ds. \end{aligned}$$

Figure 2.5: THE MAXWELL SOLID



This is the scalar analogue of equation (**2**.2) and suggests that we model $\underline{\boldsymbol{D}}$ with the **Dirichlet-Prony series**,

$$\underline{\boldsymbol{D}}(t) = \varphi(t)\underline{\boldsymbol{D}}(0) \qquad (\mathbf{2}.5)$$

where $\varphi(t)$ is a generic stress relaxation function given by

$$\varphi(t) = \varphi_0 + \sum_{i=1}^{N} \varphi_i e^{-\alpha_i t}. \qquad (\mathbf{2}.6)$$

Here the (possibly $\boldsymbol{x}$ dependent) coefficients $\{\varphi_i\}_{i=0}^N$ are non-negative and normalized so that $\varphi(0) = 1$, and the (possibly $\boldsymbol{x}$ dependent) $\{\alpha_i\}_{i=1}^N$ are non-negative. More generally one could of course write (with summation not implied),

$$D_{ijkl}(t) := (D_{ijkl})_0 + \sum_{m=1}^{N_{ijkl}} (D_{ijkl})_m \exp(-(\alpha_{ijkl})_m t).$$

The Dirichlet-Prony series is an extremely convenient form to take for large scale computational approximations to problems (**1**.1), (**1**.2) and (**1**.3) since if

$$\psi(t) := e^{-\alpha t},$$

then one can exploit the simple recurrence

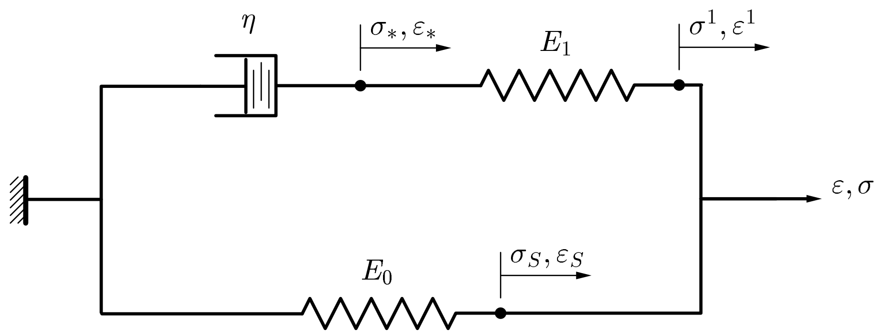
$$\psi(t+k) = e^{-\alpha k}\psi(t)$$

to update the history term arising from a discretization of the Volterra integral. For general Volterra problems one must usually store the entire solution history as the computation advances through the time levels and moreover, at each time level this history needs to be summed to approximate the integral. For such methods the number of operations required at time level N is of the order $O(N^2)$. The Dirichlet-Prony series provides a very useful short cut around this "N^2 problem". (In certain special cases one can also overcome this difficulty using other means, see for example [22, 19]).

2.3 The Reversed Maxwell Solid

One can also arrive at a Maxwell solid by switching the order in which the spring and dashpot appear in the viscous arm of the network. We call this the **Reversed Maxwell Solid**.

Figure 2.6: The Reversed Maxwell Solid



Balancing stresses such that $\sigma_* = \sigma^1$ and using $\varepsilon^1 = \varepsilon - \varepsilon_*$ now gives the differential equation for the new internal variable ε_* as,

$$\frac{d\varepsilon_*}{dt} + \frac{\varepsilon_*}{\tau} = \frac{\varepsilon}{\tau}, \qquad \text{where } \tau := \eta/E_1.$$

Note that ε_* is quite different from ε^* as introduced earlier for the Maxwell model, although there is a simple connection which we demonstrate below. This time we have,

$$\varepsilon_*(t) = \frac{1}{\tau}\int_0^t e^{-(t-s)/\tau}\varepsilon(s)\,ds,$$

where we used $\varepsilon_*(0) = 0$ (otherwise any term of the form $Ae^{-t/\tau}$ may be added on.).

Since $\varepsilon_S = \varepsilon$, the total stress σ is given by,

$$\begin{aligned}
\sigma(t) &= \sigma_S + \sigma^1 = E_0\varepsilon_S + E_1\varepsilon^1, \\
&= E_0\varepsilon + E_1(\varepsilon - \varepsilon_*), \\
&= (E_0 + E_1)\varepsilon(t) - \frac{E_1}{\tau}\int_0^t e^{-(t-s)/\tau}\varepsilon(s)\,ds, \\
&= D(0)\varepsilon(t) - \int_0^t D_s(t-s)\varepsilon(s)\,ds,
\end{aligned}$$

where again we set,

$$D(t) := E_0 + E_1 e^{-t/\tau}.$$

This is exactly as before for the Maxwell solid, and so again represents a scalar analogue of (**2**.2). The question is whether or not this reversed model is of any use, and in this context we note the possibility of strong stability estimates for solution of this ODE.

Note that ε^* and ε_* are very simply related by,

$$\varepsilon(t) = \varepsilon^*(t) + \varepsilon_*(t).$$

This is obvious since the total strain in the viscous arm of the networks is the sum of the strains in each component. This relationship is easily proven by considering the integral representations of $\varepsilon^*(t)$ and $\varepsilon_*(t)$ given earlier.

Let us now briefly try to make a connection with the "ODE formulation" as described by Janovsky *et al.* in [22]. Using $\varepsilon = \varepsilon^* + \varepsilon_*$ in the differential equation for ε_* we easily obtain a differential relationship between ε^* and ε_* as,

$$\frac{d\varepsilon_*}{dt} = \frac{\varepsilon^*}{\tau}.$$

The internal variable ε_* plays essentially the same role as (the scalar analogue of) $\boldsymbol{w}$ in [22, Equation 4.1].

Finally in this section we note that we can also arrive at ODE's for the internal stress variables. For the Maxwell solid we have,

$$\frac{d\sigma^*}{dt} + \frac{\sigma^*}{\tau} = E_1\frac{d\varepsilon}{dt},$$

and for the Reversed Maxwell solid we get:

$$\begin{aligned} & \eta\frac{d\varepsilon_*}{dt} + E_1\varepsilon_* = E_1\varepsilon, \\ \Longrightarrow\qquad & \frac{d\sigma_*}{dt} + E_1\frac{d\varepsilon_*}{dt} = E_1\frac{d\varepsilon}{dt} = \frac{d\sigma^*}{dt} + \frac{\sigma^*}{\tau}. \end{aligned}$$

Clearly $\sigma_* = \sigma^*$ and so (again) $d\varepsilon_*/dt = \varepsilon^*/\tau$.

2.4 The generalized Maxwell solid: internal variables

We now return to the Maxwell solid and generalize the conceptual spring and dashpot model in order to motivate the choice of the Dirichlet-Prony series for the relaxation function as given in (**2**.6). To begin with we assume again a state of uniaxial stress and strain.

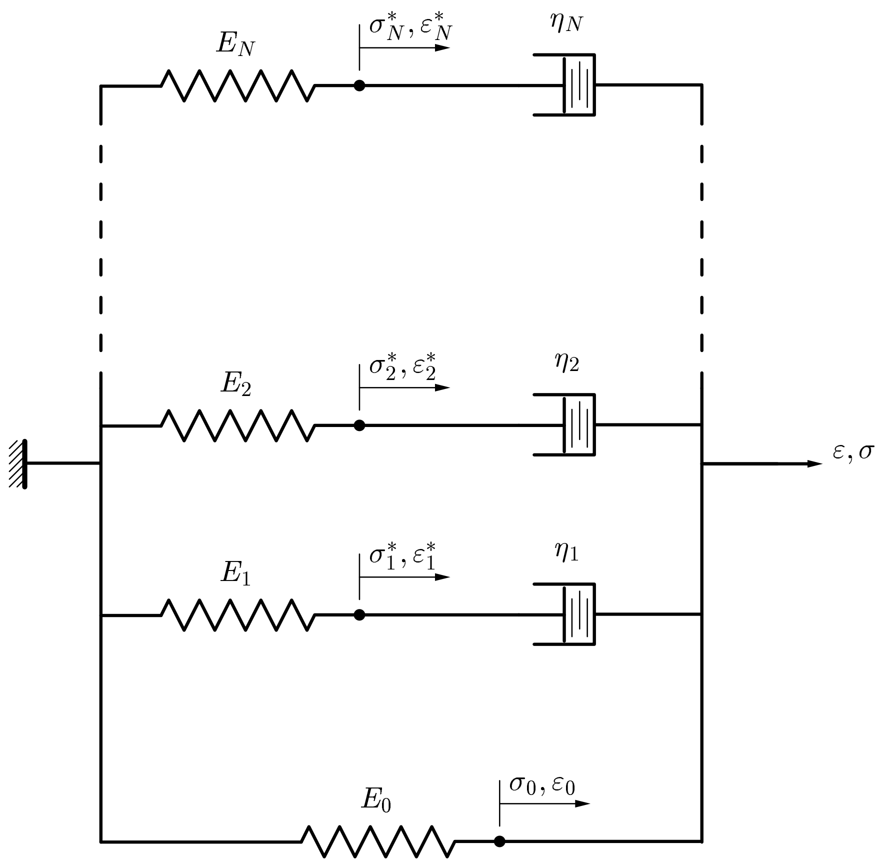
The generalized Maxwell solid, shown in Figure 2.7, consists of a Hookean spring connected in parallel to a sequence of N spring-dashpot components. In this model,

$$\varepsilon_0 = \varepsilon, \qquad \sigma_0 = E_0\varepsilon, \qquad \text{and} \qquad \sigma_i^* = E_i\varepsilon_i^*.$$

Balancing the stresses carried by each of the spring-dashpot pairs we get for each $i \in \{1, \ldots, N\}$ that,

$$\begin{aligned} & \frac{d\varepsilon_i^*}{dt} + \frac{\varepsilon_i^*}{\tau_i} = \frac{d\varepsilon}{dt}, \\ \Longrightarrow\qquad & \varepsilon_i^*(t) = e^{-t/\tau_i}\varepsilon(0) + \int_0^t e^{-(t-s)/\tau_i}\varepsilon'(s)\,ds = \varepsilon(t) - \frac{1}{\tau_i}\int_0^t e^{-(t-s)/\tau_i}\varepsilon(s)\,ds, \end{aligned}$$

Figure 2.7: The Generalized Maxwell solid.



where we used $\varepsilon_i^*(0) = \varepsilon(0)$ and this time we have set $\tau_i := E_i/\eta_i$. The total stress carried by the assemblage is therefore given by:

$$\begin{aligned}
\sigma(t) &= \sigma_0(t) + \sigma_1(t) + \cdots + \sigma_N(t), \\
&= E_0\varepsilon(t) + E_1\varepsilon_1^*(t) + \cdots + E_N\varepsilon_N^*(t), \\
&= E_0\varepsilon(0) + E_0(\varepsilon(t) - \varepsilon(0)) \\
&\qquad + \sum_{i=1}^{N}\left(E_i e^{-t/\tau_i}\varepsilon(0) + \int_0^t E_i e^{-(t-s)/\tau_i}\varepsilon'(s)\,ds\right), \\
&= E(t)\varepsilon(0) + \int_0^t E(t-s)\varepsilon'(s)\,ds, \qquad (\mathbf{2}.7)
\end{aligned}$$

where

$$E(t) := E_0 + \sum_{i=1}^{N} E_i e^{-t/\tau_i}.$$

The constitutive relationship ($\mathbf{2}$.7) is the scalar analogue of ($\mathbf{2}$.1) with the analogue of $\underline{\boldsymbol{D}}(t-s)$ given by $E(t-s)$, which itself is an example of the Dirichlet-Prony series given in ($\mathbf{2}$.6). Note that if we set $E_0 := 0$ then this generalized Maxwell solid actually models a fluid since $\lim E(t) = 0$.

Again, we can also arrive at ODE's connecting the internal stresses,

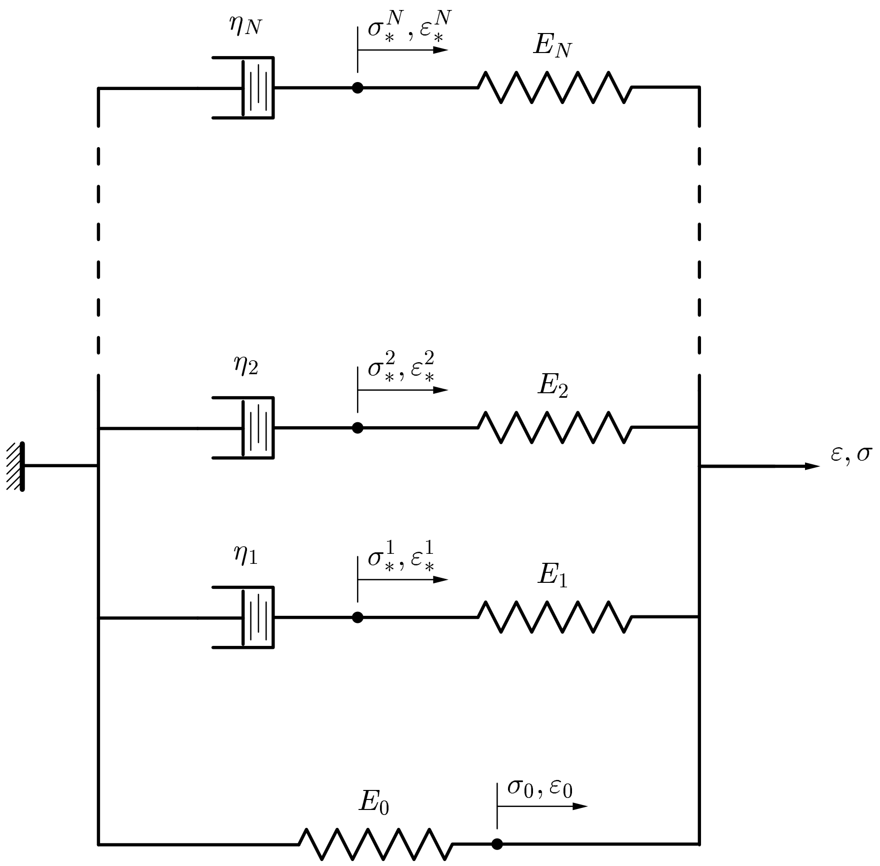
$$\frac{d\sigma_i^*}{dt} + \frac{\sigma_i^*}{\tau_i} = E_i \frac{d\varepsilon}{dt},$$

since $\sigma_i^* = E_i \varepsilon_i^*$.

2.5 The generalized Reversed Maxwell Solid

By switching the order of the springs and dashpots in each of the viscous arms of the generalized Maxwell solid network we arrive at the generalized Reversed Maxwell Solid.

Figure 2.8: THE GENERALIZED REVERSED MAXWELL SOLID.



This time we balance the stresses in each spring-dashpot pair such that $\sigma_*^i = E_i(\varepsilon - \varepsilon_*^i)$ and obtain,

$$\frac{d\varepsilon_*^i}{dt} + \frac{\varepsilon_*^i}{\tau_i} = \frac{\varepsilon}{\tau_i} \qquad \text{for each } i,$$

and where $\tau_i := \eta_i / E_i$. Again, with $\varepsilon_*^i(0) = 0$ we can solve for the internal strains to get,

$$\varepsilon_*^i(t) = \frac{1}{\tau_i} \int_0^t e^{-(t-s)/\tau_i} \varepsilon(s)\, ds,$$

as expected, and we obtain the total stress from:

$$\begin{aligned}
\sigma(t) &= \sigma_0 + E_1(\varepsilon - \varepsilon_*^1) + \cdots + E_N(\varepsilon - \varepsilon_*^N), \\
&= (E_0 + \cdots + E_N)\varepsilon(t) - \int_0^t \left(\frac{E_1}{\tau_1} e^{-(t-s)/\tau_1} + \cdots + \frac{E_N}{\tau_N} e^{-(t-s)/\tau_N} \right) \varepsilon(s)\, ds, \\
&= E(0)\varepsilon(t) - \int_0^t E_s(t-s)\varepsilon(s)\, ds,
\end{aligned}$$

where, again, we set

$$E(t) := E_0 + \sum_{i=1}^{N} E_i e^{-t/\tau_i}.$$

This constitutive relationship between σ and ε is a scalar analogue of (**2**.2), which itself is equivalent to (**2**.1). Hence the constitutive relationship generated by the generalized Reversed Maxwell Solid is equivalent to that generated by the Maxwell Solid.

Once again it is trivial to note the connection,

$$\varepsilon(t) = \varepsilon_i^*(t) + \varepsilon_*^i(t),$$

for each i, and it follows that,

$$\frac{d\varepsilon_*^i}{dt} = \frac{\varepsilon_i^*}{\tau_i}.$$

Introducing the internal stresses we also get,

$$\begin{aligned}
& \eta_i \frac{d\varepsilon_*^i}{dt} + E_i\varepsilon_*^i = E_i\varepsilon, \\
\Longrightarrow \quad & \frac{d\sigma_*^i}{dt} + E_i \frac{d\varepsilon_*^i}{dt} = E_i \frac{d\varepsilon}{dt} = \frac{d\sigma_i^*}{dt} + \frac{\sigma_i^*}{\tau_i}, \\
\Longrightarrow \quad & \frac{d}{dt}(\sigma_*^i - \sigma_i^*) + E_i \left(\frac{\varepsilon - \varepsilon_*^i}{\tau_i} \right) = \frac{\sigma_i^*}{\tau_i} \quad \Longrightarrow \quad \sigma_*^i - \sigma_i^* = \text{ constant } = 0,
\end{aligned}$$

since the stress in each arm is independent of the order in which the components are arranged.

2.6 Other constitutive relationships

The Dirichlet-Prony series is not however the only form used to model the stress relaxation functions, for example the authors of [1] use the **stretched** relaxation function

$$\varphi(t) = \varphi_0 \exp(-(\alpha t)^p) \qquad \text{for } p \in (0, 1]. \tag{2.8}$$

Obviously no simple recurrence exists for this form. Another popular choice for φ is the **power law** where

$$\varphi(t) = \varphi_0 t^{-p} \qquad \text{for } p \in (0, 1), \tag{2.9}$$

although from either of (**2**.1) or (**2**.2) this implies that either $\underline{\boldsymbol{\varepsilon}}(0)$ is zero irrespective of the magnitude of the load, or $\underline{\boldsymbol{\sigma}}(0)$ is infinite. Neither of these are physically realistic and so we would prefer to modify this law to

$$\varphi(t) = \varphi_0(t + \varphi_1)^{-p} \qquad \text{for } p \in (0,1), \tag{2.10}$$

where $\varphi_1 > 0$ in order to remove the non-physical behaviour. Nonetheless, it is instructive to see how one might "derive" the power law, and for this we borrow heavily from Chern's thesis [4] which exploits the fractional calculus.

The formulation is based on the observed fact that viscoelastic materials behave in a way intermediate to that of solids and fluids. Interpreting this literally yields a constitutive law that contains fractional derivatives. Unfortunately we are unable here to give this interpretation the depth it deserves and instead try only to illustrate the main point. Recall that the stress in a solid is proportional to the strain while the stress in a fluid is proportional to the strain rate. Accepting the intermediate nature of viscoelastic materials the idea is to define the viscoelastic constitutive law as:

$$\underline{\boldsymbol{\sigma}}(t) = \underline{\boldsymbol{D}}^{(0)}\underline{\boldsymbol{\varepsilon}}(t) + \underline{\boldsymbol{D}}^{(1)}\partial_t^{\alpha}\underline{\boldsymbol{\varepsilon}}(t), \tag{2.11}$$

for constant fourth order tensors $\underline{\boldsymbol{D}}^{(0)}$ and $\underline{\boldsymbol{D}}^{(1)}$, and where $\alpha \in [0,1)$. The fractional derivative operator may be defined as:

$$\partial_t^{\alpha}\underline{\boldsymbol{\varepsilon}}(t) := \frac{\partial}{\partial t}\left(\frac{1}{\Gamma(1-\alpha)}\int_0^t (t-s)^{-\alpha}\underline{\boldsymbol{\varepsilon}}(s)\,ds\right), \qquad \text{for } \alpha \in [0,1). \tag{2.12}$$

(Note that α can be irrational, even though the word "fractional" is always used.) By firstly integrating by parts in (**2**.12) and then taking the differentiation through, Chern arrives at a constitutive law which is suitable for use within the standard finite element framework. Two solution schemes are considered: a solution in the Laplace transform domain and a direct time domain solution. However, in this case there is no efficient history storage and so the operation counts and computer memory requirement grow without bound as the time step is diminished.

The "justification" for the power law is as follows. Carrying out this integration-differentiation process gives

$$\partial_t^{\alpha}\varepsilon(t) = \frac{t^{-\alpha}}{\Gamma(1-\alpha)}\varepsilon(0) + \frac{1}{\Gamma(1-\alpha)}\int_0^t (t-s)^{-\alpha}\varepsilon'(s)\,ds, \tag{2.13}$$

and using this in the scalar analogue of equation (**2**.11) we now arrive at the constitutive law:

$$\sigma(t) = E_0\varepsilon(t) + \frac{E_1 t^{-\alpha}}{\Gamma(1-\alpha)}\varepsilon(0) + \frac{E_1}{\Gamma(1-\alpha)}\int_0^t (t-s)^{-\alpha}\varepsilon'(s)\,ds. \tag{2.14}$$

This seems to combine (**2**.1) and (**2**.2) when $\varphi(t)$ is given by the power law, (**2**.9).

We now have several candidates for the constitutive law and these may be used to generate a variety of differential equation problems. Later in Chapter 4 we do just this and demonstrate how concrete forms of the abstract problems (**1**.1), (**1**.2) and (**1**.3), as well as some non-standard variants, can be derived to model viscoelastic behaviour.

Chapter 3

Multidimensional internal variables

3.1 Overview

So much for uniaxial states of stress and strain. In fact it can be shown that for each **relaxation mode** (i.e. each spring-dashpot pair) in higher dimensions, there is an ODE governing the evolution of each of the internal strain tensor components. Thus we have,

$$\frac{d\ {}^*\varepsilon_{ij}^m}{dt} + \frac{{}^*\varepsilon_{ij}^m}{\tau_m} = \frac{d\varepsilon_{ij}}{dt} \qquad \text{(for fixed } \boldsymbol{x}\text{)},$$

where the details are given below. The significance of these internal variable formulations for the viscoelastic constitutive behaviour lies in the fact that it is possible to solve some kinds of viscoelasticity problems, when the relaxation functions are in the form of a Dirichlet-Prony series (**2**.6), using only a linear elasticity solver and an ODE solver. This obviates the need to create special software for quasistatic viscoelasticity problems. For more on this we refer again to [24] and also to [43], but we return to this topic in a later chapter.

3.2 Multidimensional constitutive relationships

In [54] for example Shaw and Whiteman use a hereditary multidimensional constitutive relationship of the form,

$$\sigma_{ij}(t) = D_{ijkl}(0)\varepsilon_{kl}(t) - \int_0^t \frac{\partial D_{ijkl}(t-s)}{\partial s}\varepsilon_{kl}(s)\,ds$$

at each fixed point $\boldsymbol{x}$ in a viscoelastic body (see (**2**.2) given earlier). Here $\underline{\boldsymbol{D}}(t) = (D_{ijkl}(t))_{i,j,k,l=1}^n$ (for problems posed in $\mathbb{R}^n$) is a fourth-order stress relaxation tensor given, for example, by the Dirichlet-Prony series,

$$D_{ijkl}(t) := (D_{ijkl})_0 + \sum_{m=1}^{N_D} (D_{ijkl})_m e^{-t/\tau_m}.$$

However, Johnson and Tessler in [24, equation 20.9] write this in a different form and it is useful here to explore the connection.

Assuming that $t = 0$ is a reference time such that $\underset{\sim}{\boldsymbol{\varepsilon}}(t) = \underset{\sim}{\boldsymbol{0}}$ for all $t < 0$, then the constitutive equation is,

$$\begin{aligned}
\sigma_{ij}(t) &= C_{ijkl}\varepsilon_{kl}(t) + \int_{-\infty}^{t} {}^{*}C_{ijkl}(t-s)\frac{\partial \varepsilon_{kl}(s)}{\partial s}\, ds, \\
&= \Big(C_{ijkl} + {}^{*}C_{ijkl}(0)\Big)\varepsilon_{kl}(t) - \int_{0}^{t} \frac{\partial\, {}^{*}C_{ijkl}(t-s)}{\partial s}\varepsilon_{kl}(s)\, ds.
\end{aligned}$$

By comparing these two forms we see that

$$D_{ijkl}(0) = C_{ijkl} + {}^{*}C_{ijkl}(0),$$

and also,

$$\begin{aligned}
D'_{ijkl}(t) &= {}^{*}C'_{ijkl}(t), \\
\Longrightarrow \quad D_{ijkl}(t) &= {}^{*}C_{ijkl}(t) + \mathrm{constant}_{ijkl}, \\
\Longrightarrow \quad D_{ijkl}(t) &= {}^{*}C_{ijkl}(t) + C_{ijkl} \qquad \forall t.
\end{aligned}$$

In fact Johnson and Tessler in [24, equation 20.7] write,

$${}^{*}C_{ijkl}(t) := \sum_{m=1}^{N_D} C_{ijkl}^{*,m} e^{-t/\tau_m} \qquad \text{for } t \geq 0,$$

where $C_{ijkl}^{*,1}, C_{ijkl}^{*,2}, \ldots, C_{ijkl}^{*,N_D}$ are temporally constant fourth-order tensors. (In fact this is a slight generalization of Johnson and Tessler's expression, but no matter.)

Comparing terms once again with our Dirichlet-Prony series representation of $\underset{\sim}{\boldsymbol{D}}$ we see that simultaneously we must have,

$$\begin{aligned}
D_{ijkl}(t) &= C_{ijkl} + \sum_{m=1}^{N_D} C_{ijkl}^{*,m} e^{-t/\tau_m}, \\
D_{ijkl}(t) &= (D_{ijkl})_0 + \sum_{m=1}^{N_D} (D_{ijkl})_m e^{-t/\tau_m},
\end{aligned}$$

and so we infer that,

$$(D_{ijkl})_0 \equiv C_{ijkl} \qquad \text{and} \qquad (D_{ijkl})_m \equiv C_{ijkl}^{*,m}$$

for each m.

3.3 Multidimensional internal variables

It is of interest to derive "ODE" representations of these hereditary constitutive relationships in terms of internal strain variables. To begin we set,

$$ {}^*\varepsilon_{kl}^m(t) := e^{-t/\tau_m}\varepsilon_{kl}(0) + \int_0^t e^{-(t-s)/\tau_m}\frac{\partial \varepsilon_{kl}(s)}{\partial s}\,ds = \varepsilon_{kl}(t) - \frac{1}{\tau_m}\int_0^t e^{-(t-s)/\tau_m}\varepsilon_{kl}(s)\,ds, $$

and then by differentiating we obtain,

$$ \frac{\partial\ {}^*\varepsilon_{kl}^m(t)}{\partial t} = \frac{\partial \varepsilon_{kl}(t)}{\partial t} - \frac{1}{\tau_m}\left[e^{-t/\tau_m}\varepsilon_{kl}(0) + \int_0^t e^{-(t-s)/\tau_m}\frac{\partial \varepsilon_{kl}(s)}{\partial s}\,ds\right]. $$

Substituting for the integral now yields the family of differential equations governing the evolution of the internal strain variables:

$$ \frac{\partial\ {}^*\varepsilon_{kl}^m(t)}{\partial t} + \frac{{}^*\varepsilon_{kl}^m(t)}{\tau_m} = \frac{\partial \varepsilon_{kl}(t)}{\partial t}. $$

For the stresses we note that,

$$ \sigma_{ij}(t) = C_{ijkl}\varepsilon_{kl}(t) + \sum_{m=1}^{N_D} C_{ijkl}^{*,m}\ {}^*\varepsilon_{kl}^m(t). $$

We can also derive ODE's for the internal stresses defined by,

$$ {}^*\sigma_{ij}^m(t) := C_{ijkl}^{*,m}\ {}^*\varepsilon_{kl}^m(t), $$

which gives,

$$ \sigma_{ij}(t) = C_{ijkl}\varepsilon_{kl}(t) + \sum_{m=1}^{N_D} {}^*\sigma_{ij}^m(t). $$

In this case the evolution equations are,

$$ \frac{\partial\ {}^*\sigma_{ij}^m(t)}{\partial t} + \frac{{}^*\sigma_{ij}^m(t)}{\tau_m} = C_{ijkl}^{*,m}\frac{\partial \varepsilon_{kl}(t)}{\partial t}. $$

3.4 Reversed multidimensional internal variables

From our consideration of the Maxwell models we define the "reversed" internal strain tensors ${}_*\boldsymbol{\varepsilon}^m(t)$ via,

$$ {}_*\varepsilon_{kl}^m(t) := \varepsilon_{kl}(t) - {}^*\varepsilon_{kl}^m(t), $$

and of course it follows that,

$$ {}_*\varepsilon_{kl}^m(t) = \frac{1}{\tau_m}\int_0^t e^{-(t-s)/\tau_m}\varepsilon_{kl}(s)\,ds. $$

Then,

$$\frac{\partial\, {}_*\varepsilon^m_{kl}(t)}{\partial t} = \frac{{}^*\varepsilon^m_{kl}(t)}{\tau_m},$$

similar to before, and also,

$$\frac{\partial\, {}_*\varepsilon^m_{kl}(t)}{\partial t} + \frac{{}_*\varepsilon^m_{kl}(t)}{\tau_m} = \frac{\varepsilon_{kl}(t)}{\tau_m}.$$

These are clear analogues of the results for the scalar case given earlier.

Chapter 4

Model problems in viscoelasticity: Volterra formulations

4.1 Viscodynamics

To obtain the governing equations for the dynamic response of a viscoelastic body one uses Newton's second law to relate the stress field $\boldsymbol{\sigma}$ and the forces $\boldsymbol{f}$ and $\boldsymbol{g}$ to the acceleration, or **inertia**, of the body $\mathcal{G}$. This process is familiar from linear elasticity theory and gives, with boundary and initial data, the following. For $i = 1, \ldots, n$:

$$\left.\begin{array}{rcll}
\varrho u_i'' - \sigma_{ij,j} & = & f_i & \text{in } \Omega \times \mathcal{I}, \\
u_i & = & 0 & \text{in } \Gamma_D \times \mathcal{I}, \\
\sigma_{ij}\widehat{n}_j & = & g_i & \text{in } \Gamma_N \times \mathcal{I}, \\
u_i(\boldsymbol{x}, 0) & = & u_{i0} & \text{in } \Omega, \\
u_i'(\boldsymbol{x}, 0) & = & u_{i1} & \text{in } \Omega.
\end{array}\right\} \qquad (\mathbf{4}.1)$$

Here: repeated indices imply summation; $\mathcal{I} := (0, T)$ is a time interval; ϱ is the mass-density of $\mathcal{G}$; and, $\widehat{\boldsymbol{n}} := (\widehat{n}_i)_{i=1}^n$ is the unit outward directed normal to Γ_N.

Using (**2**.2) to substitute for the stress one arrives at the Partial Differential Volterra (PDV) problem: find $\boldsymbol{u}$ such that

$$\varrho u_i''(t) - \left(D_{ijkl}(0)\varepsilon_{kl}(\boldsymbol{u}(t))\right)_{,j} = f_i(t) - \int_0^t \left(\frac{\partial D_{ijkl}(t-s)}{\partial s}\varepsilon_{kl}(\boldsymbol{u}(s))\right)_{,j} ds,$$

in $\Omega \times \mathcal{I}$ with the indicated initial-boundary data. With an appropriate definition of A and $B(t, s)$ this is clearly a realization of the abstract problem (**1**.3). Note that it is "safe" to use the Dirichlet-Prony series (**2**.6) or modified power law (**2**.10) in this problem, but we may not use the power law (**2**.9) directly because we cannot then interpret $\underline{\boldsymbol{D}}(0)$.

In terms of existence, uniqueness and stability of solutions this problem has been studied in [10, 11, 34]. Numerical schemes are given in [13, 60, 39, 42].

One could also use the fractional calculus model to substitute for $\underline{\sigma}$ in Newton's second law. This will yield a PDV equation of the form

$$\varrho u_i''(t) - \left(D^{(0)}_{ijkl}\varepsilon_{kl}(\boldsymbol{u}(t))\right)_{,j} = f_i(t) + \frac{D^{(1)}_{ijkl}}{\Gamma(1-\alpha)}\frac{\partial}{\partial t}\int_0^t (t-s)^{-\alpha}\varepsilon_{kl}(\boldsymbol{u}(s))\,ds.$$

On the other hand one could use (**2**.1) and then arrive at

$$\varrho u_i''(t) - \left(D_{ijkl}(t)\varepsilon_{kl}(\boldsymbol{u}(0))\right)_{,j} = f_i(t) + \int_0^t \left(D_{ijkl}(t-s)\varepsilon_{kl}(\boldsymbol{u}'(s))\right)_{,j}\,ds.$$

Note that $\boldsymbol{u}$ does not occur as a natural "unknown" in this problem and so it is possible to replace $\boldsymbol{u}$ with $\boldsymbol{u}'$ and arrive at the alternative problem: find $\boldsymbol{u}$ such that

$$\varrho u_i'(t) + \int_0^t \left(D_{ijkl}(t-s)\varepsilon_{kl}(\boldsymbol{u}(s))\right)_{,j}\,ds = f_i(t) - \left(D_{ijkl}(t)\varepsilon_{kl}(\boldsymbol{u}_0)\right)_{,j},$$

which makes sense if $\boldsymbol{u}_0$ is smooth enough. The initial datum for this problem is now $\boldsymbol{u}(0) = \boldsymbol{u}_1$. Properties of the solution of these type of problems are studied in [11, 34] and numerical analysis is given in [35, 32].

However, one must resist the temptation to interpret this as a parabolic problem for, in general, it is not. To see this use the power law (**2**.9) with (**2**.5) to obtain (with $\varrho = 1$ and $\underline{\boldsymbol{D}}$ not $\boldsymbol{x}$ dependent for simplicity):

$$u_i'(t) + D_{ijkl}\int_0^t (t-s)^{-p}(\varepsilon_{kl}(\boldsymbol{u}(s))_{,j}\,ds = \tilde{f}_i(t), \tag{4.2}$$

where $\tilde{f}$ now incorporates the additional term in $\boldsymbol{u}_0$. In the case $p = \frac{1}{2}$ we find that the operator I defined by,

$$Iw(t) := \frac{1}{\sqrt{\pi}}\int_0^t (t-s)^{-\frac{1}{2}}w(s)\,ds$$

has the property,

$$I^2w(t) \equiv I(Iw)(t) = \int_0^t w(s)\,ds,$$

and so may be regarded as the **square root** of the definite integral operator. Applying $\partial_t^{\frac{1}{2}}$ to both sides of (**4**.2) in the case $p = \frac{1}{2}$ we arrive at

$$\left(\frac{\partial}{\partial t}\right)^{\frac{3}{2}} u_i(t) + \sqrt{\pi}D_{ijkl}(\varepsilon_{kl}(\boldsymbol{u}(t))_{,j} = \partial_t^{\frac{1}{2}}\tilde{f}_i(t).$$

This equation is **half way** between being parabolic and hyperbolic. Similar manipulations are also possible in the case $p \neq \frac{1}{2}$, with the final time derivative being of order between 1 and 2. Numerical methods for fractional order differential equations are studied in [41, 31, 12].

For more detail on these type of problems see [37], as well as the other papers in that collection.

4.2 Viscostatics

Recall that the classical linear elasticity equations are "derived" from Newton's second law (**4**.1) by dropping the inertia term $\varrho\boldsymbol{u}''$. This corresponds to modelling the problem at times long after the load has been applied when the transient response has died out, and results in a very well-known elliptic problem. A similar approach can be adopted for viscoelastic response although this time it is a true approximation since the resulting problem is not time independent due to the persistence of the Volterra term. It seems that this approximation can be useful when: the inertia term is negligible, which may occur when the load is smoothly and slowly applied, or even when the applied load is such that the dynamics of the response is dwarfed by the forced response; or when it is the long-time **creep** response that is of interest. Since the time dependence persists we refer to the resulting problems as modelling **quasistatic** viscoelastic response.

The governing equations for these type of problem are obtained from (**4**.1) by setting $\varrho\boldsymbol{u}''(t) := 0$, setting $\mathcal{J} := [0,T]$ and discarding the initial data. Thus, for $i = 1,\dots,n$ we have,

$$\left.\begin{array}{rcll} -\sigma_{ij,j} &=& f_i & \text{in } \Omega\times\mathcal{J}, \\ u_i &=& 0 & \text{in } \Gamma_D\times\mathcal{J}, \\ \sigma_{ij}\widehat{n}_j &=& g_i & \text{in } \Gamma_N\times\mathcal{J}, \end{array}\right\} \tag{4.3}$$

which are turned into differential equation problems for $\boldsymbol{u}$ by substituting for the stress using either of (**2**.1) or (**2**.2). These give respectively the PDV problems: find $\boldsymbol{u}$ such that for each $i \in \{1,\dots,n\}$,

$$-\int_0^t \left(D_{ijkl}(t-s)\varepsilon_{kl}(\boldsymbol{u}'(s))\right)_{,j} ds = f_i(t) + \left(D_{ijkl}(t)\varepsilon_{kl}(\boldsymbol{u}(0))\right)_{,j},$$

and

$$-\left(D_{ijkl}(0)\varepsilon_{kl}(\boldsymbol{u}(t))\right)_{,j} = f_i(t) - \int_0^t \left(\frac{\partial D_{ijkl}(t-s)}{\partial s}\varepsilon_{kl}(\boldsymbol{u}(s))\right)_{,j} ds.$$

The first of these is essentially a Volterra first-kind equation for $\boldsymbol{u}'$, while the second is a second-kind equation for $\boldsymbol{u}$. In both cases one obtains $\boldsymbol{u}(0)$ by solving a linear elasticity problem.

Numerical schemes and *a priori* error estimates were first provided for both of these problems in [45]. Later and for the second-kind problem only, the estimates were improved (in terms of the size of the error constant) in [44]. These latter results depend on by-passing Gronwall's inequality and using more sensitive comparison results to obtain sharp data-stability estimates. These estimates have now been generalized in [57]. Also for the second-kind problem, *a posteriori* error estimates for a space-time finite element discretization of a model problem have been given in [48] and [52]. These results are based on the error estimates in [47] and are currently being generalized to the multidimensional problem described above in [51], [54], [55] and [56].

4.3 Non-Fickian diffusion

This section is not directly relevant to the Seed Project, but it is another illustration of where viscoelastic effects play an important role, and therefore need to be modelled accurately. We include this material for interest only.

In classical diffusion theory the gradient of the concentration u of an active agent (the penetrant) diffusing through a carrier medium is related to the mass flux by Fick's law: $\boldsymbol{J} = -\lambda\nabla u$, where λ is the diffusivity of the carrier substance. Conservation of mass then demands that $u' = -\nabla \cdot \boldsymbol{J}$ which yields the familiar heat equation,

$$u'(t) = \nabla \cdot \lambda \nabla u.$$

If we define $M(t)$ as the total mass of penetrant absorbed by the carrier per unit area at time t then it is well known (from similarity solutions) that $M(t) \sim t^{\frac{1}{2}}$ for Fickian diffusion.

Diffusion in rubbery polymers, those well above their glass transition temperature (GTT), is according to Durning in [14] adequately described by Fick's law, but the situation is much more complicated for glassy polymers, those near but above their GTT. As the penetrant moves through the polymer it can force a phase change and so leave behind it the polymer carrier in its rubbery state. The stiffness and relaxation properties of the polymer change abruptly by orders of magnitude across this phase change (see for example [18]), and as a result a differential stress is set up across the penetrant boundary. Moreover, because the carrier is viscoelastic this stress is described by a hereditary constitutive law and this behaviour provides a mechanism for the observed non-Fickian effects. Workers in the field make the following very rough classification.

Case I diffusion: standard Fickian diffusion where $M(t) \sim t^{\frac{1}{2}}$, applies to polymers in the rubbery state high above the GTT.

Case II diffusion: non-Fickian diffusion, $M(t) \sim t^{\alpha}$ where $\frac{1}{2} < \alpha \leq 1$, applies to glassy polymers near to but above the GTT.

There is also a "Super Case II" category corresponding to $\alpha > 1$, see [6]. For Case II sharp fronts (rather like shocks) may appear as the penetrant diffuses through the carrier. This front moves initially at a constant speed and then slows down, [8], and this explains why $M(t)$ is almost linear, and thus $M'(t)$—the rate of absorbtion—is almost constant. By contrast $M'(t)$ for Case I is, in the words of Cox in [8], "delta-function-like", and this property of glassy polymers has an interesting application in the area of controlled drug delivery products. Cox gives a nice example.

An active agent (the drug) is embedded into a polymer through which it cannot diffuse. This may for example be a tablet which is to be swallowed. When the carrier is invaded by a solvent, such as digestive fluid, the drug can then diffuse out of the polymer through the solvent in a non-Fickian way. Since $M'(t)$ is almost constant, this allows a controlled, constant-rate delivery of the drug to the body for several hours.

The polymer doesn't have to be a tablet. In fact, according to Cohen and White in [6] (who also describe other applications of non-Fickian diffusion), such "smart" pharmaceutical products can be designed to be "swallowed, smelled, surgically implanted, rubbed on, taped on, strapped on", and can in effect be applied to any part of the body. There is an extensive literature on this science and in addition to those already cited we refer also to [20, 7, 15].

To get a flavour of the mathematical modelling that these researchers employ we borrow from [5] and consider the modelling of one-space dimensional diffusion through a glassy polymer. Our development yields a linear model, but it is unlikely that this will reproduce the sharp fronts characteristic of polymer diffusion. The references cited deal with realistic nonlinear models.

To account for the differential stress set up at the penetrant front Fick's law is modified to include a stress dependence in the following way:

$$J = -(\lambda u_x + \kappa \sigma_x).$$

Here u is the concentration, λ the usual (Fickian) diffusion constant, and κ is a proportionality constant. Conservation of mass again demands that $u' = -J_x$ and this gives

$$u' = \lambda u_{xx} + \kappa \sigma_{xx}.$$

The stress is viscoelastic and the usual approach is to adopt the Maxwell model, given earlier in (**2**.3), with the assumption that u depends linearly on strain rate ε' (this is in order to get true Case II behaviour—see [9]). Thus,

$$\frac{\partial \sigma}{\partial t} + \frac{\sigma}{\tau} = \mu u,$$

where μ is a proportionality constant. In the nonlinear theory the dependence of τ on u is crucial, but here we shall assume that τ is constant. Integrating we get,

$$\sigma(t) = \mu e^{-t/\tau} u(0) + \mu \int_0^t e^{-(t-s)/\tau} u(s)\, ds.$$

Eliminating the stress from the transport equation and using mass conservation gives the single differential-Volterra equation,

$$u'(t) = \lambda u_{xx} + \kappa \mu e^{-t/\tau} u_{xx}(0) + \kappa \mu \int_0^t e^{-(t-s)/\tau} u_{xx}(s)\, ds.$$

Assuming for simplicity that $u(0) = 0$ we can generalize this to a multidimensional model and obtain the PDV equation,

$$u'(t) = \nabla \cdot \lambda \nabla u + \nabla \cdot \left(\kappa \nabla \int_0^t \mu e^{-(t-s)/\tau} u(s)\, ds \right).$$

This is a concrete realization of the abstract problem (**1**.2).

Equations of this nature have been studied in [36] and [21], and some numerical analysis is given in [60, 3, 58, 40, 59]. Also, *a priori* and *a posteriori* error estimates for a finite element discretization of a scalar prototype ODE with memory, of the type that arises after spatial finite element semi-discretization of this problem, are provided in [46].

Chapter 5

Model problems in viscoelasticity: internal variable formulations

5.1 Overview and notation

In this chapter we look at how to formulate viscoelasticity problems described by internal variables. Once again our main reference is Johnson and Tessler, [24]. Before starting we introduce some notation. For problems posed in $\mathbb{R}^n$ we set,

$$\boldsymbol{H}^m(\Omega) := H^m(\Omega) \times \cdots \times H^m(\Omega) \qquad (n \text{ times}),$$

and, bearing in mind the notation introduced earlier in equations (**4**.1) and (**4**.3), we set,

$$H := \Big\{ \boldsymbol{v} \in \boldsymbol{H}^1(\Omega) : \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma_D \Big\}.$$

Also, thinking in terms of weak formulations we also set,

$$(\underline{\boldsymbol{\sigma}}, \boldsymbol{\varepsilon}) := \int_\Omega \sigma_{ij} \varepsilon_{ij}(\boldsymbol{v}) \, d\Omega,$$

and,

$$L(t; \boldsymbol{v}) := \int_\Omega \boldsymbol{v} \cdot \boldsymbol{f}(t) \, d\Omega + \oint_{\Gamma_N} \boldsymbol{v} \cdot \boldsymbol{g}(t) \, d\Gamma.$$

We begin with quasistatic problems.

5.2 Viscostatics: weak formulations

We recall first the governing (quasi-equilibrium) equations for quasistatic viscoelasticity introduced previously in (**4**.3). Then, using the fundamental Green's formula,

$$-\int_\Omega w_{,j} v \, d\Omega = \int_\Omega w v_{,j} \, d\Omega - \oint_{\partial\Omega} w v \widehat{n}_j \, d\Gamma,$$

and also the symmetry of the stress tensor to see that,

$$\sigma_{ij}v_{i,j} = \frac{1}{2}\Big(\sigma_{ij}v_{i,j} + \sigma_{ji}v_{j,i}\Big) = \sigma_{ij}\varepsilon_{ij}(\boldsymbol{v}),$$

we then have for $\boldsymbol{v} \in H$ that,

$$\int_\Omega f_i v_i \, d\Omega = -\int_\Omega \sigma_{ij,j} v_i \, d\Omega = \int_\Omega \sigma_{ij} v_{i,j} \, d\Omega - \oint_{\Gamma_N} \sigma_{ij}\widehat{n}_j v_i \, d\Gamma.$$

This implies,

$$\int_\Omega \sigma_{ij}\varepsilon_{ij}(\boldsymbol{v}) \, d\Omega = \int_\Omega \boldsymbol{v} \cdot \boldsymbol{f}(t) \, d\Omega + \oint_{\Gamma_N} \boldsymbol{v} \cdot \boldsymbol{g}(t) \, d\Gamma.$$

Recalling the definitions made earlier we can write this more compactly as,

$$(\underline{\boldsymbol{\sigma}}, \underline{\boldsymbol{\varepsilon}}(\boldsymbol{v})) = L(t; \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in H.$$

This virtual work statement (weak formulation) is the standard “jumping off point” for finite element approximations (to be considered later).

There now seems to be (at least) two ways to proceed. The first is to take the route described by Johnson and Tessler in [24], we describe this first.

Route 1

We substitute for the stress using the multidimensional internal variables described earlier, where we recall that,

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl}(\boldsymbol{u}(t)) + \sum_{m=1}^{N_D} C_{ijkl}^{*,m}\, {}^{*}\varepsilon_{kl}^{m}(t),$$

which gives the representation,

$$(\underline{\boldsymbol{\sigma}}, \underline{\boldsymbol{\varepsilon}}(\boldsymbol{v})) = \int_\Omega C_{ijkl}\varepsilon_{kl}(\boldsymbol{u}(t))\varepsilon_{ij}(\boldsymbol{v}) \, d\Omega + \sum_{m=1}^{N_D} \int_\Omega C_{ijkl}^{*,m}\, {}^{*}\varepsilon_{kl}^{m}(t)\varepsilon_{ij}(\boldsymbol{v}) \, d\Omega.$$

We now introduce internal displacements ${}^{*}\boldsymbol{u}^m$ such that,

$${}^{*}\varepsilon_{kl}^{m}({}^{*}\boldsymbol{u}^m) := \frac{1}{2}\left(\frac{\partial\, {}^{*}u_k^m}{\partial x_l} + \frac{\partial\, {}^{*}u_l^m}{\partial x_k}\right),$$

and this gives the virtual work statement in the form,

$$a(\boldsymbol{u}(t), \boldsymbol{v}) = L(t; \boldsymbol{v}) - \sum_{m=1}^{N_D} {}^{*}a^m({}^{*}\boldsymbol{u}^m(t), \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in H.$$

Here,

$$\begin{aligned} a(\boldsymbol{u}(t), \boldsymbol{v}) &:= \int_\Omega C_{ijkl}\varepsilon_{kl}(\boldsymbol{u}(t))\varepsilon_{ij}(\boldsymbol{v})\,d\Omega, \\ {}^*a^m({}^*\boldsymbol{u}^m(t), \boldsymbol{v}) &:= \int_\Omega C^{*,m}_{ijkl}\,{}^*\varepsilon^m_{kl}({}^*\boldsymbol{u}^m(t))\varepsilon_{ij}(\boldsymbol{v})\,d\Omega. \end{aligned}$$

The internal strains satisfy,

$$\frac{\partial\,{}^*\varepsilon^m_{kl}}{\partial t} + \frac{{}^*\varepsilon^m_{kl}}{\tau_m} = \frac{\partial \varepsilon_{kl}}{\partial t},$$

and it is important to realise that these hold at a point in space and so are effectively ordinary differential equations.

Thinking in terms of the reversed internal variables,

$${}_*\underset{\sim}{\boldsymbol{\varepsilon}}^m := \underset{\sim}{\boldsymbol{\varepsilon}} - {}^*\underset{\sim}{\boldsymbol{\varepsilon}}^m,$$

we may replace these ordinary differential equations by,

$$\frac{\partial\,{}_*\varepsilon^m_{kl}}{\partial t} + \frac{{}_*\varepsilon^m_{kl}}{\tau_m} = \frac{\varepsilon_{kl}}{\tau_m},$$

and solve these instead. This may make more sense because we know $\underset{\sim}{\boldsymbol{\varepsilon}}$ but not $\partial\underset{\sim}{\boldsymbol{\varepsilon}}/\partial t$.

Thus we arrive at a statement of the problem.

Find $\boldsymbol{u} : \mathcal{J} \to H$ such that,

$$a(\boldsymbol{u}(t), \boldsymbol{v}) = L(t; \boldsymbol{v}) - \sum_{m=1}^{N_D} {}^*a^m({}^*\boldsymbol{u}^m(t), \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in H.$$

Here the ${}^*\boldsymbol{u}^m$ are implicitly (but not uniquely) defined by the tensors ${}^*\underset{\sim}{\boldsymbol{\varepsilon}}^m$ which in turn are given by,

$$\frac{\partial\,{}^*\varepsilon^m_{kl}}{\partial t} + \frac{{}^*\varepsilon^m_{kl}}{\tau_m} = \frac{\partial \varepsilon_{kl}}{\partial t} \qquad \text{or} \qquad \frac{\partial\,{}_*\varepsilon^m_{kl}}{\partial t} + \frac{{}_*\varepsilon^m_{kl}}{\tau_m} = \frac{\varepsilon_{kl}}{\tau_m}$$

with ${}_*\varepsilon^m_{kl} = \varepsilon_{kl} - {}^*\varepsilon^m_{kl}$.

We now describe the second route.

Route 2

On the other hand we note that there is no need to introduce the internal displacements ${}^*\boldsymbol{u}^m$ at all. If we recall that,

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl}(\boldsymbol{u}) + \sum_{m=1}^{N_D} {}^*\sigma^m_{ij},$$

where,

$$^*\sigma_{ij}^m := C_{ijkl}^{*,m}\ ^*\varepsilon_{kl}^m,$$

then the virtual work statement takes the form,

$$a(\boldsymbol{u}(t),\boldsymbol{v}) = L(t;\boldsymbol{v}) - \sum_{m=1}^{N_D} (\ ^*\underline{\boldsymbol{\sigma}}^m(t), \underline{\boldsymbol{\varepsilon}}(\boldsymbol{v})) \qquad \forall \boldsymbol{v} \in H.$$

This generates a different problem statement.

Find $\boldsymbol{u} : \mathcal{J} \to H$ such that,

$$a(\boldsymbol{u}(t),\boldsymbol{v}) = L(t;\boldsymbol{v}) - \sum_{m=1}^{N_D} (\ ^*\underline{\boldsymbol{\sigma}}^m(t), \underline{\boldsymbol{\varepsilon}}(\boldsymbol{v})) \qquad \forall \boldsymbol{v} \in H.$$

Here the internal stresses $^*\underline{\boldsymbol{\sigma}}^m$ are given by,

$$\frac{\partial\ ^*\sigma_{ij}^m}{\partial t} + \frac{^*\sigma_{ij}^m}{\tau_m} = C_{ijkl}^{*,m}\frac{\partial \varepsilon_{kl}}{\partial t},$$

or: by $^*\sigma_{ij}^m = C_{ijkl}^{*,m}\ ^*\varepsilon_{kl}^m$ where,

$$\frac{\partial\ ^*\varepsilon_{kl}^m}{\partial t} + \frac{^*\varepsilon_{kl}^m}{\tau_m} = \frac{\partial \varepsilon_{kl}}{\partial t} \qquad \text{or} \qquad \frac{\partial\ _*\varepsilon_{kl}^m}{\partial t} + \frac{_*\varepsilon_{kl}^m}{\tau_m} = \frac{\varepsilon_{kl}}{\tau_m}$$

with $_*\varepsilon_{kl}^m = \varepsilon_{kl} -\ ^*\varepsilon_{kl}^m$.

5.3 Viscodynamics: weak formulations

Given the quasistatic problem it is now straightforward to pose the viscodynamic problem. Hence,

$$(\boldsymbol{u}_{tt}(t),\boldsymbol{v}) + a(\boldsymbol{u}(t),\boldsymbol{v}) = L(t;\boldsymbol{v}) - \sum_{m=1}^{N_D}\ ^*a^m(\ ^*\boldsymbol{u}^m(t),\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in H.$$

The internal strain and displacement variables are exactly as given earlier. This research project is not concerned with dynamic problems and so we leave the development of this area to another time.

Part II

Numerical Algorithms for stress analysis

Chapter 6

Quasistatic internal variable problems

6.1 Overview and notation

In this chapter we consider some of the possibilities for forming numerical algorithms for the quasistatic problem modelled by internal variables. We consider semi- and fully discrete schemes, in that order, and detail the computational implementation. We follow here **Route 2** as detailed earlier in Chapter 5.

First we establish our notation. If W is a function space then for a vector-valued function $\boldsymbol{w} := (w_i)_{i=1}^{n}$, for which $w_i \in W$ for each i, we shall write $\boldsymbol{w} \in \boldsymbol{W}$ where,

$$\boldsymbol{W} := W \times \cdots \times W \qquad (n \text{ times}).$$

Moreover, if $\underline{\boldsymbol{w}} := (w_{ij})_{i,j=1}^{n}$ is a second-order tensor-valued function for which $w_{ij} \in W$ for each pair (i, j), then we set,

$$\begin{aligned}
\underline{\boldsymbol{W}} &= \boldsymbol{W} \times \cdots \times \boldsymbol{W} \qquad (n \text{ times}),\\
&:= \overbrace{\Big(W \times \cdots \times W\Big)}^{n \text{ times}} \times \cdots \times \overbrace{\Big(W \times \cdots \times W\Big)}^{n \text{ times}} \qquad (n \text{ times}),
\end{aligned}$$

and write $\underline{\boldsymbol{w}} \in \underline{\boldsymbol{W}}$.

6.2 Semidiscrete finite element approximation

We consider the **Route 2** problem in the following form. Find $\boldsymbol{u} \in L_\infty(\mathcal{J}; H)$ and, for each m, ${}^*\underline{\boldsymbol{\sigma}}^m \in W_\infty^1(\mathcal{J}; \underline{\boldsymbol{L}}_2(\Omega))$ such that,

$$a(\boldsymbol{u}(t), \boldsymbol{v}) = L(t; \boldsymbol{v}) - \sum_{m=1}^{N_D} \big(\, {}^*\underline{\boldsymbol{\sigma}}^m(t), \underline{\boldsymbol{\varepsilon}}(\boldsymbol{v})\big) \qquad \forall \boldsymbol{v} \in H,$$

and, for each m,

$$\left(\frac{\partial\, {}^{*}\underline{\boldsymbol{\sigma}}^m}{\partial t} + \frac{{}^{*}\underline{\boldsymbol{\sigma}}^m}{\tau_m}, \underline{\boldsymbol{w}}\right) = \left(\frac{\partial \widehat{\underline{\boldsymbol{\sigma}}}^m}{\partial t}, \underline{\boldsymbol{w}}\right) \qquad \forall \underline{\boldsymbol{w}} \in \underline{\boldsymbol{L}}_2(\Omega),$$

where we define,

$$\widehat{\sigma}_{ij}^m := C_{ijkl}^{*,m} \varepsilon_{kl}(\boldsymbol{u}).$$

Alternatively, we might decide to work instead with the internal strains and then this last set of constitutive differential equations is replaced by,

$$\left(\frac{\partial\, {}^{*}\underline{\boldsymbol{\varepsilon}}^m}{\partial t} + \frac{{}^{*}\underline{\boldsymbol{\varepsilon}}^m}{\tau_m}, \underline{\boldsymbol{w}}\right) = \left(\frac{\partial \underline{\boldsymbol{\varepsilon}}(\boldsymbol{u})}{\partial t}, \underline{\boldsymbol{w}}\right) \qquad \forall \underline{\boldsymbol{w}} \in \underline{\boldsymbol{L}}_2(\Omega).$$

In this latter case of course we seek not only the ${}^{*}\underline{\boldsymbol{\sigma}}^m$ but the ${}^{*}\underline{\boldsymbol{\varepsilon}}^m \in W_\infty^1(\mathcal{J}; \underline{\boldsymbol{L}}_2(\Omega))$ also. (Note also that we need $\boldsymbol{u}$ to be time differentiable, although this requirement could be removed by employing the reversed internal variables introduced earlier in Chapter 2)

To form a notional semidiscrete approximation to this problem we firstly partition $\mathcal{J} := [0, T]$ into time intervals $\{\mathcal{J}_i\}_{i=1}^N$, each given by,

$$\mathcal{J}_i := (t_{i-1}, t_i) \qquad \text{with} \qquad k_i := t_i - t_{i-1}$$

denoting the time steps. Now, for each time interval $\mathcal{J}_i$ we partition Ω (in the usual way) into a family of disjoint triangles suitable for piecewise linear finite element approximation, and then let $\boldsymbol{U}$ denote the piecewise linear finite element approximation of $\boldsymbol{u}$. We denote the space of piecewise linear functions with respect to this mesh (i.e. during $\mathcal{J}_i$) as H_i, and we let Ω_j represent the j^{th} element (i.e. triangle) in this partition.

In a similar way we let ${}^{*}\underline{\boldsymbol{\varsigma}}^m$ denote the piecewise constant (on each element) approximation to ${}^{*}\underline{\boldsymbol{\sigma}}^m$, and denote the space of piecewise constant functions with respect to this mesh (i.e. during $\mathcal{J}_i$) as L_i.

In terms of the internal strains the resulting semidiscrete finite element approximation to this **Route 2** problem is: find $\boldsymbol{U} \in L_\infty(\mathcal{J}; H)$, satisfying $\boldsymbol{U}|_{\mathcal{J}_i} \in L_\infty(\mathcal{J}_i; H_i)$, and for each m, ${}^{*}\underline{\boldsymbol{\varsigma}}^m \in W_\infty^1(\mathcal{J}; \underline{\boldsymbol{L}}_2(\Omega))$, satisfying ${}^{*}\underline{\boldsymbol{\varsigma}}^m|_{\mathcal{J}_i} \in W_\infty^1(\mathcal{J}_i; \underline{\boldsymbol{L}}_i)$, such that for each $i = 1, 2, \ldots, N$ in turn,

$$a(\boldsymbol{U}(t), \boldsymbol{v}) = L(t; \boldsymbol{v}) - \sum_{m=1}^{N_D} \left({}^{*}\underline{\boldsymbol{\varsigma}}^m(t), \underline{\boldsymbol{\varepsilon}}(\boldsymbol{v})\right) \qquad \forall \boldsymbol{v} \in H_i,$$

where ${}^{*}\varsigma_{ij}^m := C_{ijkl}^{*,m}\, {}^{*}\epsilon_{kl}^m$, and the ${}^{*}\epsilon_{kl}^m$ are the finite element approximations of ${}^{*}\varepsilon_{kl}^m$ satisfying (in each $\mathcal{J}_i$),

$$\left(\frac{\partial\, {}^{*}\underline{\boldsymbol{\epsilon}}^m}{\partial t} + \frac{{}^{*}\underline{\boldsymbol{\epsilon}}^m}{\tau_m}, \underline{\boldsymbol{w}}\right) = \left(\frac{\partial \underline{\boldsymbol{\varepsilon}}(\boldsymbol{U})}{\partial t}, \underline{\boldsymbol{w}}\right) \qquad \forall \underline{\boldsymbol{w}} \in \underline{\boldsymbol{L}}_i.$$

6.3 Semidiscrete system equations

To represent a tensor-valued function in $\underline{\boldsymbol{L}}_i$ (for times $t \in \mathcal{J}_i$) we let $\{\phi_{ij}\}$ be a basis consisting of piecewise constant functions in the following way,

$$\phi_{ij} := \begin{cases} 1, & \text{for } \boldsymbol{x} \in \Omega_j = j^{\text{th}} \text{ element in the mesh,} \\ 0, & \text{elsewhere.} \end{cases}$$

Then, $\phi_{ij} \in L_i$ for each j and for an arbitrary $\underline{\boldsymbol{\theta}} \in \underline{\boldsymbol{L}}_i$ we have that,

$$\theta_{kl} = \sum_j \alpha_j \phi_{ij},$$

where α_j is the (constant) value of the component θ_{kl} on the element Ω_j.

In the expression

$$(\underline{\boldsymbol{\theta}}, \underline{\boldsymbol{w}}) \qquad \text{for } \underline{\boldsymbol{w}} \in \underline{\boldsymbol{L}}_i,$$

we now choose $\underline{\boldsymbol{w}}$ such that $w_{kl} = \phi_{ij}$ is the only non-zero component, then,

$$\begin{aligned} (\underline{\boldsymbol{\theta}}, \underline{\boldsymbol{w}}) &= \int_\Omega \theta_{kl} w_{kl}\, d\Omega = \int_{\Omega_j} \theta_{kl}\phi_{ij}\, d\Omega, \\ &= \operatorname{meas}(\Omega_j)\, \alpha_j. \end{aligned}$$

Picking $\underline{\boldsymbol{\theta}} := {}^*\underline{\boldsymbol{\epsilon}}^m$ we therefore find that,

$$\frac{1}{\tau_m}({}^*\underline{\boldsymbol{\epsilon}}^m, \underline{\boldsymbol{w}}) = \frac{\operatorname{meas}(\Omega_j)}{\tau_m}\alpha_j,$$

and

$$\left(\frac{\partial\, {}^*\underline{\boldsymbol{\epsilon}}^m}{\partial t}, \underline{\boldsymbol{w}}\right) = \operatorname{meas}(\Omega_j)\, \dot{\alpha}_j,$$

Here of course α_j is the spatially constant value of ${}^*\varepsilon^m_{kl}$ on Ω_j and $\dot{\alpha}_j := \dfrac{d\alpha_j}{dt}$.

Also, since $\underline{\boldsymbol{\varepsilon}}(\boldsymbol{U}) \in \underline{\boldsymbol{L}}_i$, we let $\underline{\boldsymbol{\beta}}_j(\boldsymbol{U})$ denote the spatially constant value of $\underline{\boldsymbol{\varepsilon}}(\boldsymbol{U})$ on Ω_j and then we may write,

$$\underline{\boldsymbol{\varepsilon}}(\boldsymbol{U}) = \sum_j \underline{\boldsymbol{\beta}}_j(\boldsymbol{U})\phi_{ij}.$$

Then, with the same choice for $\underline{\boldsymbol{w}}$ we have,

$$\begin{aligned} \left(\frac{\partial \underline{\boldsymbol{\varepsilon}}(\boldsymbol{U})}{\partial t}, \underline{\boldsymbol{w}}\right) &= \sum_j \int_{\Omega_j} \left(\frac{d\underline{\boldsymbol{\beta}}_j(\boldsymbol{U})}{dt}\right)_{kl} w_{kl}\, d\Omega = \int_{\Omega_j} \left(\frac{d\underline{\boldsymbol{\beta}}_j(\boldsymbol{U})}{dt}\right)_{kl} d\Omega, \\ &= \operatorname{meas}(\Omega_j)\, \frac{d\beta_j(\boldsymbol{U})}{dt}. \end{aligned}$$

Thus, the finite element approximation of the internal variable constitutive equations become,

$$\frac{d\alpha_j}{dt} + \frac{\alpha_j}{\tau_m} = \frac{d\beta_j(\boldsymbol{U})}{dt},$$

where,

α_j is the spatially constant value of ${}^*\epsilon^m_{kl}$ on each Ω_j, for every pair (k,l) and for each m;

β_j is the spatially constant value of $\varepsilon_{kl}(\boldsymbol{U})$ on Ω_j, for each pair (k,l).

To obtain the system representation of the equilibrium equations we note first of all that,

$$\begin{aligned} a(\boldsymbol{U}(t),\boldsymbol{v}) &\longrightarrow \mathbf{AU}, \\ L(t;\boldsymbol{v}) &\longrightarrow \mathbf{F}(t), \end{aligned}$$

in a straightforward way, where $\mathbf{A}$ is the elastic stiffness matrix and $\mathbf{F}$ is the load vector. It now remains to consider the viscous term.

At this point A.R. Johnson would introduce the ${}^*\boldsymbol{U}^m$ internal displacement variables and use the "**Route 1**" formulation with the evolution equations written pointwise in terms of the ${}^*\boldsymbol{U}^m$. Strictly, one should integrate these equations over Ω since derivatives of ${}^*\boldsymbol{U}^m$ are involved—this seems to introduce unwanted complications. We consider constructing the viscous terms directly.

Let $\{\psi_{il}\}_{l>0}$ be the canonical piecewise linear basis with respect to the mesh during $\mathcal{J}_i$, and let $\{\boldsymbol{\psi}_l\}_{l>0}$ be the resulting basis for H_i. In the viscous term

$$(\,{}^*\underline{\varsigma}^m(t), \underline{\varepsilon}(\boldsymbol{v})),$$

we choose $\boldsymbol{v} = \boldsymbol{\psi}_l$ for some l and let $S = \mathrm{supp}(\boldsymbol{\psi}_l)$, then:

$$\begin{aligned} (\,{}^*\underline{\varsigma}^m(t), \underline{\varepsilon}(\boldsymbol{v})) &= \sum_{\Omega_l \subset S} \int_{\Omega_l} C^{*,m}_{ijkl}\; {}^*\epsilon^m_{kl} \varepsilon_{ij}(\boldsymbol{\psi}_l)\, d\Omega_l, \\ &= \sum_{\Omega_l \subset S} C^{*,m}_{ijkl}\; {}^*\epsilon^m_{kl}|_{\Omega_l} \varepsilon_{ij}(\boldsymbol{\psi}_l)|_{\Omega_l} \mathrm{meas}(\Omega_l)\;, \\ &=: \mathsf{F}_l(t). \end{aligned}$$

Defining the vector of viscous forces ${}^*\mathbf{F}^m(t) := (\mathsf{F}_l(t))_{l>0}$ in this way the system equations take the form,

$$\mathbf{AU}(t) = \mathbf{F}(t) - \sum_{m=1}^{N_D} {}^*\mathbf{F}^m(t),$$

$$\frac{d\alpha_j}{dt} + \frac{\alpha_j}{\tau_m} = \frac{d\beta_j(\boldsymbol{U})}{dt} \qquad \text{for all } \alpha_j := {}^*\epsilon^m_{kl} \text{ on each } \Omega_j.$$

Note that ${}^*\mathbf{F}^m$ is in principle easier to calculate than the stiffness contribution ${}^*\mathbf{A}^m\;{}^*\mathbf{U}^m$ described by Johnson and Tessler in [24], and requires less memory. However, direct access to the isoparametric "builder" routines in the finite element code is required.

6.4 Equivalence of semidiscrete formulations

We now aim to show that the semidiscrete internal variable formulation described above is completely equivalent (it has the same solution) to the semidiscrete Volterra formulation that we have used previously (in for example, [45], [44], [54]).

Firstly, solving the ODE for α_j in terms of $\beta_j(\boldsymbol{U})$ and recalling the definitions of these terms we have that,

$$^*\epsilon_{kl}^m(t) = \varepsilon_{kl}(\boldsymbol{U}(t)) - \frac{1}{\tau_m}\int_0^t e^{-(t-s)/\tau_m}\varepsilon_{kl}(\boldsymbol{U}(s))\,ds$$

(except on element boundaries), which is expected. Thus it still makes sense to define,

$$_*\epsilon_{kl}^m(t) := \varepsilon_{kl}(\boldsymbol{U}(t)) - {}^*\epsilon_{kl}^m(t) = \frac{1}{\tau_m}\int_0^t e^{-(t-s)/\tau_m}\varepsilon_{kl}(\boldsymbol{U}(s))\,ds,$$

and we still have,

$$\frac{\partial\, _*\epsilon_{kl}^m}{\partial t} = \frac{^*\epsilon_{kl}^m}{\tau_m}.$$

Note that by choosing $\underline{\boldsymbol{w}} \in \underline{\boldsymbol{L}}_i$ such that $w_{ij} = C_{klij}^{*,m}\varepsilon_{kl}(\boldsymbol{v})$ for any $\boldsymbol{v} \in H_i$ we have,

$$\begin{aligned}
(\,^*\underline{\boldsymbol{\epsilon}}^m, \underline{\boldsymbol{w}}) &= \int_\Omega {}^*\epsilon_{ij}^m w_{ij}\,d\Omega = \int_\Omega {}^*\epsilon_{ij}^m C_{klij}^{*,m}\varepsilon_{kl}(\boldsymbol{v})\,d\Omega,\\
&= \int_\Omega {}^*\varsigma_{kl}^m\varepsilon_{kl}(\boldsymbol{v})\,d\Omega = (\,^*\underline{\boldsymbol{\varsigma}}^m, \underline{\boldsymbol{\varepsilon}}(\boldsymbol{v})),\\
&= \tau_m\left(\frac{\partial\underline{\boldsymbol{\varepsilon}}(\boldsymbol{U})}{\partial t} - \frac{\partial\,^*\underline{\boldsymbol{\epsilon}}^m}{\partial t}, \underline{\boldsymbol{w}}\right),\\
&= \tau_m\left(\frac{\partial\,_*\underline{\boldsymbol{\epsilon}}^m}{\partial t}, \underline{\boldsymbol{w}}\right),
\end{aligned}$$

where we are now working in terms of the reversed internal strains. Combining equations then gives,

$$a(\boldsymbol{U}(t), \boldsymbol{v}) = L(t; \boldsymbol{v}) - \sum_{m=1}^{N_D}\tau_m\left(\frac{\partial\,_*\underline{\boldsymbol{\epsilon}}^m}{\partial t}, \underline{\boldsymbol{w}}\right) \qquad \forall \boldsymbol{v} \in H_i,$$

and where $w_{ij} := C_{klij}^{*,m}\varepsilon_{kl}(\boldsymbol{v})$.

Now, bearing in mind the combined equation given above we note the following. For $w_{kl} := C_{ijkl}^{*,m}\varepsilon_{ij}(\boldsymbol{v})$ and $\boldsymbol{v} \in H_i$,

$$\begin{aligned}
\tau_m&\left(\frac{\partial\,_*\underline{\boldsymbol{\epsilon}}^m}{\partial t}, \underline{\boldsymbol{w}}\right) = (\,^*\underline{\boldsymbol{\epsilon}}^m, \underline{\boldsymbol{w}}) = \int_\Omega C_{ijkl}^{*,m}\,{}^*\epsilon_{kl}^m(t)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega,\\
&= \int_\Omega C_{ijkl}^{*,m}\left(\varepsilon_{kl}(\boldsymbol{U}(t)) - \frac{1}{\tau_m}\int_0^t e^{-(t-s)/\tau_m}\varepsilon_{kl}(\boldsymbol{U}(s))\,ds\right)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega,\\
&= \int_\Omega C_{ijkl}^{*,m}\varepsilon_{kl}(\boldsymbol{U}(t))\varepsilon_{ij}(\boldsymbol{v})\,d\Omega - \frac{1}{\tau_m}\int_0^t\int_\Omega C_{ijkl}^{*,m}e^{-(t-s)/\tau_m}\varepsilon_{kl}(\boldsymbol{U}(s))\varepsilon_{ij}(\boldsymbol{v})\,d\Omega ds.
\end{aligned}$$

Substituting this into the "combined equation" shown above then gives,

$$\begin{aligned} a(\boldsymbol{U}(t), \boldsymbol{v}) &= L(t; \boldsymbol{v}) - \sum_{m=1}^{N_D} \int_\Omega C^{*,m}_{ijkl} \varepsilon_{kl}(\boldsymbol{U}(t)) \varepsilon_{ij}(\boldsymbol{v}) \, d\Omega \\ &\quad + \sum_{m=1}^{N_D} \frac{1}{\tau_m} \int_0^t \int_\Omega C^{*,m}_{ijkl} e^{-(t-s)/\tau_m} \varepsilon_{kl}(\boldsymbol{U}(s)) \varepsilon_{ij}(\boldsymbol{v}) \, d\Omega ds, \end{aligned}$$

which, using the definition of $a(\,\cdot\,,\,\cdot\,)$ from earlier, is the same as,

$$\begin{aligned} &\int_\Omega \left(C_{ijkl} + \sum_{m=1}^{N_D} C^{*,m}_{ijkl} \right) \varepsilon_{kl}(\boldsymbol{U}(t)) \varepsilon_{ij}(\boldsymbol{v}) \, d\Omega = L(t; \boldsymbol{v}) \\ &\qquad + \int_0^t \int_\Omega \frac{\partial}{\partial s} \left(\sum_{m=1}^{N_D} C^{*,m}_{ijkl} e^{-(t-s)/\tau_m} \right) \varepsilon_{kl}(\boldsymbol{U}(s)) \varepsilon_{ij}(\boldsymbol{v}) \, d\Omega ds, \end{aligned}$$

or, changing back to the "D" notation for the relaxation function,

$$\int_\Omega D_{ijkl}(0) \varepsilon_{kl}(\boldsymbol{U}(t)) \varepsilon_{ij}(\boldsymbol{v}) \, d\Omega = L(t; \boldsymbol{v}) + \int_0^t \int_\Omega \frac{\partial D_{ijkl}(t-s)}{\partial s} \varepsilon_{kl}(\boldsymbol{U}(s)) \varepsilon_{ij}(\boldsymbol{v}) \, d\Omega ds.$$

This is precisely the Volterra formulation we were aiming at. (Once again, this is not surprising—but we have to be certain.) More recently (in e.g. [54]) we write this in the form,

$$A(\boldsymbol{U}(t), \boldsymbol{v}) = L(t; \boldsymbol{v}) + \int_0^t B(t, s; \boldsymbol{U}(s), \boldsymbol{v}) \, ds \qquad \forall \boldsymbol{v} \in H_i,$$

and we will return to this later. Now we consider a fully discrete approximation.

6.5 Fully discrete internal variables

In view of the equivalence just demonstrated it would be a neat trick to find a time discretization of the evolution equations that is equivalent to our "Volterra approximation", and then the existing error estimates given in [54, 55] can be applied. It is doubtful whether this can be done. However, the solutions given by the two schemes should at least be close, and our next task is to attempt to find an expression connecting them.

Let γ_j be the spatially constant value of ${}_*\epsilon^m_{kl}$, on each Ω_j, for every pair (k, l) in turn, and for each m in turn. Then by exactly the same reasoning as used above to obtain the ODE's for the α_j's we can also arrive at the family,

$$\frac{d\gamma_j}{dt} + \frac{\gamma_j}{\tau_m} = \frac{\beta_j(\boldsymbol{U})}{\tau_m}.$$

(Note that the time derivative of $\beta_j(\boldsymbol{U})$ is **not** required.) This is still semidiscrete. To discretize in time we will use piecewise constant approximations to γ_j and $\boldsymbol{U}$ denoted during each time interval $\mathcal{J}_q = (t_{q-1}, t_q)$ by $\widehat{\gamma}_q$ and $\widehat{\boldsymbol{U}}_q$. A simple finite element approximation of the ODE's given above then yields, for $q = 1, 2, \ldots$,

$$\widehat{\gamma}_q - \widehat{\gamma}_{q-1} + \frac{k_q \widehat{\gamma}_q}{\tau_m} = \frac{k_q \beta_j(\widehat{\boldsymbol{U}}_q)}{\tau_m} \qquad \text{(in each } \Omega_j \text{ etc.)},$$

subject to $\widehat{\gamma}_0 = {}_*\epsilon^m_{kl}(0) = 0$.

Setting $r_q := k_q/\tau_m$ we have in general that,

$$(1 + r_q)\widehat{\gamma}_q = r_q \beta_j(\widehat{\boldsymbol{U}}_q) + \widehat{\gamma}_{q-1},$$

and so the first few solutions are given by,

$$\begin{aligned}
\widehat{\gamma}_1 &= \frac{r_1 \beta_j(\widehat{\boldsymbol{U}}_1)}{1 + r_1}, \\
\widehat{\gamma}_2 &= \frac{r_2 \beta_j(\widehat{\boldsymbol{U}}_2)}{1 + r_2} + \frac{r_1 \beta_j(\widehat{\boldsymbol{U}}_1)}{(1 + r_2)(1 + r_1)}, \\
\widehat{\gamma}_3 &= \frac{r_3 \beta_j(\widehat{\boldsymbol{U}}_3)}{1 + r_3} + \frac{r_2 \beta_j(\widehat{\boldsymbol{U}}_2)}{(1 + r_3)(1 + r_2)} + \frac{r_1 \beta_j(\widehat{\boldsymbol{U}}_1)}{(1 + r_3)(1 + r_2)(1 + r_1)}.
\end{aligned}$$

It is not hard to see that in general,

$$\widehat{\gamma}_q = \sum_{p=1}^{q} \frac{r_p \beta_j(\widehat{\boldsymbol{U}}_p)}{\prod_{s=p}^{q} (1 + r_s)},$$

and that this holds for every $\widehat{\gamma}_q \approx \gamma_j$ on $\mathcal{J}_q$ and where $\gamma_j = {}_*\epsilon^m_{kl}$ on Ω_j. (Note that this assumes the space meshes are constant in time.) In tensor form we can write this as,

$$({}_*\underline{\boldsymbol{\epsilon}}^m_j)_q = \sum_{p=1}^{q} \frac{r_p \underline{\boldsymbol{\beta}}_j(\widehat{\boldsymbol{U}}_p)}{\prod_{s=p}^{q} (1 + r_s)} \qquad \text{on every element } \Omega_j,$$

and where $({}_*\underline{\boldsymbol{\epsilon}}^m_j)_q$ denotes the fully discrete approximation to ${}_*\underline{\boldsymbol{\epsilon}}^m_j$ during times in $\mathcal{J}_q$. Dropping the subscript j we therefore have that everywhere on the mesh,

$$({}_*\underline{\boldsymbol{\epsilon}}^m)_q = \sum_{p=1}^{q} \frac{r_p \underline{\boldsymbol{\varepsilon}}(\widehat{\boldsymbol{U}}_p)}{\prod_{s=p}^{q} (1 + r_s)} \qquad \text{except at element boundaries.}$$

The question now arises as to how we determine the $\widehat{\boldsymbol{U}}_p$. For this we use a finite element discretization (in time) of the semidiscrete equilibrium equations. Using the "combined equation" given earlier we then have, for all suitable space-time dependent test functions $\boldsymbol{v}$ that,

$$\sum_q \int_{t_{q-1}}^{t_q} a(\widehat{\boldsymbol{U}}_q(t), \boldsymbol{v}(t))\, dt = \int_0^T L(t; \boldsymbol{v}(t))\, dt - \sum_q \int_{t_{q-1}}^{t_q} \sum_{m=1}^{N_D} (\underline{\boldsymbol{\varepsilon}}(\widehat{\boldsymbol{U}}_q) - ({}_*\underline{\boldsymbol{\epsilon}}^m)_q, \underline{\boldsymbol{w}}),$$

where we used the semidiscrete equation to write, for $w_{ij} = C^{*,m}_{klij}\varepsilon_{kl}(\boldsymbol{v})$,

$$\tau_m\left(\frac{\partial\,{}_*\boldsymbol{\epsilon}^m}{\partial t}, \boldsymbol{w}\right) = (\boldsymbol{\varepsilon}(\widehat{\boldsymbol{U}}(t)) - {}_*\boldsymbol{\epsilon}^m(t), \boldsymbol{w})$$

and then used the finite element approximations.

Choosing the test function $\boldsymbol{v}$ such that it is non-zero only in $\mathcal{J}_q$ then results in,

$$a(\widehat{\boldsymbol{U}}_q, \boldsymbol{v}) = \frac{1}{k_q}\int_{t_{q-1}}^{t_q} L(t;\boldsymbol{v})\,dt - \sum_{m=1}^{N_D}(\boldsymbol{\varepsilon}(\widehat{\boldsymbol{U}}_q) - ({}_*\boldsymbol{\epsilon}^m)_q, \boldsymbol{w})$$

for all $\boldsymbol{v} \in H_q$ and where $w_{ij} = C^{*,m}_{klij}\varepsilon_{kl}(\boldsymbol{v})$. In this case we can write,

$$(\boldsymbol{\varepsilon}(\widehat{\boldsymbol{U}}_q) - ({}_*\boldsymbol{\epsilon}^m)_q, \boldsymbol{w}) = \int_\Omega C^{*,m}_{ijkl}\varepsilon_{kl}(\widehat{\boldsymbol{U}}_q)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega - \int_\Omega C^{*,m}_{ijkl}({}_*\epsilon^m_{kl})_q\varepsilon_{ij}(\boldsymbol{v})\,d\Omega,$$

and so the fully discrete solutions satisfy,

$$\begin{aligned}\int_\Omega\left(C_{ijkl} + \sum_{m=1}^{N_D} C^{*,m}_{ijkl}\right)\varepsilon_{kl}(\widehat{\boldsymbol{U}}_q)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega \quad &= \quad \frac{1}{k_q}\int_{t_{q-1}}^{t_q} L(t;\boldsymbol{v})\,dt \\ &\quad + \sum_{m=1}^{N_D}\int_\Omega C^{*,m}_{ijkl}({}_*\epsilon^m_{kl})_q\varepsilon_{ij}(\boldsymbol{v})\,d\Omega.\end{aligned}$$

Using our explicit solution for $({}_*\boldsymbol{\epsilon}^m)_q$ we then get,

$$\begin{aligned}\int_\Omega\left(C_{ijkl} + \sum_{m=1}^{N_D} C^{*,m}_{ijkl}\right)\varepsilon_{kl}(\widehat{\boldsymbol{U}}_q)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega &= \frac{1}{k_q}\int_{t_{q-1}}^{t_q} L(t;\boldsymbol{v})\,dt \\ &\quad + \sum_{m=1}^{N_D}\int_\Omega C^{*,m}_{ijkl}\sum_{p=1}^{q}\frac{r_p\varepsilon_{kl}(\widehat{\boldsymbol{U}}_p)}{\prod_{s=p}^{q}(1+r_s)}\varepsilon_{ij}(\boldsymbol{v})\,d\Omega, \\ &= \frac{1}{k_q}\int_{t_{q-1}}^{t_q} L(t;\boldsymbol{v})\,dt \\ &\quad + \sum_{m=1}^{N_D}\sum_{p=1}^{q}\left(\frac{r_p}{\prod_{s=p}^{q}(1+r_s)}\right)\int_\Omega C^{*,m}_{ijkl}\varepsilon_{kl}(\widehat{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega.\end{aligned}$$

Tidying this up, we find that the fully discrete displacements satisfy,

$$\begin{aligned}\int_\Omega D_{ijkl}(0)\varepsilon_{kl}(\widehat{\boldsymbol{U}}_q)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega &= \frac{1}{k_q}\int_{t_{q-1}}^{t_q} L(t;\boldsymbol{v})\,dt \\ &\quad + \sum_{p=1}^{q}\sum_{m=1}^{N_D}\left(\frac{r_p}{\prod_{s=p}^{q}(1+r_s)}\right)\int_\Omega C^{*,m}_{ijkl}\varepsilon_{kl}(\widehat{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega.\end{aligned}$$

Now we effect a comparison between this fully discrete solution and the one produced by the “Volterra formulation”.

6.6 Comparison of fully discrete formulations

Setting $\hat{t}_p := \min\{t, t_p\}$ we can discretize the semidiscrete Volterra problem using a piecewise constant temporal approximation to the displacements to get,

$$A(\widetilde{\boldsymbol{U}}_q, \boldsymbol{v}) = \frac{1}{k_q}\int_{t_{q-1}}^{t_q} L(t; \boldsymbol{v})\, dt + \frac{1}{k_q}\int_{t_{q-1}}^{t_q} \sum_{p=1}^{q} \int_{t_{p-1}}^{\hat{t}_p} B(t, s; \widetilde{\boldsymbol{U}}_p, \boldsymbol{v})\, dsdt \qquad \forall \boldsymbol{v} \in H_q.$$

For the term on the far right we have,

$$\begin{aligned}
&\frac{1}{k_q}\int_{t_{q-1}}^{t_q} \sum_{p=1}^{q} \int_{t_{p-1}}^{\hat{t}_p} \Big(\int_\Omega \frac{\partial D_{ijkl}(t-s)}{\partial s} \varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\, d\Omega\Big)\, dsdt \\
&\qquad = \frac{1}{k_q}\int_{t_{q-1}}^{t_q} \sum_{p=1}^{q} \int_{t_{p-1}}^{\hat{t}_p} \left(\int_\Omega \sum_{m=1}^{N_D} \frac{C^{*,m}_{ijkl}}{\tau_m} e^{-(t-s)/\tau_m} \varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\, d\Omega\right) dsdt, \\
&\qquad = \sum_{m=1}^{N_D} \sum_{p=1}^{q} \frac{1}{\tau_m k_q} \left(\int_{t_{q-1}}^{t_q} \int_{t_{p-1}}^{\hat{t}_p} e^{-(t-s)/\tau_m}\, dsdt\right) \int_\Omega C^{*,m}_{ijkl} \varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\, d\Omega.
\end{aligned}$$

Now, for $p = q$ we have,

$$\int_{t_{q-1}}^{t_q} \int_{t_{q-1}}^{t} e^{-(t-s)/\tau_m}\, dsdt = \tau_m k_q + \tau_m^2 \Big(e^{-k_q/\tau_m} - 1\Big),$$

while for $p < q$,

$$\int_{t_{q-1}}^{t_q} \int_{t_{p-1}}^{t_p} e^{-(t-s)/\tau_m}\, dsdt = \tau_m^2 e^{-(t_q - t_{p-1})/\tau_m} \Big(e^{k_q/\tau_m} - 1\Big)\Big(e^{k_p/\tau_m} - 1\Big).$$

Using these we then have that the fully discrete approximation from the Volterra formulation satisfies,

$$\begin{aligned}
A(\widetilde{\boldsymbol{U}}_q, \boldsymbol{v}) &= \frac{1}{k_q}\int_{t_{q-1}}^{t_q} L(t; \boldsymbol{v})\, dt + \sum_{m=1}^{N_D} \left(1 + \frac{e^{-k_q/\tau_m} - 1}{r_q}\right) \int_\Omega C^{*,m}_{ijkl} \varepsilon_{kl}(\widetilde{\boldsymbol{U}}_q)\varepsilon_{ij}(\boldsymbol{v})\, d\Omega \\
&+ \sum_{m=1}^{N_D} \sum_{p=1}^{q-1} \left(\frac{e^{-(t_q - t_{p-1})/\tau_m}}{r_q} \Big(e^{k_q/\tau_m} - 1\Big)\Big(e^{k_p/\tau_m} - 1\Big)\right) \int_\Omega C^{*,m}_{ijkl} \varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\, d\Omega.
\end{aligned}$$

Using,

$$e^{(t_{p-1} - t_q)/\tau_m} = \frac{1}{e^{r_p}\, e^{r_{p+1}} \cdots e^{r_q}},$$

and expanding to first order gives,

$$\frac{e^{-(t_q - t_{p-1})/\tau_m}}{r_q} \Big(e^{k_q/\tau_m} - 1\Big)\Big(e^{k_p/\tau_m} - 1\Big) = \frac{r_p}{\prod_{s=p}^{q}(1 + r_s)} + O(r_p r_q).$$

A similar approximation leads also to,

$$1 + \frac{e^{-k_q/\tau_m} - 1}{r_q} = \frac{r_q - 1}{r_q} + \frac{1}{r_q e^{r_q}} = \frac{r_q}{1 + r_q} + O(r_q^2).$$

Thus, the fully discrete approximation from the Volterra formulation also satisfies,

$$\int_\Omega D_{ijkl}(0)\varepsilon_{kl}(\widetilde{\boldsymbol{U}}_q)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega = \frac{1}{k_q}\int_{t_{q-1}}^{t_q} L(t;\boldsymbol{v})\,dt$$
$$+\sum_{m=1}^{N_D}\sum_{p=1}^{q}\left(\frac{r_p}{\prod_{s=p}^{q}(1+r_s)}\right)\int_\Omega C_{ijkl}^{*,m}\varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega$$
$$+\sum_{m=1}^{N_D}\sum_{p=1}^{q}E(r_q,r_p)\int_\Omega C_{ijkl}^{*,m}\varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega,$$

where $E(r_q,r_p)=O(r_qr_p)$ is the truncation error resulting from the approximation of the exponentials.

We can now obtain an "error equation" connecting these two numerical solutions as follows,

$$\begin{aligned} A(\widetilde{\boldsymbol{U}}_q-\widehat{\boldsymbol{U}}_q,\boldsymbol{v}) &= \sum_{m=1}^{N_D}\sum_{p=1}^{q}\left(\frac{r_p}{\prod_{s=p}^{q}(1+r_s)}\right)\int_\Omega C_{ijkl}^{*,m}\varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p-\widehat{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega \\ &\quad+\sum_{m=1}^{N_D}\sum_{p=1}^{q}E(r_q,r_p)\int_\Omega C_{ijkl}^{*,m}\varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega. \end{aligned}$$

Note also that by regarding the integrals as approximations of the rational functions we could easily replace $\widetilde{\boldsymbol{U}}_p$ with $-\widehat{\boldsymbol{U}}_p$ in the last term, and make an adjustment to the first term on the right. That is, writing,

$$\frac{r_p}{\prod_{s=p}^{q}(1+r_s)}=\frac{1}{\tau_m k_q}\int_{t_{q-1}}^{t_q}\int_{t_{p-1}}^{\hat{t}_p} e^{-(t-s)/\tau_m}\,dsdt - E(r_qr_p),$$

we have also,

$$A(\widetilde{\boldsymbol{U}}_q-\widehat{\boldsymbol{U}}_q,\boldsymbol{v}) =$$
$$\sum_{m=1}^{N_D}\sum_{p=1}^{q}\left(\frac{1}{\tau_m k_q}\int_{t_{q-1}}^{t_q}\int_{t_{p-1}}^{\hat{t}_p} e^{-(t-s)/\tau_m}\,dsdt\right)\int_\Omega C_{ijkl}^{*,m}\varepsilon_{kl}(\widetilde{\boldsymbol{U}}_p-\widehat{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega$$
$$-\sum_{m=1}^{N_D}\sum_{p=1}^{q}E(r_q,r_p)\int_\Omega C_{ijkl}^{*,m}\varepsilon_{kl}(\widehat{\boldsymbol{U}}_p)\varepsilon_{ij}(\boldsymbol{v})\,d\Omega.$$

The point about these results is that we can now measure the distance between the two different numerical solutions. This opens up the possibility of using our error estimates for the Volterra formulation to construct an adaptive solver for the internal variable formulations already used by Dr. Johnson. However, in the next chapter we take a simpler and more direct route toward an adaptive solver built on the concept of internal stress variables.

Chapter 7

The numerical algorithm

7.1 Introduction

In this chapter we fix on the numerical scheme proposed in [54] for which both *a priori* and *a posteriori* error estimates have been derived there and in [55, 56]. This numerical scheme is based on the Volterra formulation of the problem and discretizes the Volterra integral directly. The result is that the entire solution history plays a role in the *a posteriori* error estimate.

With this in mind, below we pay particular attention to marrying the theoretical error estimates derived for the Volterra formulation with a practical implementation drawing on internal variables. We view this as especially important since the error bounds contain a potentially troublesome (and expensive) term (denoted in the next chapter by $\mathcal{E}_V$). This term arises directly from the residual of the finite element solution associated with the Volterra integral on **de-refined** meshes. It seems inconceivable that the effect of this term can be anything other than to degrade the quality of the error estimate, and it seems possible that an internal variable formulation could be used to remove it completely with comparatively minor additional computational expense.

We say more about the *a posteriori* error estimate in the next chapter where we also show some adaptive solutions. Here we outline the finite element discretization of the problem and quote from [54] the *a priori* error estimate, and then give an alternative numerical implementation to that in [54] by invoking internal stress variables. We term the resulting scheme a hybrid Volterra–internal-variable formulation since the numerical scheme and error analyses in [54, 55, 56] are all framed in the context of the Volterra formulation, while the material below draws on the internal variables discussed in the previous chapters and is ultimately based on the work of Johnson *et al.* in [24, 25, 23].

After outlining the solution algorithm we design some test problems for which we know the exact solutions and then compute some numerical solutions, and their errors, to demonstrate consistency, spatial convergence and temporal convergence.

7.2 Finite element discretization

To discretize the problem (**4**.3) with (**2**.2) using the finite element method we first identify a suitable test space H to capture the spatial regularity, and then take the scalar product of (**4**.3) with a test function and integrate by parts over the domain Ω for an arbitrary time $t \in \mathcal{J}$. Later we then allow the test function to be time dependent and integrate through time as well to prepare for a space-time finite element discretization.

Since the displacements are vector valued we first define for $s = 0, 1, 2, \ldots$ the product Hilbert spaces,

$$\Big(\boldsymbol{H}^s(\Omega), (\,\cdot\,,\,\cdot\,)_s\Big) := \Big(H^s(\Omega), (\,\cdot\,,\,\cdot\,)_{H^s(\Omega)}\Big) \times \cdots \times \Big(H^s(\Omega), (\,\cdot\,,\,\cdot\,)_{H^s(\Omega)}\Big) \qquad (n \text{ times}),$$

where the inner products are given by,

$$(\boldsymbol{w}, \boldsymbol{v})_s := \sum_{i=1}^{n} (w_i, v_i)_{H^s(\Omega)}$$

for all $\boldsymbol{w}$, $\boldsymbol{v} \in \boldsymbol{H}^s(\Omega)$. These spaces have the natural norms $\|\cdot\|_s := \sqrt{(\,\cdot\,,\,\cdot\,)_s}$ and, of course, $\boldsymbol{L}_2(\Omega) \equiv \boldsymbol{H}^0(\Omega)$. Also, and as is usual for time dependent problems, for a Banach space $(B, \|\cdot\|_B)$ we define the $L_p(0,t;B)$ norms by,

$$\|\boldsymbol{v}\|_{L_p(0,t;B)} := \left(\int_0^t \|\boldsymbol{v}(s)\|_B^p \, ds\right)^{\frac{1}{p}}$$

for $t \in \mathcal{J}$, and with the obvious "ess sup" modification when $p = \infty$.

Using the essential boundary condition we now define the (spatial) test space,

$$H := \Big\{\boldsymbol{v} \in \boldsymbol{H}^1(\Omega) : \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma_D\Big\}, \tag{7.1}$$

and (see e.g. [54] for details) arrive at the "semi-weak" problem: find $\boldsymbol{u} \in L_\infty(\mathcal{J}; H)$ such that,

$$A(\boldsymbol{u}(t), \boldsymbol{v}) = L(t; \boldsymbol{v}) + \int_0^t B(t, s; \boldsymbol{u}(s), \boldsymbol{v})\, ds \qquad \forall \boldsymbol{v} \in H, \text{ a.e. in } \mathcal{J}. \tag{7.2}$$

We call this problem "semi weak" because later we will integrate over $\mathcal{J}$ also to obtain a "fully weak" formulation. (These terms are just labels and no mathematical significance should be attached to the "semi" and "fully" qualifiers.) Here for the triangle,

$$\mathcal{T} := \{(t,s) \in \mathcal{J} \times \mathcal{J} : 0 \leq s \leq t \in \mathcal{J}\},$$

the bilinear forms $A : H \times H \to \mathbb{R}$ and $B : \mathcal{T} \times H \times H \to \mathbb{R}$ are defined by,

$$A(\boldsymbol{w}, \boldsymbol{v}) := \int_\Omega D_{ijkl}(0)\varepsilon_{kl}(\boldsymbol{w})\varepsilon_{ij}(\boldsymbol{v})\, d\Omega, \tag{7.3}$$

$$B(t, s; \boldsymbol{w}, \boldsymbol{v}) := \int_\Omega \frac{\partial D_{ijkl}(t-s)}{\partial s}\varepsilon_{kl}(\boldsymbol{w})\varepsilon_{ij}(\boldsymbol{v})\, d\Omega, \tag{7.4}$$

for all $\boldsymbol{w}$, $\boldsymbol{v} \in H$, and $L : \mathcal{J} \times H \to \mathbb{R}$ is a time dependent linear form defined by,

$$L(t; \boldsymbol{v}) := \int_\Omega \boldsymbol{v} \cdot \boldsymbol{f}(t)\, d\Omega + \oint_{\Gamma_N} \boldsymbol{v} \cdot \boldsymbol{g}(t)\, d\Gamma. \tag{7.5}$$

For full details on this formulation along with some (rather standard) assumptions we refer to [54]. In particular we ensure that $(H, A(\,\cdot\,,\,\cdot\,))$ is a Hilbert space with (elastic) **energy norm** $\|\cdot\|_H := \sqrt{A(\,\cdot\,,\,\cdot\,)}$ and dual H', and then from [57] we have the stability estimate,

$$\|\boldsymbol{u}\|_{L_p(0,t;H)} \leq S(t)\|L\|_{L_p(0,t;H')}. \tag{7.6}$$

In this estimate $S : \mathcal{J} \to [0, \infty)$ is a stability factor which for isotropic problems can be expressed in terms of the eigenvalues of the stress relaxation tensor (or matrix) $\underline{\boldsymbol{D}}$. This factor is of crucial importance since it appears in the *a posteriori* error estimates and governs the rate at which the discretization errors can grow with time.

To carry out a space-time discretization we need now to allow the test function to be time dependent and then integrate the weak form over the time interval $\mathcal{J}$. Thus we arrive at the "fully-weak" formulation of this problem as: find $\boldsymbol{u} \in L_\infty(\mathcal{J}; H)$ such that,

$$a(\boldsymbol{u}, \boldsymbol{v}) = l(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in L_1(\mathcal{J}; H), \tag{7.7}$$

where:

$$a(\boldsymbol{u}, \boldsymbol{v}) := \int_0^T A(\boldsymbol{u}(t), \boldsymbol{v}(t))\, dt - \int_0^T \int_0^t B(t, s; \boldsymbol{u}(s), \boldsymbol{v}(t))\, ds dt, \tag{7.8}$$

$$l(\boldsymbol{v}) := \int_0^T L(t; \boldsymbol{v}(t))\, dt. \tag{7.9}$$

This equation is the starting point for the space-time finite element discretization of the problem, and to this we now turn. We firstly partition $\mathcal{J}$ into N time intervals $\{\mathcal{J}_i\}_{i=1}^N$, where $\mathcal{J}_i := (t_{i-1}, t_i)$, of lengths $k_i := t_i - t_{i-1} > 0$ and such that,

$$\mathcal{J} \equiv \overline{\mathcal{J}_1 \cup \mathcal{J}_2 \cup \cdots \cup \mathcal{J}_N},$$

so that $t_0 = 0$ and $t_N = T$. We use k to denote the piecewise constant function such that $k|_{\mathcal{J}_i} := k_i$.

For each of these $\mathcal{J}_i$ we construct on $\overline{\Omega}$ (in the usual way) a triangular/tetrahedral space-mesh of M_i elements and denote the domain Ω with this mesh by Ω_i. Element j of Ω_i will be denoted Ω_{ij} and we set,

$$h_{ij} := \mathrm{diam}(\Omega_{ij}),$$

and use h to denote the piecewise constant mesh function given by $h|_{\Omega_{ij}} := h_{ij}$. We also use h_i to denote the mesh function at times $t \in \mathcal{J}_i$ given by $h_i|_{\Omega_{ij}} := h_{ij}$, and use these notations to see that $h|_{\mathcal{J}_i} := h_i$. We need to assume that arbitrary partitions of $\Omega \times \mathcal{J}$ of this nature always exist because in an adaptive solution the $\{\mathcal{J}_i\}$ and $\{\Omega_i\}$ are of course not known in advance.

Having now broken the prismatic domain $\Omega \times \mathcal{J}$ into the laminae (or "slabs") $\Omega \times \mathcal{J}_i$, indexed by the time levels $i \in \mathbb{N}(1, N)$, we define for each Ω_i the semidiscrete (spatial) finite element spaces,

$$H_i := \Big\{ \boldsymbol{v} \in H \cap (C(\overline{\Omega}))^n : \boldsymbol{v} \text{ is linear on } \Omega_{ij} \text{ for each } j \in \mathbb{N}(1, M_i) \Big\}.$$

The space-time finite element spaces are now given by V_r where,

$$V_r := \Big\{ \boldsymbol{v} \in L_p(\mathcal{J}; H) : v|_{\mathcal{J}_i} \in \mathbb{P}_r(\mathcal{J}_i; H_i) \ \forall i \in \mathbb{N}(1, N) \Big\}.$$

Here $\mathbb{P}_r(\mathcal{J}_i; H_i)$ is the vector space of polynomials of degree at most r defined on $\mathcal{J}_i$ with coefficients in H_i. Note that our approximating functions in V_r are continuous in space but in general discontinuous at the knots $\{t_i\}_{i=1}^{N-1}$. These discontinuities allow the space-meshes to change with time.

We define also the set of internal edges (for triangular Ω_{ij} in $\mathbb{R}^2$), or faces (for tetrahedral Ω_{ij} in $\mathbb{R}^3$) in each Ω_i as,

$$\mathcal{L}_i := \Big\{ \ell \subset \Omega : \exists j \in \mathbb{N}(1, M_i) \text{ such that } \ell \text{ is an edge/face of } \Omega_{ij} \Big\},$$

and the set of edges, or faces, on the Neumann boundary Γ_N as,

$$\mathcal{F}_i := \Big\{ \ell \subset \Gamma_N : \exists j \in \mathbb{N}(1, M_i) \text{ such that } \ell \text{ is an edge/face of } \Omega_{ij} \Big\}.$$

We assume that there are respectively $N_{\mathcal{L}_i}$ and $N_{\mathcal{F}_i}$ such edges in the time interval $\mathcal{J}_i$. We also define,

$$\mathcal{H}_{ij} := \mathcal{L}_j \setminus (\mathcal{L}_i \cap \mathcal{L}_j) \qquad \text{for } 1 \leq j \leq i \leq N, \tag{7.10}$$

as the set of internal edges/faces belonging to $\mathcal{L}_j$ but not to $\mathcal{L}_i$. Note that $\mathcal{H}_{ii} \equiv \emptyset$, and that if we control our adaptivity and allow only nested refinements such that $H_{i-1} \subseteq H_i$ for $i \in \mathbb{N}(2, N)$, then $\mathcal{L}_i \cap \mathcal{L}_j = \mathcal{L}_j$ and $\mathcal{H}_{ij} \equiv \emptyset$.

Once we choose a value for r, which for us will be $r = 0$ or $r = 1$, we form the finite element approximation to (**7**.7) as: find $\boldsymbol{U} \in V_r$ such that,

$$a(\boldsymbol{U}, \boldsymbol{v}) = l(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in V_r, \tag{7.11}$$

and subtracting this from (**7**.7) gives the fundamentally important Galerkin "orthogonality" relationship:

$$a(\boldsymbol{u} - \boldsymbol{U}, \boldsymbol{v}) = 0 \qquad \forall \boldsymbol{v} \in V_r. \tag{7.12}$$

This property, when coupled to the strong data stability of an associated dual backward problem, is the basic building block in the error estimation technique developed by Johnson *et al.* in for example [16]. For our problem it is, however, of limited use since we do not have strong temporal stability of the solution.

Note that as the foregoing suggests the material in [54, 55, 56] is applicable to both two- and three-dimensional problems, although here we are concerned only with two-dimensions. Also, we take $r = 0$ in all that follows which corresponds to a piecewise constant temporal approximation. The numerical algorithm corresponding to piecewise linear time discretization is summarized in [54].

In [55] we give a detailed list of assumptions on the data and in particular on the approximation properties of the spaces $\{H_i\}$ and V_r. These again are rather standard and would be out of place here. Note also that we assume that $\underline{\boldsymbol{D}}$ is piecewise constant in space. We could easily allow the case where $\underline{\boldsymbol{D}}$ is piecewise smooth in space at the price of an extra term in the error estimate, but the piecewise constant case is more likely to arise in practical problems since most materials have piecewise constant properties.

The *a priori* error estimate for this problem is derived in [54]; it takes the following form.

Theorem 1 (*A priori* Galerkin energy-error estimate) *Under certain natural assumptions, and for approximation in V_r, for $r = 0, 1$, the Galerkin error $\boldsymbol{e} := \boldsymbol{u} - \boldsymbol{U}$ satisfies the a priori error estimate,*

$$\|\boldsymbol{u} - \boldsymbol{U}\|_{L_\infty(\mathcal{J};H)} \leq C(T) \left(\Pi_h \left\| hD^2\boldsymbol{u} \right\|_{L_\infty(\mathcal{J};\boldsymbol{L}_2(\Omega))} + \Pi_k \left\| k^{r+1} \frac{\partial^{r+1}\boldsymbol{u}}{\partial t^{r+1}} \right\|_{L_\infty(\mathcal{J};H)} \right),$$

where $C(T)$ is a constant. This estimate holds for $r = 1$ only if each k_q is small enough, and depends also on the ratios k_q/k_{q-1}.

Such a result is reassuring in that it guarantees convergence and also demonstrates the form required for the upper bound on the *a posteriori* estimate in order to guarantee robustness. However, from a software implementation standpoint the *a posteriori* error estimate is much more relevant since it can form the basis of an adaptive code. We consider adaptivity in the next chapter and below illustrate the convergence of the scheme with a few example exact solutions. First we detail the practical implementation of this scheme.

7.3 Internal variable formulation

Before we get to the numerical scheme in the next section it is useful first to re-write (**7**.7) in terms of internal stress variables. Recall first that the constitutive law is given by,

$$\boldsymbol{\sigma}(t) = \boldsymbol{D}(0)\boldsymbol{\varepsilon}(\boldsymbol{u}(t)) - \int_0^t \boldsymbol{D}_s(t-s)\boldsymbol{\varepsilon}(\boldsymbol{u}(s))\, ds$$

(neglecting $\boldsymbol{x}$ dependence), where,

$$\boldsymbol{\sigma} := \begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{\varepsilon} := \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{pmatrix}.$$

Also, we consider only isotropic materials and so write,

$$\boldsymbol{D}(t) = \lambda(t)\boldsymbol{I}_\lambda + \mu(t)\boldsymbol{I}_\mu,$$

where,

$$\boldsymbol{I}_\lambda := \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \boldsymbol{I}_\mu := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/2 \end{pmatrix},$$

and $\lambda(t)$, $\mu(t)$ are given by Prony series-type relaxation functions. For the stress we then have,

$$\boldsymbol{\sigma}(t) = \varsigma(t) - \boldsymbol{\Sigma}(\boldsymbol{u}; t),$$

where,

$$\begin{aligned} \varsigma(t) &:= \boldsymbol{D}(0)\boldsymbol{\varepsilon}(\boldsymbol{u}(t)) = \text{ instantaneous elastic stress}, \\ \boldsymbol{\Sigma}(\boldsymbol{u}; t) &:= \int_0^t \boldsymbol{D}_s(t-s)\boldsymbol{\varepsilon}(\boldsymbol{u}(s))\, ds = \text{ inherited viscous stress}. \end{aligned}$$

Setting $\boldsymbol{\lambda}_i := \lambda_i \boldsymbol{I}_\lambda$ and $\boldsymbol{\mu}_i := \mu_i \boldsymbol{I}_\mu$ and defining the internal stress variables,

$$\begin{aligned} \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{u}; t) &:= \boldsymbol{\lambda}_i l_i \int_0^t e^{-l_i(t-s)}\boldsymbol{\varepsilon}(\boldsymbol{u}(s))\, ds \\ \boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{u}; t) &:= \boldsymbol{\mu}_i m_i \int_0^t e^{-m_i(t-s)}\boldsymbol{\varepsilon}(\boldsymbol{u}(s))\, ds, \end{aligned}$$

we have,

$$\boldsymbol{\Sigma}(\boldsymbol{u}; t) = \int_0^t \boldsymbol{D}_s(t-s)\boldsymbol{\varepsilon}(\boldsymbol{u}(s))\, ds = \sum_{i=1}^{N_\lambda} \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{u}; t) + \sum_{i=1}^{N_\mu} \boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{u}; t).$$

Note that these internal variables satisfy evolution equations of the form,

$$\boldsymbol{\Sigma}'_{\lambda_i}(\boldsymbol{u}; t) + l_i \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{u}; t) = \boldsymbol{\lambda}_i l_i \boldsymbol{\varepsilon}(\boldsymbol{u}(t)),$$

and also that the following "update recurrences" apply,

$$\begin{aligned} \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{u}; t) &= e^{-l_i(t-\tau)}\boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{u}; \tau) + \boldsymbol{\lambda}_i l_i \int_\tau^t e^{-l_i(t-s)}\boldsymbol{\varepsilon}(\boldsymbol{u}(s))\, ds, \\ \boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{u}; t) &= e^{-m_i(t-\tau)}\boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{u}; \tau) + \boldsymbol{\mu}_i m_i \int_\tau^t e^{-m_i(t-s)}\boldsymbol{\varepsilon}(\boldsymbol{u}(s))\, ds. \end{aligned}$$

Using these new definitions with (**7**.8) and (**7**.4) we get an alternative representation of the history as,

$$\begin{aligned} \int_0^t B(t, s; \boldsymbol{u}(s), \boldsymbol{v})\, ds &= \int_\Omega \left(\int_0^t \boldsymbol{D}_s(t-s)\boldsymbol{\varepsilon}(\boldsymbol{u}(s))\, ds \right) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v})\, d\Omega, \\ &= \int_\Omega \boldsymbol{\Sigma}(\boldsymbol{u}; t) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v})\, d\Omega. \end{aligned}$$

The point to note here is that the history integral has now vanished and been replaced by a local (in time) term. We will use a similar approach below to form the numerical algorithm.

7.4 The numerical scheme

To obtain a practical scheme from (**7**.11) in the case $r = 0$ we choose $\boldsymbol{v}$ such that $\boldsymbol{v} = \boldsymbol{0}$ outside of the time interval $\mathcal{J}_q$, and then $\boldsymbol{v} \in H_q$ is a piecewise linear function of $\boldsymbol{x}$ during times $t \in \mathcal{J}_q$. Writing $\boldsymbol{U}_q := \boldsymbol{U}|_{\mathcal{J}_q}$ for the discrete solution restricted to this time interval equation (**7**.11) then becomes,

$$a(\boldsymbol{U}_q, \boldsymbol{v}) = l(\boldsymbol{v}) \qquad \forall \boldsymbol{v}|_{\mathcal{J}_q} \in H_q,\ \boldsymbol{v}|_{\mathcal{J}\setminus\mathcal{J}_q} = \boldsymbol{0},$$

for each $q = 1, 2, \ldots$. Here,

$$l(\boldsymbol{v}) = \int_{t_{q-1}}^{t_q} L(t; \boldsymbol{v})\, dt,$$

and,

$$a(\boldsymbol{U}_q, \boldsymbol{v}) = \int_{t_{q-1}}^{t_q} A_0(t - t_{q-1}; \boldsymbol{U}_q, \boldsymbol{v})\, dt - \int_{t_{q-1}}^{t_q} \int_0^{t_{q-1}} B(t, s; \boldsymbol{U}(s), \boldsymbol{v})\, ds dt,$$

where,

$$A_0(t - t_{q-1}; \boldsymbol{U}_q, \boldsymbol{v}) := \int_\Omega D_{ijkl}(t - t_{q-1}) \varepsilon_{kl}(\boldsymbol{U}_q) \varepsilon_{ij}(\boldsymbol{v})\, d\Omega.$$

As above, we assume throughout an isotropic material so that the tensor $\underline{\boldsymbol{D}}$ can be equivalently thought of (in two space dimensions) as the matrix,

$$\boldsymbol{D} = \begin{pmatrix} \lambda(t) + \mu(t) & \mu(t) & 0 \\ \mu(t) & \lambda(t) + \mu(t) & 0 \\ 0 & 0 & \mu(t)/2 \end{pmatrix}.$$

Now, taking $\boldsymbol{v}$ to be each basis function for H_q in turn, and imposing essential boundary data etc. in the usual way we arrive at the equation system,

$$\int_{t_{q-1}}^{t_q} \mathsf{A}(t - t_{q-1})\, dt\ \boldsymbol{U}_q = \int_{t_{q-1}}^{t_q} \mathsf{F}(t)\, dt + \text{“history”}.$$

We turn to the “history” contribution arising from the double time integral below since we want to give an internal variable interpretation along the same lines as in the previous section. Note that this is different to the discrete scheme presented in [54] where we considered a direct discretization of the Volterra integral. In the equations above A and F are essentially the same as the standard stiffness matrix and load vector that arise in standard finite element discretizations of the linear elasticity problem. The only difference is that the Lamé functions and loads are time dependent.

In the practical scheme we integrate these equations to arrive at the problem: for each $q = 1, 2, \ldots$ in turn, find $\boldsymbol{U}_q \in H_q$ such that,

$$\boldsymbol{A}_q \boldsymbol{U}_q = \boldsymbol{F}_q + \text{ “history”}.$$

Now $\boldsymbol{A}$ is precisely the stiffness matrix for linear elasticity but built with the modified Lamé functions,

$$\int_{t_{q-1}}^{t_q} \lambda(t - t_{q-1})\,dt \qquad \text{and} \qquad \int_{t_{q-1}}^{t_q} \mu(t - t_{q-1})\,dt.$$

For the generic Prony series relaxation function,

$$\chi(t) = \chi_0 + \sum_{i=1}^{N_\chi} \chi_i e^{-\alpha_i t},$$

we have,

$$\int_{t_{q-1}}^{t_q} \chi(t - t_{q-1})\,dt = k_q\chi_0 + \sum_{i=1}^{N_\chi} \frac{\chi_i}{\alpha_i}(1 - e^{-\alpha_i k_q}),$$

where $k_q := t_q - t_{q-1}$. These λ and μ terms are then easily evaluated during equation assembly by simple function calls. To evaluate the time integral of the load vector we apply the following Gauss rule to the body forces $\boldsymbol{f}$ and tractions $\boldsymbol{g}$,

$$\int_0^1 m(z)\,dz \approx \varpi_-\Big(m(\xi_+) + m(\xi_-)\Big) + \varpi_+\Big(m(\eta_+) + m(\eta_-)\Big),$$

where,

$$\varpi_\pm := \frac{1}{4} \pm \frac{\sqrt{30}}{72}, \quad \xi_\pm; = \frac{1}{2} \pm \sqrt{\frac{15 + 2\sqrt{30}}{140}} \quad \text{and} \quad \eta_\pm := \frac{1}{2} \pm \sqrt{\frac{15 - 2\sqrt{30}}{140}}.$$

This rule is exact for $m(z) = z^7$, and again we note that the load vector calculation and assembly routines in a standard code need only be modified in a trivial way.

To build the "history" term into the equations recall that we need to include the double time integral,

$$\int_{t_{q-1}}^{t_q} \int_0^{t_{q-1}} B(t, s; \boldsymbol{U}(s), \boldsymbol{v})\,dsdt.$$

Recalling our internal stress variables from the previous section note first that we have,

$$\begin{aligned} \int_0^{t_{q-1}} B(t, s; \boldsymbol{U}(s), \boldsymbol{v})\,ds \;&=\; \sum_{i=1}^{N_\lambda} \int_\Omega e^{-l_i(t - t_{q-1})} \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_{q-1}) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v})\,d\Omega, \\ &+\; \sum_{i=1}^{N_\mu} \int_\Omega e^{-m_i(t - t_{q-1})} \boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{U}; t_{q-1}) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v})\,d\Omega. \end{aligned}$$

From this it follows that we may write the double time integral of the strain history as,

$$\int_{t_{q-1}}^{t_q} \int_0^{t_{q-1}} B(t, s; \boldsymbol{U}(s), \boldsymbol{v})\,dsdt$$

$$= \sum_{i=1}^{N_\lambda} \int_\Omega \left[\int_{t_{q-1}}^{t_q} e^{-l_i(t-t_{q-1})} \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_{q-1})\, dt \right] \cdot \boldsymbol{\varepsilon}(\boldsymbol{v})\, d\Omega$$
$$+ \sum_{i=1}^{N_\mu} \int_\Omega \left[\int_{t_{q-1}}^{t_q} e^{-m_i(t-t_{q-1})} \boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{U}; t_{q-1})\, dt \right] \cdot \boldsymbol{\varepsilon}(\boldsymbol{v})\, d\Omega.$$

Noting expressions of the form,

$$\int_{t_{q-1}}^{t_q} e^{-l_i(t-t_{q-1})}\, dt = l_i^{-1}(1 - e^{-l_i k_q}),$$

the double time integral simplifies to give,

$$\begin{aligned} \int_{t_{q-1}}^{t_q} \int_0^{t_{q-1}} B(t, s; \boldsymbol{U}(s), \boldsymbol{v})\, ds dt &= \sum_{i=1}^{N_\lambda} \int_\Omega l_i^{-1}(1 - e^{-l_i k_q}) \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_{q-1}) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v})\, d\Omega \\ &+ \sum_{i=1}^{N_\mu} \int_\Omega m_i^{-1}(1 - e^{-m_i k_q}) \boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{U}; t_{q-1}) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v})\, d\Omega. \end{aligned}$$

For a piecewise linear (spatial) finite element approximation $\boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_{q-1})$, $\boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{U}; t_{q-1})$ and $\boldsymbol{\varepsilon}(\boldsymbol{v})$ (for $\boldsymbol{v} \in H_q$) are constant on each element and so these terms are trivially integrated to determine the local viscous load vectors. Thus the solution algorithm takes the following outline form.

Outline algorithm

Initialize: $\boldsymbol{\Sigma}_{\lambda_i} = \boldsymbol{\Sigma}_{\mu_i} = 0$ for each i.

Do: for time levels $q = 1, 2, \ldots,$

- Given

  $$\boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_{q-1}) \qquad \text{and} \qquad \boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{U}; t_{q-1})$$

  from the previous time step initialize, for each element Ω_{qj}, the local component viscous force vectors,

  $$\boldsymbol{F}_{\lambda_i}^{qj} = \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_{q-1}) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v}) \qquad \text{and} \qquad \boldsymbol{F}_{\mu_i}^{qj} = \boldsymbol{\Sigma}_{\mu_i}(\boldsymbol{U}; t_{q-1}) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v}).$$

- From these form,

  $$\boldsymbol{F}_{\lambda_i}^{qj} \longleftarrow \boldsymbol{F}_{\lambda_i}^{qj} \int_{\Omega_{qj}} l_i^{-1}(1 - e^{-l_i k_q})\, d\Omega_{qj}$$

  (and similarly for the $\boldsymbol{F}_{\mu_i}^{qj}$).

- Form the global component viscous forces,

  $$\boldsymbol{F}_{\lambda_i}^{q} = \sum_{\Omega_{qj} \subset \overline{\Omega}_q} \boldsymbol{F}_{\lambda_i}^{qj}$$

  (and similarly for the $\boldsymbol{F}_{\mu_i}^{qj}$).

- Assemble the global total viscous forces,

$$\boldsymbol{F}_{\text{visc}} = \sum_{i=1}^{N_\lambda} \boldsymbol{F}^q_{\lambda_i} + \sum_{i=1}^{N_\mu} \boldsymbol{F}^q_{\mu_i}.$$

- Solve the global system,

$$\boldsymbol{A}_q \boldsymbol{U}_q = \boldsymbol{F}_q + \boldsymbol{F}_{\text{visc}}.$$

- Update history data: **UpdateViscousStresses()**.

next q

Stop

In all of the numerical results subsequently presented, the global equations are solved with a diagonally scaled conjugate gradient iteration. Unless explicitly stated otherwise we invariably use a residual tolerance of $\epsilon_{\text{cg}} = 10^{-7}$ as a stopping criterion.

After the solution of the global equations the call to **UpdateViscousStresses()** performs updates of the following form,

$$\boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_q) = e^{-l_i k_q} \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_{q-1}) + \boldsymbol{\lambda}_i l_i \int_{t_{q-1}}^{t_q} e^{-l_i(t_q - s)} \boldsymbol{\varepsilon}(\boldsymbol{U}_q)\, ds.$$

The viscous stresses are now ready for the equation set to be solved at the next time level. Note that,

$$l_i \int_{t_{q-1}}^{t_q} e^{-l_i(t_q-s)}\, ds = 1 - e^{-l_i k_q},$$

and so for piecewise constant temporal approximation the update equation simplifies to,

$$\boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_q) = e^{-l_i k_q} \boldsymbol{\Sigma}_{\lambda_i}(\boldsymbol{U}; t_{q-1}) + \boldsymbol{\lambda}_i (1 - e^{-l_i k_q}) \boldsymbol{\varepsilon}(\boldsymbol{U}_q).$$

We now detail some numerical tests.

7.5 Numerical tests

In this section we demonstrate the convergence of the numerical algorithm by comparing the known solution against artificial exact solutions. Invariably we design the loads and tractions so that the exact displacements have the form,

$$u_1(x, y, t) = T(t) X(x, y) \qquad \text{and} \qquad u_2(x, y, t) = T(t) Y(x, y).$$

These forms make it easier to determine the loads $\boldsymbol{f}$ and $\boldsymbol{g}$.

In the following examples we use the data,

$$E = 2.776\,090\,556\,\text{GPa} \qquad \text{and} \qquad \nu = 0.4, \tag{7.13}$$

which imply for plane stress that,

$$\lambda(0) = 1.321\,947\,884\,\text{GPa} \qquad \text{and} \qquad \mu(0) = 1.982\,921\,826\,\text{GPa}. \tag{7.14}$$

The reason for these choices will become clear later when we look at particular data for Maranyl Nylon 6.6. For the moment we use the arbitrary relaxation functions,

$$\lambda(t) := \lambda(0)\Big(0.3 + 0.2e^{-t} + 0.5e^{-0.1t}\Big), \tag{7.15}$$

$$\mu(t) := \mu(0)\Big(0.2 + 0.1e^{-3t} + 0.2e^{-0.7t} + 0.3e^{-2t} + 0.2e^{-0.2t}\Big). \tag{7.16}$$

Also, since the numerical results presented below use various norms a word on how these norms are calculated is in order.

For the energy and $\boldsymbol{L}_2(\Omega)$ norms we use a seven point Gauss rule (exact for quintics—see e.g. [29]), while for the $\boldsymbol{L}_\infty(\Omega)$ norm, which we don't take too seriously, we simply use the nodal values. We denote the resulting approximate norms with "hats" as in,

$$\|\cdot\|_H \approx \|\cdot\|_{\widehat{H}}, \qquad \|\cdot\|_{\boldsymbol{L}_2(\Omega)} \approx \|\cdot\|_{\widehat{\boldsymbol{L}}_2(\Omega)} \qquad \text{and} \qquad \|\cdot\|_{\boldsymbol{L}_\infty(\Omega)} \approx \|\cdot\|_{\widehat{\boldsymbol{L}}_\infty(\Omega)}.$$

For the norms on time dependence we approximate the local L_∞ norms by sampling at the upper end of the time interval as in,

$$\|w\|_{L_\infty(\mathcal{J}_q;H)} \approx \|w\|_{\widehat{L}_\infty(\mathcal{J}_q;\widehat{H})} := \|w(t_q)\|_{\widehat{H}}.$$

We then approximate the global L_∞ norms by:

$$\|w\|_{L_\infty(0,t;H)} \approx \|w\|_{\widehat{L}_\infty(0,t;\widehat{H})} := \max\Big\{\|w\|_{\widehat{L}_\infty(\mathcal{J}_q;\widehat{H})} : 0 < q \le p\Big\} \qquad \text{for } t \in \mathcal{J}_p.$$

The only exception to these rules is for the results given for the $L_\infty(\mathcal{J};\boldsymbol{L}_\infty(\Omega))$ norm of the error where we use the following approximation,

$$\|w\|_{L_\infty(\mathcal{J};\boldsymbol{L}_\infty(\Omega))} \approx \|w\|_\infty := \max\Big\{\|w(\hat{t}_q)\|_{\widehat{\boldsymbol{L}}_\infty(\Omega)} : q = 1, 2, \ldots\Big\},$$

and where $\hat{t}_q := (t_q + t_{q-1})/2$ denotes the midpoint of the time interval.

Also, the tables of results presented below and in the next chapter are `verbatim` output from the finite element code described later in Chapter 9. The key to the column heading is given in the table below.

<table>
<tr><th>“<code>verbatim</code> output”</th><th>=</th><th>“meaning”</th></tr>
<tr><td><code>k_i</code></td><td>=</td><td>time step k</td></tr>
<tr><td><code>N elem</code></td><td>=</td><td>Number of elements M</td></tr>
<tr><td><code>||u||_E</code></td><td>=</td><td>Maximum energy norm $\|\boldsymbol{u}\|_{L_\infty(\mathcal{J};H)}$ (when known)</td></tr>
<tr><td><code>||uh||_E</code></td><td>=</td><td>Maximum energy norm $\|\boldsymbol{U}\|_{L_\infty(\mathcal{J};H)}$</td></tr>
<tr><td><code>|uh|+|e|</code></td><td>=</td><td>$\|\boldsymbol{U}\|_{L_\infty(\mathcal{J};H)} + \|\boldsymbol{u} - \boldsymbol{U}\|_{L_\infty(\mathcal{J};H)}$ (when known)</td></tr>
<tr><td><code>|uh|+est|e|</code></td><td>=</td><td>$\|\boldsymbol{U}\|_{L_\infty(\mathcal{J};H)} + \mathcal{E}_\Omega(T;\boldsymbol{U})$</td></tr>
<tr><td><code>||u-uh||_E</code></td><td>=</td><td>Maximum energy error $\|\boldsymbol{u} - \boldsymbol{U}\|_{L_\infty(\mathcal{J};H)}$ (when known)</td></tr>
<tr><td><code>est||e||_E</code></td><td>=</td><td>Estimated energy error $\mathcal{E}_\Omega(T;\boldsymbol{U})$</td></tr>
<tr><td><code>inf|u-uh|</code></td><td>=</td><td>Not used — ignore this column</td></tr>
<tr><td><code>||u-uh||</code></td><td>=</td><td>Exact displacement error $\|\boldsymbol{u} - \boldsymbol{U}\|_{L_\infty(\mathcal{J};\boldsymbol{L}_2(\Omega))}$ (when known)</td></tr>
<tr><td><code>||s-sh||</code></td><td>=</td><td>Exact stress error $\|\underline{\boldsymbol{\sigma}} - \underline{\boldsymbol{\sigma}}^h\|_{L_\infty(\mathcal{J};\boldsymbol{L}_2(\Omega))}$ (when known)</td></tr>
</table>

7.6 consistency test

Using a domain of arbitrary shape, shown in Figure 7.1, we solve a problem for which the Galerkin approximation is exact by setting,

$$T(t) := 1, \qquad X(x,y) := -0.03x \qquad \text{and} \qquad Y(x,y) := -0.04y.$$

For boundary conditions we impose a homogeneous Dirichlet condition on u along the left-most vertical edge, and the same on v along the bottom-most horizontal edge. All other edges have tractions imposed.

Since the approximation is continuous piecewise linear in x and y and discontinuous piecewise constant in t it follows that the numerical solution should coincide with the exact solution up to quadrature error.

We solve for a constant time step of $k = 0.5$ to the final time $T = 1$ for mesh widths $h = 0.4, 0.2, 0.1, \ldots$. The first two of these “regular” meshes are shown in Figure 7.1.

The results are shown in the table.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>5.0000e-01</td><td>10</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>1.037705e-06</td><td>2.298352e-07</td><td>1.844e-11</td><td>1.060e-11</td><td>5.488e-02</td></tr>
<tr><td>5.0000e-01</td><td>45</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>1.037425e-06</td><td>1.571306e-07</td><td>1.845e-11</td><td>1.060e-11</td><td>5.488e-02</td></tr>
<tr><td>5.0000e-01</td><td>177</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>1.055355e-06</td><td>1.205493e-07</td><td>1.970e-11</td><td>1.062e-11</td><td>5.532e-02</td></tr>
<tr><td>5.0000e-01</td><td>762</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>1.070804e-06</td><td>8.789404e-08</td><td>1.946e-11</td><td>1.061e-11</td><td>5.555e-02</td></tr>
<tr><td>5.0000e-01</td><td>3109</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>1.094567e-06</td><td>6.704061e-08</td><td>1.940e-11</td><td>1.062e-11</td><td>5.601e-02</td></tr>
</table>

From the table we see that the errors are small and in fact at machine round-off once the magnifying effect of $\lambda(0)$ and $\mu(0)$ are taken into account.

Repeating the calculations but with $k = 0.25$ instead of $k = 0.5$ we get the following tabulated results.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>2.5000e-01</td><td>10</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>5.411507e-09</td><td>1.337637e-09</td><td>9.551e-14</td><td>5.521e-14</td><td>3.173e-04</td></tr>
<tr><td>2.5000e-01</td><td>45</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>1.935447e-07</td><td>4.963813e-08</td><td>1.159e-12</td><td>2.779e-13</td><td>8.384e-03</td></tr>
<tr><td>2.5000e-01</td><td>177</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.049913e-07</td><td>3.918273e-08</td><td>1.466e-12</td><td>3.185e-13</td><td>8.049e-03</td></tr>
<tr><td>2.5000e-01</td><td>762</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>3.662663e-07</td><td>5.598072e-08</td><td>1.642e-12</td><td>5.039e-13</td><td>1.369e-02</td></tr>
<tr><td>2.5000e-01</td><td>3109</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>2.630e+03</td><td>4.556104e-07</td><td>5.152344e-08</td><td>1.816e-12</td><td>5.880e-13</td><td>1.754e-02</td></tr>
</table>

The estimated error quantities shown in the tables is related to the *a posteriori* error estimate and will be explained fully in the next chapter. We include it here in order to be able to refer back later on.

Figure 7.1: Regular meshes for $h = 0.4$ and $h = 0.2$

11 nodes, 10 elements
10 boundary sides during
times $t \in (0.500000, 1.000000)$.

33 nodes, 45 elements
19 boundary sides during
times $t \in (0.500000, 1.000000)$.

7.7 Spatial convergence

We now keep $T(t)$ as above and change $X(x,y)$ and $Y(x,y)$ to,

$$\begin{aligned} X(x,y) &:= -0.03x^7 + 0.025\sin(2\pi x)\sin(\pi y + \pi/2), \\ Y(x,y) &:= -0.04y^9 + 0.035\sin(\pi x + \pi/2)\sin(2\pi y). \end{aligned}$$

All other data is as before (with $k = 0.5$) and we again loop through the same sequence of regular meshes to examine the spatial error convergence.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>5.0000e-01</td><td>10</td><td>8.551e+03</td><td>5.856e+03</td><td>8.561e+03</td><td>9.183e+03</td><td>6.244285e+03</td><td>7.072830e+03</td><td>4.919e-02</td><td>2.393e-02</td><td>2.967e+08</td></tr>
<tr><td>5.0000e-01</td><td>45</td><td>8.561e+03</td><td>8.065e+03</td><td>8.570e+03</td><td>8.689e+03</td><td>2.898224e+03</td><td>3.233216e+03</td><td>1.325e-02</td><td>3.972e-03</td><td>1.314e+08</td></tr>
<tr><td>5.0000e-01</td><td>177</td><td>8.561e+03</td><td>8.440e+03</td><td>8.564e+03</td><td>8.594e+03</td><td>1.448311e+03</td><td>1.616137e+03</td><td>7.472e-03</td><td>2.378e-03</td><td>6.703e+07</td></tr>
<tr><td>5.0000e-01</td><td>762</td><td>8.561e+03</td><td>8.531e+03</td><td>8.561e+03</td><td>8.567e+03</td><td>7.154630e+02</td><td>7.818353e+02</td><td>1.724e-03</td><td>5.423e-04</td><td>3.299e+07</td></tr>
<tr><td>5.0000e-01</td><td>3109</td><td>8.561e+03</td><td>8.554e+03</td><td>8.561e+03</td><td>8.562e+03</td><td>3.422946e+02</td><td>3.783819e+02</td><td>2.925e-04</td><td>5.892e-05</td><td>1.559e+07</td></tr>
<tr><td>5.0000e-01</td><td>12628</td><td>8.561e+03</td><td>8.559e+03</td><td>8.561e+03</td><td>8.561e+03</td><td>1.663714e+02</td><td>1.856497e+02</td><td>5.813e-05</td><td>1.400e-05</td><td>7.579e+06</td></tr>
</table>

It is evident that the energy error is $O(h)$ as expected.

7.8 Temporal convergence

To examine the time discretization error in isolation we now set,

$$T(t) := 1 + t/2 + \sin(2\pi t), \qquad X(x,y) := -0.03x \qquad \text{and} \qquad Y(x,y) := -0.04y.$$

We should expect to see errors converge to zero as $k \to 0$ independently of h. Here we choose $h = 0.4$ corresponding to the mesh shown on the left of Figure 7.1.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>10</td><td>3.945e+03</td><td>3.178e+03</td><td>3.269e+03</td><td>3.193e+03</td><td>7.677814e+02</td><td>3.119966e+02</td><td>1.834e-03</td><td>7.469e-03</td><td>5.821e+07</td></tr>
<tr><td>5.0000e-01</td><td>10</td><td>3.945e+03</td><td>4.641e+03</td><td>4.834e+03</td><td>4.656e+03</td><td>1.986738e+03</td><td>7.177276e+02</td><td>1.482e-02</td><td>1.926e-02</td><td>1.340e+08</td></tr>
<tr><td>2.5000e-01</td><td>10</td><td>5.589e+03</td><td>4.764e+03</td><td>4.987e+03</td><td>4.782e+03</td><td>1.798382e+03</td><td>6.519429e+02</td><td>3.490e-03</td><td>1.743e-02</td><td>1.218e+08</td></tr>
<tr><td>1.2500e-01</td><td>10</td><td>5.589e+03</td><td>5.404e+03</td><td>5.420e+03</td><td>5.405e+03</td><td>1.050321e+03</td><td>3.812408e+02</td><td>1.062e-03</td><td>1.018e-02</td><td>7.120e+07</td></tr>
<tr><td>6.2500e-02</td><td>10</td><td>5.589e+03</td><td>5.562e+03</td><td>5.563e+03</td><td>5.562e+03</td><td>5.476365e+02</td><td>1.988857e+02</td><td>2.730e-04</td><td>5.309e-03</td><td>3.714e+07</td></tr>
<tr><td>3.1250e-02</td><td>10</td><td>5.589e+03</td><td>5.593e+03</td><td>5.593e+03</td><td>5.593e+03</td><td>2.771164e+02</td><td>1.006664e+02</td><td>6.807e-05</td><td>2.687e-03</td><td>1.880e+07</td></tr>
<tr><td>1.5625e-02</td><td>10</td><td>5.597e+03</td><td>5.595e+03</td><td>5.595e+03</td><td>5.595e+03</td><td>1.390760e+02</td><td>5.052768e+01</td><td>1.708e-05</td><td>1.348e-03</td><td>9.435e+06</td></tr>
<tr><td>7.8125e-03</td><td>10</td><td>5.597e+03</td><td>5.597e+03</td><td>5.597e+03</td><td>5.597e+03</td><td>6.962842e+01</td><td>2.529828e+01</td><td>4.275e-06</td><td>6.750e-04</td><td>4.724e+06</td></tr>
<tr><td>3.9062e-03</td><td>10</td><td>5.598e+03</td><td>5.597e+03</td><td>5.597e+03</td><td>5.597e+03</td><td>3.483190e+01</td><td>1.265596e+01</td><td>1.069e-06</td><td>3.377e-04</td><td>2.363e+06</td></tr>
<tr><td>1.9531e-03</td><td>10</td><td>5.598e+03</td><td>5.598e+03</td><td>5.598e+03</td><td>5.598e+03</td><td>1.741975e+01</td><td>6.329462e+00</td><td>2.672e-07</td><td>1.689e-04</td><td>1.182e+06</td></tr>
</table>

This time we see that the energy error is $O(k)$, again as expected.

7.9 The general case: $\boldsymbol{D} \neq \boldsymbol{D}^T$

For more general anisotropic problems we cannot assume that $\boldsymbol{D}(t) = \boldsymbol{D}^T(t)$ unless $t = 0$ or $t = \infty$. Hence $\mathbf{A} \neq \mathbf{A}^T$ and we would have the additional complexity of solving a nonsymmetric system at each time level. In this case we could use the evolution equations for the internal variables and couple these to an elasticity solver (with additional viscous loads) in an iterative solution algorithm. This is essentially the solution technique described by Johnson and Tessler in [24] and is suited also to constitutively nonlinear problems.

Chapter 8

Adaptive error control

8.1 Introduction

In this chapter we summarize the *a posteriori* error estimates developed recently by Shaw and Whiteman, in [55, 56], for the Volterra formulation of the quasistatic problem defined by (**4**.3) with (**2**.2). These results follow on from the exploratory work in [44, 47, 57, 53]. We then implement these error bounds in the context of an adaptive space-time finite element solver for the linear problem with viscoelasticity described by relaxation functions of Prony type. We note here that while the estimates themselves do not rely on this form of relaxation function, it is the only convenient choice for numerical computation since it leads to an economical history storage strategy (see for example [45]). Also, as already demonstrated, the Prony series leads to a natural connection with internal variable methods.

The plan for the chapter is as follows. We first summarize the basic error bound as given in [55], and to do this we refer back to the weak form of the problem as well as its space-time finite element discretization given in the previous chapter. Since in many ways the problem is close to linear elasticity we first present the spatial error control strategy in this context, and also give some numerical results on error control via adaptive mesh refinement.

In [55] we give two forms for the term in the *a posteriori* error estimate that reflects the time discretization error. In the first the estimate is unstable (as $h \to 0$) and so is useless for error control. The second form on the other hand is robust but requires that the residual be measured in a discrete negative norm. This would require a stiffness matrix inversion and is therefore likely to be prohibitively expensive.

These difficulties are due to a lack of **strong temporal stability** in the underlying differential equations which, in this context, means roughly that there are no time derivatives present on the left of (**4**.3) (as would be the case with, say, an ODE or parabolic equation). To overcome this fundamental limitation (which has hampered other work for scalar pure-time Volterra equations in, for example, [28] and [2]) we are also working on deriving *a posteriori* error estimates in a weak norm in [56]. This is based on the prototype work in [57], and we expect this error estimate to allow for robust temporal error control through adaptive time stepping as well as the adaptive meshing we describe below.

8.2 *A posteriori* error estimate

In [55] we give the following basic *a posteriori* Galerkin-energy error estimate: for each discrete time $t_1, t_2, \ldots, t_p, \ldots$,

$$\|\boldsymbol{u} - \boldsymbol{U}\|_{L_\infty(0,t_p;H)} \le S(t_p)\Big(\mathcal{E}_\Omega(t_p;\boldsymbol{U}) + \mathcal{E}_\mathcal{J}(t_p;\boldsymbol{U}) + \mathcal{E}_V(t_p;\boldsymbol{U})\Big), \tag{8.1}$$

where $\mathcal{E}_\Omega$, $\mathcal{E}_\mathcal{J}$ and $\mathcal{E}_V$ are residuals which are computable in terms of the data and the finite element solution $\boldsymbol{U}$, and $S(t)$ is the stability factor introduced before in (**7**.6).

In this section and the next we will be concerned only with $\mathcal{E}_\Omega$. This term contains the spatial discretization error (in the case where $\mathcal{H}_{ij} = \emptyset$ for all $j \le i$) and can be used to guide adaptive space mesh refinement. It is essentially identical to the residual derived for linear elasticity by Johnson and Hansbo in [27], and this is useful because in the next section we may illustrate its interpretation and use in this less crowded context. The extension to viscoelasticity to come later will then be straightforward.

The residual $\mathcal{E}_\mathcal{J}$ is the one described earlier as being unstable (useless) or—when written in a different form—prohibitively expensive to implement. As described above, we eventually hope to provide an alternative error estimate in which $\mathcal{E}_\Omega$ and $\mathcal{E}_V$ are essentially the same, while $\mathcal{E}_\mathcal{J}$ is stabilized at the expense of estimating the error in a weaker norm.

It is the term $\mathcal{E}_V$ that causes the greatest difficulty in this estimate. The spatial residuals in $\mathcal{E}_\Omega$ are constructed by integrating the discrete solution over each element to arrive at a **distributional** divergence of the discrete stress (compare (**4**.3)). This divergence comprises two parts: the smooth function inside the element (which is zero in our case of piecewise linear approximation), and the stress jumps across inter-element boundaries. The difficulty arises because the stress is history dependent. This means that we have to integrate by parts over not just the elements in the current mesh, but also over all elements in all previous meshes. The internal edges that appeared in previous meshes but are no loner present in the current mesh (e.g. due to derefinement) are therefore "left behind" when forming the standard residual $\boldsymbol{f} + \nabla \cdot \underline{\boldsymbol{\sigma}}^h$ (which constitutes $\mathcal{E}_\Omega$), and so we consign the stress jumps across these edges to the term $\mathcal{E}_V$. In the particular case where only nested refinements are permitted (so that $\mathcal{H}_{ij} = \emptyset$ for all $j \le i$) then no edges are left behind in this way and we have $\mathcal{E}_V \equiv 0$. This is the case in all of our examples below.

To deal with mesh derefinement would appear to require fairly complex data structures in the computer code in order to track all these resulting previous edges. Also, it is not likely that $\mathcal{E}_V$ will act in any way other than to degrade the quality of the estimate since it contains historical contributions to the current stress. These can then only act to reinforce one another in the estimate when in fact the residual could be much smaller due to cancellation. This "loss of cancellation" problem has been noted by others in the context of stress-jump residual-type estimators and the memory in the Volterra integral acts here only to exacerbate the problem. Our feeling at the moment is that a representation of the algorithm in terms of internal variables could go some way toward removing the $\mathcal{E}_V$ residual since then all hereditary information is automatically represented on the current mesh. The price of this is that the error estimates will then be restricted to viscoelasticity problems for which Prony series relaxation functions are appropriate. This does not seem to be an unreasonable restriction.

We now look in more detail at the residual term $\mathcal{E}_\Omega$. This is defined at each discrete time level $t_1, t_2, \ldots, t_p, \ldots$ as:

$$\mathcal{E}_\Omega(t_p; \boldsymbol{U}) := \max_{1\le q\le p} \left\{ \Pi_{\Omega_q} \|h_q \boldsymbol{f}\|_{L_\infty(\mathcal{J}_q;\boldsymbol{L}_2(\Omega))} + \Pi_\ell \|h_q \mathcal{G}\|_{L_\infty(\mathcal{J}_q;L_2(\Omega))} \right\}, \tag{8.2}$$

where Π_{Ω_q} and Π_ℓ are constants appearing in certain interpolation-error bounds, and $h_q = h_q(\boldsymbol{x})$ is the piecewise constant mesh function for the mesh during times in $\mathcal{J}_q$.

The first term in the estimate is straightforward to interpret since it involves only the $\boldsymbol{L}_2(\Omega)$ norm of the body forces weighted with the mesh function. To define the second term we need to establish various other notation.

We use $\widehat{\boldsymbol{n}}^{(im)} = (\widehat{n}_l^{(im)})_{l=1}^n$ to denote the unit outward directed normal vector to the the boundary $\partial\Omega_{im}$ of Ω_{im}, for $m \in \mathbb{N}(1, M_i)$, and for an edge/face $\ell \in \mathcal{L}_i$ we use the notation $[\![\gamma_k]\!]_\ell$ to denote the jump in value across the edge/face ℓ of the components of any $\boldsymbol{\gamma} = (\gamma_k)_{k=1}^n$. That is, for $\boldsymbol{x} \in \ell \in \mathcal{L}_i$ and each $i \in \mathbb{N}(1, N)$,

$$[\![\gamma_k(\boldsymbol{x})]\!]_\ell := \pm \lim_{\epsilon\to 0} \left(\gamma_k(\boldsymbol{x} - \epsilon\widehat{\boldsymbol{n}}^{(\ell)}) - \gamma_k(\boldsymbol{x} + \epsilon\widehat{\boldsymbol{n}}^{(\ell)}) \right), \tag{8.3}$$

where: $\widehat{\boldsymbol{n}}^{(\ell)}$ is a unit normal vector to the edge/face ℓ; and, to avoid elaborate notation we use "$\pm$" to acknowledge here that the sign of this jump quantity is of no interest at all.

We denote surface (edge/face) integrals on the element boundaries by

$$(\boldsymbol{w}, \boldsymbol{v})_\ell := \int_\ell \boldsymbol{w}\cdot\boldsymbol{v}\, d\ell \qquad \text{with} \qquad \|\cdot\|_\ell := \sqrt{(\,\cdot\,,\,\cdot\,)_\ell},$$

and we define the discrete traction,

$$\widetilde{\boldsymbol{g}}(t; \boldsymbol{U}(t)) := (\widetilde{g}_k(t; \boldsymbol{U}(t)))_{k=1}^n,$$

where for a unit vector $\widehat{\boldsymbol{n}}$,

$$\widetilde{g}_k(t; \boldsymbol{U}(t)) := \left(D_{klij}(0)\varepsilon_{ij}(\boldsymbol{U}(t)) - \int_0^t \frac{\partial D_{klij}(t-s)}{\partial s} \varepsilon_{ij}(\boldsymbol{U}(s))\, ds \right) \widehat{n}_l \tag{8.4}$$

(compare the natural boundary condition in (4.3)). In general these discrete tractions will not be uniquely defined on any edge/face $\ell \in \mathcal{L}_i$, but the jumps $[\![\widetilde{\boldsymbol{g}}]\!]_\ell$ will.

Now, for each time level we define $\boldsymbol{r}|_{\mathcal{J}_q} = (r_k)_{k=1}^n$ by,

$$r_k(t; \boldsymbol{U}(t)) := \begin{cases} \dfrac{1}{2}\left|[\![\widetilde{g}_k(t; \boldsymbol{U}(t))]\!]_\ell\right|, & \text{for } \ell \in \mathcal{L}_q, \\ \left|g_k(t) - \widetilde{g}_k(t; \boldsymbol{U}(t))\right|, & \text{on } \Gamma_N, \\ 0, & \text{on } \Gamma_D, \end{cases}$$

and then with all of these definitions we define $\mathcal{G} \in L_2(\Omega)$ by,

$$\mathcal{G}|_{\Omega_{qj}} := \frac{\|\boldsymbol{r}(t; \boldsymbol{U}(t))\|_{\boldsymbol{L}_2(\partial\Omega_{qj})}}{\sqrt{h_{qj}\operatorname{meas}(\Omega_{qj})}}$$

for all $j \in \mathbb{N}(1, M_q)$. Note that,

$$\|h_q \mathcal{G}\|_{L_2(\Omega)} \equiv \left(\sum_{\Omega_{qj} \subset \overline{\Omega}_q} \left\| h_{qj}^{\frac{1}{2}} \boldsymbol{r}(t; \boldsymbol{U}(t)) \right\|^2_{\boldsymbol{L}_2(\partial\Omega_{qj})} \right)^{\frac{1}{2}}. \tag{8.5}$$

We now use $\mathcal{E}_\Omega$ to derive element error indicators and a mesh refinement criterion in the context of linear elasticity.

8.3 Adaptive meshing for linear elasticity

In the linear elasticity context there is of course no time dependence and so we get the simpler result,

$$\|\boldsymbol{u} - \boldsymbol{U}\|_H \le \mathcal{E}_\Omega(\boldsymbol{U}) := \Pi_\Omega \|h\boldsymbol{f}\|_{\boldsymbol{L}_2(\Omega)} + \Pi_\ell \|h\mathcal{G}\|_{L_2(\Omega)}.$$

Note that the L_∞ norms and the subscripts p and q marking the time levels are not needed here. Also we have temporarily set $S(t) \equiv 1$. It will be straightforward to extend the following results to the viscoelasticity problem later.

The goal is to design a mesh for which $\|\boldsymbol{u} - \boldsymbol{U}\|_H \le \mathsf{TOL}$ where $\mathsf{TOL} > 0$ is a user-defined tolerance level. Further, this mesh should be optimal in the sense that the error control is achieved with as few degrees of freedom as possible. Although impossibly difficult to obtain in an exact sense (see for example the discussion in [17]), such a mesh can be approximated with sensible and restrained use of an *a posteriori* error estimate. We aim for a mesh size modification strategy for each element Ω_j in the mesh.

The error control is clearly achieved if we ensure that $\mathcal{E}_\Omega(\boldsymbol{U}) \le \mathsf{TOL}$, and to do this we use the technique of **equidistribution** whereby each element is allowed to contribute equally to the global error (regardless of element size). In practice this means that we assign a local tolerance $\mathsf{tol} > 0$ to each element and attempt to control the local element errors to within tol.

For each element Ω_j in the mesh we seek a local meshwidth h_j and solution $\boldsymbol{U}$ for which,

$$\eta_j^2 := \Pi_\Omega^2 h_j^2 \|\boldsymbol{f}\|^2_{\boldsymbol{L}_2(\Omega_j)} + \Pi_\ell^2 h_j^2 \|\mathcal{G}\|^2_{L_2(\Omega_j)} = \mathsf{tol}.$$

Rearranging this we then arrive at an adaptive mesh size selector via,

$$h_j^{\text{new}} := \sqrt{\frac{\mathsf{tol}}{\Pi_\Omega^2 \|\boldsymbol{f}\|^2_{\boldsymbol{L}_2(\Omega_j)} + \Pi_\ell^2 \|\mathcal{G}\|^2_{L_2(\Omega_j)}}} \tag{8.6}$$

for each element Ω_j. Then, globally, where M is the number of elements in the mesh we get,

$$M \times \mathsf{tol} \;=\; \sum_{j=1}^{M} \mathsf{tol} \;=\; \sum_{j=1}^{M} \eta_j^2,$$

$$= \quad \Pi_\Omega^2 \|h\boldsymbol{f}\|^2_{\boldsymbol{L}_2(\Omega)} + \Pi_\ell^2 \|h\mathcal{G}\|^2_{L_2(\Omega)} \quad \geq \quad \frac{1}{2}\Big(\mathcal{E}_\Omega(\boldsymbol{U})\Big)^2,$$

where we used the inequality $a^2 + b^2 \geq (|a| + |b|)^2/2$. Hence,

$$\mathcal{E}_\Omega(\boldsymbol{U}) \leq \mathsf{TOL} \qquad \text{is guaranteed if} \qquad \mathsf{tol} := \frac{\mathsf{TOL}^2}{2M}.$$

This is then our formula for the local error tolerances.

The basic form of the adaptive algorithm we implement is then as follows.

1. *Set $i = 0$ and generate an initial mesh $\mathcal{M}^i$.*

2. *Solve for the displacements $\boldsymbol{U}^i$ on $\mathcal{M}^i$.*

3. *Compute η_j^2 on each element Ω_j of $\mathcal{M}^i$, then:*
  if $\sum_j \eta_j^2 \leq \mathsf{TOL}^2/2$ **STOP**
  else *determine a new mesh $\mathcal{M}^{i+1}$ from* (**8**.6)

4. *Set $i = i + 1$ and repeat from Step 2.*

At this point we should sound a note of caution. This algorithm should not be interpreted glibly in that the **else** clause in Step 3 should not be taken literally. If every element is refined according to (**8**.6) then, unless the initial mesh $\mathcal{M}^0$ is already nearly optimal, one is likely to get severe over-refinement in the adapted meshes due to pollution error. Instead, rather than attempt to derive the required mesh in a "one shot" manner by subdividing each elements as indicated by (**8**.6), one should only halve the size of the elements marked for refinement and then loop back and recompute.

Before moving on to viscoelasticity we now give a couple of examples to illustrate the adaptive solution of linear elasticity problems. We again assume an isotropic material in which case Hooke's law becomes,

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{pmatrix} = \begin{pmatrix} \lambda+\mu & \lambda & 0 \\ \lambda & \lambda+\mu & 0 \\ 0 & 0 & \mu/2 \end{pmatrix} \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{pmatrix},$$

where λ and μ are the **Lamé coefficients** related to the Young's modulus E and Poisson's ratio ν through,

$$\lambda := \frac{\nu E}{(1+\nu)(1-2\nu)} \qquad \text{and} \qquad \mu := \frac{E}{1+\nu}.$$

Note that also $\mu = 2G$ where G is the shear modulus of the material. In two-dimensional problems in which the stress in the third direction is negligible engineers frequently employ the **plane stress** approximation where,

$$\lambda := \frac{\nu E}{1-\nu^2}.$$

The first definition of λ in a two-dimensional formulation implies **plane strain** (in which case the strain in the third direction is assumed negligible). Typically one may use plane stress for very thin components, and plane strain for very thick ones.

In the following examples we use the same data as before: (**7**.13), (**7**.14), (**7**.15) and (**7**.16).

8.3.1 The interpolation-error constants Π_Ω and Π_ℓ

To complete the description of the *a posteriori* error estimate for linear elasticity we need to specify the interpolation-error constants Π_Ω and Π_ℓ. For this we adapt the approximate values calculated by Ludwig in [33, Tables 6.2 and 6.4]. Here the interpolation constants are given for a variety of Poisson ratios, for plane strain with $E = 1$, and assuming a mesh of right-angled triangles. The values are reproduced here in Table 8.1.

Table 8.1: Interpolation-error constants

ν	Π_Ω	Π_ℓ	Π_Ω/Π_ℓ
0.1	1.143177	2.683068	0.426071
0.2	1.172415	2.715093	0.431814
0.3	1.197708	2.734093	0.438064
0.4	1.219404	2.741196	0.444844
$0.5-\epsilon$	1.237806	(2.947157)	0.42*

The value for Π_ℓ for $\nu \to 0.5$ is not given by Ludwig. To calculate it here we assume a scaling of $\Pi_\Omega/\Pi_\ell = 0.42$ and then work from the tabulated value of Π_Ω. These values can be incorporated into a computer code as a look-up table and then values for any value of ν obtained by linear interpolation. The constants scale with E in the following way:

$$\frac{\Pi_\Omega}{\widehat{\Pi}_\Omega} = \frac{\Pi_\ell}{\widehat{\Pi}_\ell} = \sqrt{\frac{\widehat{E}}{E}}.$$

(Note also that Ludwig uses 2μ in Hooke's law where we use μ, but since this does not affect E and ν this difference is immaterial).

We need now to obtain the corresponding interpolation constants for plane stress, note first that:

$$\left.\begin{aligned} \lambda &= \frac{\nu E}{(1-2\nu)(1+\nu)} \\ \mu &= \frac{E}{1+\nu} \end{aligned}\right\} \implies \left\{\begin{aligned} E &= \frac{(\mu+3\lambda)\mu}{\mu+2\lambda} \\ \nu &= \frac{\lambda}{\mu+2\lambda} \end{aligned}\right. \quad \text{for plane strain,}$$

$$\left.\begin{aligned} \lambda &= \frac{\nu E}{(1-\nu)^2} \\ \mu &= \frac{E}{1+\nu} \end{aligned}\right\} \implies \left\{\begin{aligned} E &= \frac{(\mu+2\lambda)\mu}{\mu+\lambda} \\ \nu &= \frac{\lambda}{\mu+\lambda} \end{aligned}\right. \quad \text{for plane stress.}$$

So, given E and ν for plane stress we can first calculate λ and μ, and then work backwards with these values to find the corresponding E and ν for plane strain—$\widehat{E}$ and $\widehat{\nu}$ say. From these we can then determine the interpolation constants from the table and the scaling given above.

In our numerical results we take $\nu = 0.4$ and $E = 2.776\,090\,556\,\text{GPa}$ which give the plane stress values,

$$\lambda = 0.476\,190\,476E \qquad \text{and} \qquad \mu = 0.714\,285\,714E.$$

These give the corresponding $\widehat{E}$ and $\widehat{\nu}$ plane strain values,

$$\widehat{E} = 0.918\,367\,346E \qquad \text{and} \qquad \widehat{\nu} = 0.2857\ldots.$$

Assuming $\widehat{E} = 1$ the table gives for this Poisson ratio,

$$\widehat{\Pi}_\Omega \approx 1.2 \qquad \text{and} \qquad \widehat{\Pi}_\ell \approx 2.8,$$

and then using the scaling for $\widehat{E}$ we finally get the values,

$$\Pi_\Omega = \frac{1.252198}{\sqrt{E}} \qquad \text{and} \qquad \Pi_\ell = \frac{2.9218}{\sqrt{E}}.$$

However, there is some doubt as to whether this exercise is worthwhile. These constants represent the worse possible case in interpolation error and often end up making the *a posteriori* error estimate significantly over-estimate the finite element error. To address this difficulty we have **calibrated** these constants against exact solutions in order to render the *a posteriori* estimates more realistic. The end result is that we divide the values given above by a factor of ten and twenty respectively. These values are suggested by the calculations for the exact solution given in an earlier table, but we will see below that it is desirable to determine a more systematic calibration technique.

8.3.2 Example: exact solution

For the moment we switch off all time dependencies and viscoelasticity effects and consider the adaptive solution of a linear elasticity problem. We use the elastic coefficients $\lambda := \lambda(0)$ and $\mu := \mu(0)$ from the previous chapter, and impose loads and tractions such that the exact solutions are,

$$T(t) := 10^{-2}, \qquad X(x,y) := -0.03x^{20} \qquad \text{and} \qquad Y(x,y) := -0.04y^{20}.$$

We need these because adaptivity doesn't really show up as useful for smooth solutions—so we simulate a singularity.

The notation here is exactly as in the previous chapter and we impose boundary conditions in the same way. For these numerical tests we simply solve to time $T = 1$ using a single time step $k = 1$. The first set of results show reference solutions calculated for uniform meshes with $h = 0.4, 0.2, \ldots$. Again, these are exactly as in the previous chapter.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>10</td><td>8.023e+01</td><td>5.224e+01</td><td>8.198e+01</td><td>1.380e+02</td><td>6.317922e+01</td><td>1.277228e+02</td><td>5.289e-04</td><td>2.822e-04</td><td>3.896e+06</td></tr>
<tr><td>1.0000e+00</td><td>45</td><td>8.370e+01</td><td>6.588e+01</td><td>8.377e+01</td><td>9.146e+01</td><td>5.173867e+01</td><td>6.344186e+01</td><td>1.402e-04</td><td>4.603e-05</td><td>3.109e+06</td></tr>
<tr><td>1.0000e+00</td><td>177</td><td>8.397e+01</td><td>7.671e+01</td><td>8.397e+01</td><td>8.380e+01</td><td>3.416709e+01</td><td>3.374743e+01</td><td>1.086e-04</td><td>2.062e-05</td><td>2.098e+06</td></tr>
<tr><td>1.0000e+00</td><td>762</td><td>8.398e+01</td><td>8.167e+01</td><td>8.398e+01</td><td>8.342e+01</td><td>1.954781e+01</td><td>1.700767e+01</td><td>2.818e-05</td><td>6.674e-06</td><td>1.186e+06</td></tr>
<tr><td>1.0000e+00</td><td>3109</td><td>8.398e+01</td><td>8.347e+01</td><td>8.398e+01</td><td>8.386e+01</td><td>9.258688e+00</td><td>8.078055e+00</td><td>1.351e-05</td><td>2.936e-06</td><td>5.496e+05</td></tr>
<tr><td>1.0000e+00</td><td>12628</td><td>8.398e+01</td><td>8.387e+01</td><td>8.398e+01</td><td>8.396e+01</td><td>4.291128e+00</td><td>3.861923e+00</td><td>2.782e-06</td><td>7.406e-07</td><td>2.556e+05</td></tr>
</table>

Looking at the estimated errors we now tabulate adapted solutions for $\mathsf{TOL} = 15$, 12.5, 10, 7.5, 5, with $h = 0.1$ for the initial mesh.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>488</td><td>8.398e+01</td><td>8.300e+01</td><td>8.398e+01</td><td>8.376e+01</td><td>1.275528e+01</td><td>1.124042e+01</td><td>1.806e-05</td><td>3.435e-06</td><td>7.766e+05</td></tr>
<tr><td>1.0000e+00</td><td>639</td><td>8.398e+01</td><td>8.325e+01</td><td>8.398e+01</td><td>8.382e+01</td><td>1.103850e+01</td><td>9.734371e+00</td><td>1.838e-05</td><td>2.866e-06</td><td>6.731e+05</td></tr>
<tr><td>1.0000e+00</td><td>711</td><td>8.398e+01</td><td>8.333e+01</td><td>8.398e+01</td><td>8.386e+01</td><td>1.042688e+01</td><td>9.379712e+00</td><td>1.772e-05</td><td>2.634e-06</td><td>6.384e+05</td></tr>
<tr><td>1.0000e+00</td><td>1085</td><td>8.398e+01</td><td>8.360e+01</td><td>8.398e+01</td><td>8.389e+01</td><td>7.918571e+00</td><td>6.890573e+00</td><td>9.779e-06</td><td>1.720e-06</td><td>4.819e+05</td></tr>
<tr><td>1.0000e+00</td><td>1858</td><td>8.398e+01</td><td>8.377e+01</td><td>8.398e+01</td><td>8.393e+01</td><td>5.874564e+00</td><td>5.137793e+00</td><td>6.566e-06</td><td>1.122e-06</td><td>3.582e+05</td></tr>
</table>

Repeating these calculations but starting with an initial mesh of $h = 0.2$ gives the following results.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>512</td><td>8.398e+01</td><td>8.302e+01</td><td>8.398e+01</td><td>8.375e+01</td><td>1.263565e+01</td><td>1.099203e+01</td><td>2.475e-05</td><td>3.259e-06</td><td>7.600e+05</td></tr>
<tr><td>1.0000e+00</td><td>561</td><td>8.398e+01</td><td>8.311e+01</td><td>8.398e+01</td><td>8.375e+01</td><td>1.206217e+01</td><td>1.036425e+01</td><td>2.377e-05</td><td>2.912e-06</td><td>7.248e+05</td></tr>
<tr><td>1.0000e+00</td><td>749</td><td>8.398e+01</td><td>8.333e+01</td><td>8.398e+01</td><td>8.384e+01</td><td>1.045034e+01</td><td>9.294968e+00</td><td>1.896e-05</td><td>1.916e-06</td><td>6.337e+05</td></tr>
<tr><td>1.0000e+00</td><td>1289</td><td>8.398e+01</td><td>8.361e+01</td><td>8.398e+01</td><td>8.388e+01</td><td>7.842236e+00</td><td>6.718701e+00</td><td>1.051e-05</td><td>1.479e-06</td><td>4.728e+05</td></tr>
<tr><td>1.0000e+00</td><td>2403</td><td>8.398e+01</td><td>8.379e+01</td><td>8.398e+01</td><td>8.393e+01</td><td>5.561009e+00</td><td>4.808581e+00</td><td>6.485e-06</td><td>7.713e-07</td><td>3.352e+05</td></tr>
</table>

Although the effect on the number of elements is not too serious, this seems to verify the popular wisdom that the effectiveness of the adaptive meshing procedure is influenced by the initial mesh. We will show some pictures of adapted meshes in the next section where we move on and perform the analogous numerical experiments for linear viscoelasticity.

8.4 Adaptive meshing for linear viscoelasticity

With the strategy for adaptive space meshing established in the context of the linear elasticity problem, it is now straightforward to extend it to the time dependent viscoelasticity problem. All that we do is apply the mesh refinement criteria at each time level in turn, as if we were dealing with a sequence of linear eleasticity problems. The only major difference is the inclusion of the time dependence in the *a posteriori* error estimate. Recalling this estimate from (**8**.1) we now assume that we are given a tolerance TOL_Ω with which to control the mesh such that,

$$S(t_p)\mathcal{E}_\Omega(t_p;\boldsymbol{U}) \leq \mathsf{TOL}_\Omega, \qquad \text{for each time level: } p = 1, 2, \ldots.$$

We describe the stability factor in more detail below and for now note only that it is a non-decreasing function. So, replacing $S(t_p)$ with $S(T)$ and recalling the definition in (**8**.2) we see that this error control will be guaranteed if we ensure that,

$$\Pi_{\Omega_q}\|h_q\boldsymbol{f}\|_{L_\infty(\mathcal{J}_q;\boldsymbol{L}_2(\Omega))} + \Pi_\ell\|h_q\mathcal{G}\|_{L_\infty(\mathcal{J}_q;L_2(\Omega))} \leq \frac{\mathsf{TOL}_\Omega}{S(T)}$$

at each time level t_q. The strategy is now exactly as above for the linear elasticity problem. Assuming a local tolerance tol we derive the adaptive mesh size selector,

$$h_{qj}^{\text{new}} := \sqrt{\frac{\mathsf{tol}}{\Pi_\Omega^2 \|\boldsymbol{f}\|^2_{L_\infty(\mathcal{J}_q;\boldsymbol{L}_2(\Omega_j))} + \Pi_\ell^2 \|\mathcal{G}\|^2_{L_\infty(\mathcal{J}_q;L_2(\Omega_j))}}}, \tag{8.7}$$

and then $S(t_p)\mathcal{E}_\Omega(t_p;\boldsymbol{U}) \leq \mathsf{TOL}_\Omega$ is guaranteed if we choose

$$\mathsf{tol} = \frac{\mathsf{TOL}_\Omega^2}{2MS^2(T)}.$$

Below, in the implementation, we replace the indicated norms with practical approximations as discussed earlier. Note that in the above we have set $\Pi_{\Omega_q} = \Pi_\Omega$ for each q: i.e. we take the interpolation-error constants as time independent (even though the mesh is not), and then use the values for Π_Ω and Π_ℓ given previously.

We now outline the form of the stability factor and then follow with the extension of the numerical results for linear elasticity to this time dependent problem. For the moment we assume that there are no time discretization errors since we want to consider the more difficult problem of temporal error control in a separate section below.

8.4.1 The stability factor $S(T)$

For an isotropic viscoelastic material in two dimensions the stability factor $S(t)$ has been derived rather precisely by Shaw and Whiteman in [57]. We give here only the main result and refer to the reference for details.

For a viscoelastic solid we have,

$$S(t) := \left(1 - \int_0^t \phi(s)\,ds\right)^{-1} \qquad \text{where} \qquad \phi(t) := \max\{\omega_1(t), \omega_2(t)\},$$

and,

$$\omega_1(t) := -\frac{\lambda'(t) + \mu'(t)}{\lambda(0) + \mu(0)}, \qquad \omega_2(t) := -\frac{\mu'(t)}{\mu(0)}.$$

For example, using our test data from (**7**.15) and (**7**.16) we have,

$$\phi(t) := \begin{cases} \omega_2(t), & \text{for } 0 \leq t \leq t^* := 2.28476\ldots, \\ \omega_1(t), & \text{for } t \geq t^*. \end{cases}$$

these give,

$$S(t) := \begin{cases} \dfrac{\mu(0)}{\mu(t)}, & \text{for } 0 \leq t \leq t^*, \\ \left(\dfrac{\mu(t^*)}{\mu(0)} + \dfrac{\lambda(t) - \lambda(t^*) + \mu(t) - \mu(t^*)}{\lambda(0) + \mu(0)}\right)^{-1}, & \text{for } t \geq t^* \end{cases}$$

Note also (thinking ahead to the “Maranyl” Nylon 6.6 data given later) that for a synchronous (solid) viscoelastic material there exists a generic relaxation function $\varphi(t)$, normalized to $\varphi(0) = 1$, and such that,

$$\frac{\lambda(t)}{\lambda(0)} = \frac{\mu(t)}{\mu(0)} = \varphi(t).$$

In this case we have the simpler result $S(t) = 1/\varphi(t)$.

8.4.2 Example: exact solution

We continue with the “singular” linear elasticity example used above, but now use the assynchronous relaxation data for $\lambda(t)$ and $\mu(t)$ as described by equations (**7**.13)—(**7**.16). In particular we still take $T(t) = 10^{-2}$ and solve up to time $T = 1$, this means that the solution is time independent and the only time discretization error is due to numerical integration of the load terms $\boldsymbol{f}$ and $\boldsymbol{g}$. To keep this quadrature error “under control” we first do some tests to determine an appropriate time step.

To follow the pattern of the linear elasticity calculations we first show results for uniform meshes $h = 0.4, 0.2, \ldots$ in the tables below for $k = 1.0$, 0.5 and 0.25. The first table is for $k = 1.0$.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>10</td><td>8.023e+01</td><td>5.237e+01</td><td>8.209e+01</td><td>1.686e+02</td><td>6.321785e+01</td><td>1.602739e+02</td><td>5.074e-04</td><td>2.774e-04</td><td>2.584e+06</td></tr>
<tr><td>1.0000e+00</td><td>45</td><td>8.370e+01</td><td>6.616e+01</td><td>8.400e+01</td><td>1.032e+02</td><td>5.175530e+01</td><td>7.917671e+01</td><td>1.777e-04</td><td>5.827e-05</td><td>2.041e+06</td></tr>
<tr><td>1.0000e+00</td><td>177</td><td>8.397e+01</td><td>7.677e+01</td><td>8.404e+01</td><td>8.760e+01</td><td>3.417289e+01</td><td>4.217796e+01</td><td>1.328e-04</td><td>3.048e-05</td><td>1.391e+06</td></tr>
<tr><td>1.0000e+00</td><td>762</td><td>8.398e+01</td><td>8.171e+01</td><td>8.401e+01</td><td>8.442e+01</td><td>1.955150e+01</td><td>2.121986e+01</td><td>3.469e-05</td><td>9.753e-06</td><td>7.834e+05</td></tr>
<tr><td>1.0000e+00</td><td>3109</td><td>8.398e+01</td><td>8.348e+01</td><td>8.399e+01</td><td>8.408e+01</td><td>9.260549e+00</td><td>1.005926e+01</td><td>1.368e-05</td><td>2.587e-06</td><td>3.615e+05</td></tr>
</table>

The analogous results for $k = 0.5$ now follow.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>5.0000e-01</td><td>10</td><td>8.023e+01</td><td>5.242e+01</td><td>8.214e+01</td><td>1.686e+02</td><td>6.323676e+01</td><td>1.602618e+02</td><td>5.159e-04</td><td>2.791e-04</td><td>3.008e+06</td></tr>
<tr><td>5.0000e-01</td><td>45</td><td>8.370e+01</td><td>6.625e+01</td><td>8.408e+01</td><td>1.032e+02</td><td>5.176755e+01</td><td>7.914604e+01</td><td>1.893e-04</td><td>6.411e-05</td><td>2.384e+06</td></tr>
<tr><td>5.0000e-01</td><td>177</td><td>8.397e+01</td><td>7.680e+01</td><td>8.406e+01</td><td>8.761e+01</td><td>3.417703e+01</td><td>4.216685e+01</td><td>1.403e-04</td><td>3.398e-05</td><td>1.619e+06</td></tr>
<tr><td>5.0000e-01</td><td>762</td><td>8.398e+01</td><td>8.172e+01</td><td>8.403e+01</td><td>8.443e+01</td><td>1.955415e+01</td><td>2.121487e+01</td><td>3.670e-05</td><td>1.085e-05</td><td>9.129e+05</td></tr>
<tr><td>5.0000e-01</td><td>3109</td><td>8.398e+01</td><td>8.348e+01</td><td>8.400e+01</td><td>8.409e+01</td><td>9.261886e+00</td><td>1.005697e+01</td><td>1.372e-05</td><td>2.724e-06</td><td>4.219e+05</td></tr>
</table>

And now the results for $k = 0.25$.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>2.5000e-01</td><td>10</td><td>8.023e+01</td><td>5.244e+01</td><td>8.216e+01</td><td>1.686e+02</td><td>6.324584e+01</td><td>1.602583e+02</td><td>5.217e-04</td><td>2.804e-04</td><td>3.361e+06</td></tr>
<tr><td>2.5000e-01</td><td>45</td><td>8.370e+01</td><td>6.629e+01</td><td>8.411e+01</td><td>1.032e+02</td><td>5.177362e+01</td><td>7.913404e+01</td><td>1.942e-04</td><td>6.671e-05</td><td>2.671e+06</td></tr>
<tr><td>2.5000e-01</td><td>177</td><td>8.397e+01</td><td>7.681e+01</td><td>8.407e+01</td><td>8.762e+01</td><td>3.417912e+01</td><td>4.216233e+01</td><td>1.435e-04</td><td>3.548e-05</td><td>1.809e+06</td></tr>
<tr><td>2.5000e-01</td><td>762</td><td>8.398e+01</td><td>8.172e+01</td><td>8.403e+01</td><td>8.443e+01</td><td>1.955547e+01</td><td>2.121283e+01</td><td>3.754e-05</td><td>1.132e-05</td><td>1.021e+06</td></tr>
<tr><td>2.5000e-01</td><td>3109</td><td>8.398e+01</td><td>8.349e+01</td><td>8.400e+01</td><td>8.409e+01</td><td>9.262551e+00</td><td>1.005604e+01</td><td>1.375e-05</td><td>2.818e-06</td><td>4.725e+05</td></tr>
</table>

One can see that the quadrature error has only a marginal affect on the “energy” quantities, which are our primary concern, and hence we use only a single time step for the following adaptive solutions.

For the adaptive solutions we use the same set of tolerances as before: $\mathsf{TOL} = 15$, 12.5, 10, 7.5, 5. The initial mesh is again given by $h = 0.1$, and the results are shown in the table below (with $k = 1$).

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>695</td><td>8.398e+01</td><td>8.331e+01</td><td>8.398e+01</td><td>8.415e+01</td><td>1.061322e+01</td><td>1.186403e+01</td><td>2.015e-05</td><td>3.930e-06</td><td>4.303e+05</td></tr>
<tr><td>1.0000e+00</td><td>711</td><td>8.398e+01</td><td>8.333e+01</td><td>8.398e+01</td><td>8.415e+01</td><td>1.042769e+01</td><td>1.171360e+01</td><td>1.986e-05</td><td>3.928e-06</td><td>4.234e+05</td></tr>
<tr><td>1.0000e+00</td><td>1056</td><td>8.398e+01</td><td>8.359e+01</td><td>8.398e+01</td><td>8.405e+01</td><td>8.086324e+00</td><td>8.724684e+00</td><td>1.217e-05</td><td>2.726e-06</td><td>3.251e+05</td></tr>
<tr><td>1.0000e+00</td><td>1584</td><td>8.398e+01</td><td>8.373e+01</td><td>8.398e+01</td><td>8.402e+01</td><td>6.478367e+00</td><td>7.022869e+00</td><td>8.442e-06</td><td>2.027e-06</td><td>2.612e+05</td></tr>
<tr><td>1.0000e+00</td><td>3064</td><td>8.398e+01</td><td>8.385e+01</td><td>8.398e+01</td><td>8.400e+01</td><td>4.699467e+00</td><td>5.016549e+00</td><td>4.422e-06</td><td>1.241e-06</td><td>1.884e+05</td></tr>
</table>

The adapted meshes for a selection of tolerances and some plots of the stresses are shown in the Figure 8.1.

This example is of course non-physical and is useful only because we know the exact solution. Below we give some more physical examples using material data for a Nylon 6.6 compound.

8.5 Physical examples with "Maranyl"

We assume a synchronous material wherein λ and μ exhibit the same time dependence (which implies a constant Poisson ratio) and take the data as given by (**7**.13) and (**7**.14). For the single stress relaxation function we take $E\varphi(t)$ where,

$$\varphi(t) = \sum_{i=0}^{2} \varphi_i e^{-\alpha_i t},$$

with:

$$\begin{aligned} \varphi_0 &= 0.183\,429\,971 & \qquad \alpha_0 &= 0.0 \\ \varphi_1 &= 0.385\,804\,129 & \qquad \alpha_1 &= 53.821\,223\,820 \\ \varphi_2 &= 0.430\,765\,899 & \qquad \alpha_2 &= 1.592\,948\,754 \end{aligned}$$

Here the φ_i are dimensionless while the α_i have units $(\text{years})^{-1}$. These data are derived from experimental creep response curves for Nylon 6.6 compound "Maranyl", and are taken from [45, Equation (5.39)]. (Note that we have "modernized" the units by using the conversion factor $1\,\text{psi} = 6894.8\,\text{Pa}$.)

8.5.1 Example: L-shaped lever arm

We now consider an example of an L-shaped lever arm as shown in Figure 8.2. The arm is fixed rigidly in both displacements along its leftmost vertical edge. In addition a vertical traction of $-5\,\text{MPa}$ is applied along the horizontal top edge, and a horizontal traction of $-500y\,\text{kPa}$ is applied along the rightmost vertical edge. All other data is as before (e.g. $k = 1$ and $T = 1$).

The first tabulated results are for uniform refinements with $h = 0.4,\ 0.2,\ 0.1, \ldots$.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>11</td><td>0.000e+00</td><td>2.566e+02</td><td>3.629e+02</td><td>2.590e+02</td><td>2.566369e+02</td><td>3.472313e+01</td><td>1.784e-02</td><td>9.893e-03</td><td>3.200e+06</td></tr>
<tr><td>1.0000e+00</td><td>50</td><td>0.000e+00</td><td>3.080e+02</td><td>4.356e+02</td><td>3.109e+02</td><td>3.080140e+02</td><td>4.257241e+01</td><td>2.752e-02</td><td>1.549e-02</td><td>4.162e+06</td></tr>
<tr><td>1.0000e+00</td><td>204</td><td>0.000e+00</td><td>3.543e+02</td><td>5.010e+02</td><td>3.558e+02</td><td>3.542776e+02</td><td>3.285800e+01</td><td>3.701e-02</td><td>2.169e-02</td><td>4.812e+06</td></tr>
<tr><td>1.0000e+00</td><td>890</td><td>0.000e+00</td><td>3.693e+02</td><td>5.223e+02</td><td>3.700e+02</td><td>3.693209e+02</td><td>2.215892e+01</td><td>4.005e-02</td><td>2.329e-02</td><td>5.017e+06</td></tr>
<tr><td>1.0000e+00</td><td>3744</td><td>0.000e+00</td><td>3.764e+02</td><td>5.323e+02</td><td>3.765e+02</td><td>3.763879e+02</td><td>1.080548e+01</td><td>4.145e-02</td><td>2.414e-02</td><td>5.072e+06</td></tr>
<tr><td>1.0000e+00</td><td>15208</td><td>0.000e+00</td><td>3.781e+02</td><td>5.347e+02</td><td>3.781e+02</td><td>3.780725e+02</td><td>7.078156e+00</td><td>4.172e-02</td><td>2.430e-02</td><td>5.094e+06</td></tr>
</table>

The next table of results are for adapted solutions with $\mathsf{TOL} = 32$, 24, 16, 8 and where we again start with an initial mesh of $h = 1.0$. Some of the meshes and plots of the stress surfaces are shown in the Figure 8.2.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>308</td><td>0.000e+00</td><td>3.642e+02</td><td>5.151e+02</td><td>3.653e+02</td><td>3.642289e+02</td><td>2.733388e+01</td><td>3.884e-02</td><td>2.268e-02</td><td>4.936e+06</td></tr>
<tr><td>1.0000e+00</td><td>608</td><td>0.000e+00</td><td>3.715e+02</td><td>5.254e+02</td><td>3.721e+02</td><td>3.714864e+02</td><td>2.074337e+01</td><td>4.034e-02</td><td>2.357e-02</td><td>5.018e+06</td></tr>
<tr><td>1.0000e+00</td><td>1542</td><td>0.000e+00</td><td>3.757e+02</td><td>5.314e+02</td><td>3.760e+02</td><td>3.757406e+02</td><td>1.441982e+01</td><td>4.119e-02</td><td>2.402e-02</td><td>5.065e+06</td></tr>
<tr><td>1.0000e+00</td><td>6460</td><td>0.000e+00</td><td>3.782e+02</td><td>5.348e+02</td><td>3.782e+02</td><td>3.781621e+02</td><td>7.942375e+00</td><td>4.170e-02</td><td>2.430e-02</td><td>5.093e+06</td></tr>
</table>

Here it requires over 15,000 elements to achieve a similar estimated error as the adaptive solution produces with only 6460 elements.

8.5.2 Example: a simple crack

We now consider a simple horizontal crack in a rectangular component. Due to symmetry we consider only the upper half of the component, and load the top edge with a vertical traction of 5 MPa—see Figure 8.3.

Following exactly the same pattern as above, the first tabulated results are for uniform refinements with $h = 0.4,\ 0.2,\ 0.1, \ldots$.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>8</td><td>0.000e+00</td><td>2.130e+02</td><td>3.012e+02</td><td>2.144e+02</td><td>2.129519e+02</td><td>2.522603e+01</td><td>7.835e-03</td><td>2.542e-03</td><td>2.833e+06</td></tr>
<tr><td>1.0000e+00</td><td>34</td><td>0.000e+00</td><td>2.303e+02</td><td>3.256e+02</td><td>2.320e+02</td><td>2.302629e+02</td><td>2.801396e+01</td><td>1.012e-02</td><td>3.225e-03</td><td>3.172e+06</td></tr>
<tr><td>1.0000e+00</td><td>146</td><td>0.000e+00</td><td>2.444e+02</td><td>3.456e+02</td><td>2.453e+02</td><td>2.444114e+02</td><td>2.100032e+01</td><td>1.216e-02</td><td>3.836e-03</td><td>3.356e+06</td></tr>
<tr><td>1.0000e+00</td><td>618</td><td>0.000e+00</td><td>2.514e+02</td><td>3.555e+02</td><td>2.519e+02</td><td>2.514081e+02</td><td>1.620563e+01</td><td>1.309e-02</td><td>4.139e-03</td><td>3.460e+06</td></tr>
<tr><td>1.0000e+00</td><td>2596</td><td>0.000e+00</td><td>2.567e+02</td><td>3.630e+02</td><td>2.569e+02</td><td>2.567010e+02</td><td>9.700642e+00</td><td>1.379e-02</td><td>4.373e-03</td><td>3.540e+06</td></tr>
<tr><td>1.0000e+00</td><td>10394</td><td>0.000e+00</td><td>2.587e+02</td><td>3.659e+02</td><td>2.588e+02</td><td>2.587460e+02</td><td>6.990774e+00</td><td>1.405e-02</td><td>4.462e-03</td><td>3.573e+06</td></tr>
</table>

The next table of results are for adapted solutions with $\mathsf{TOL} = 32$, 24, 16, 8 and where we again start with an initial mesh of $h = 1.0$. Some of the meshes and plots of the stress surfaces are shown in the figure.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>146</td><td>0.000e+00</td><td>2.444e+02</td><td>3.456e+02</td><td>2.453e+02</td><td>2.444114e+02</td><td>2.100032e+01</td><td>1.216e-02</td><td>3.836e-03</td><td>3.356e+06</td></tr>
<tr><td>1.0000e+00</td><td>146</td><td>0.000e+00</td><td>2.444e+02</td><td>3.456e+02</td><td>2.453e+02</td><td>2.444114e+02</td><td>2.100032e+01</td><td>1.216e-02</td><td>3.836e-03</td><td>3.356e+06</td></tr>
<tr><td>1.0000e+00</td><td>301</td><td>0.000e+00</td><td>2.543e+02</td><td>3.597e+02</td><td>2.548e+02</td><td>2.543484e+02</td><td>1.592360e+01</td><td>1.340e-02</td><td>4.255e-03</td><td>3.520e+06</td></tr>
<tr><td>1.0000e+00</td><td>2235</td><td>0.000e+00</td><td>2.596e+02</td><td>3.672e+02</td><td>2.597e+02</td><td>2.596160e+02</td><td>7.157102e+00</td><td>1.413e-02</td><td>4.496e-03</td><td>3.591e+06</td></tr>
</table>

Here it requires over 10,000 elements to achieve a similar estimated error as the adaptive solution produces with only 2235 elements.

8.5.3 Example: webbed angle bracket

Our next example is of a webbed angle bracket as shown in Figure 8.4. The bracket is constrained both horizontally and vertically at the two lower horizontal "pads" and traction-loaded only at the top vertical pad on the right. The horizontal loading is -5 MPa and the vertical -500 kPa.

We again begin by tabulating resukts for uniform refinements with $h = 0.4$, 0.2, $0.1, \ldots$.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>33</td><td>0.000e+00</td><td>1.790e+02</td><td>2.532e+02</td><td>1.798e+02</td><td>1.790362e+02</td><td>1.699838e+01</td><td>1.919e-02</td><td>4.961e-03</td><td>2.203e+06</td></tr>
<tr><td>1.0000e+00</td><td>39</td><td>0.000e+00</td><td>1.845e+02</td><td>2.609e+02</td><td>1.852e+02</td><td>1.844667e+02</td><td>1.681407e+01</td><td>2.007e-02</td><td>5.352e-03</td><td>2.305e+06</td></tr>
<tr><td>1.0000e+00</td><td>78</td><td>0.000e+00</td><td>1.910e+02</td><td>2.701e+02</td><td>1.922e+02</td><td>1.910198e+02</td><td>2.147025e+01</td><td>2.115e-02</td><td>6.009e-03</td><td>2.479e+06</td></tr>
<tr><td>1.0000e+00</td><td>276</td><td>0.000e+00</td><td>2.082e+02</td><td>2.944e+02</td><td>2.091e+02</td><td>2.081676e+02</td><td>1.986391e+01</td><td>2.438e-02</td><td>7.869e-03</td><td>2.717e+06</td></tr>
<tr><td>1.0000e+00</td><td>1488</td><td>0.000e+00</td><td>2.190e+02</td><td>3.097e+02</td><td>2.193e+02</td><td>2.190010e+02</td><td>1.118244e+01</td><td>2.800e-02</td><td>9.279e-03</td><td>2.869e+06</td></tr>
<tr><td>1.0000e+00</td><td>6235</td><td>0.000e+00</td><td>2.220e+02</td><td>3.139e+02</td><td>2.221e+02</td><td>2.219738e+02</td><td>6.972105e+00</td><td>2.910e-02</td><td>9.670e-03</td><td>2.913e+06</td></tr>
<tr><td>1.0000e+00</td><td>25466</td><td>0.000e+00</td><td>2.231e+02</td><td>3.155e+02</td><td>2.231e+02</td><td>2.230782e+02</td><td>4.052005e+00</td><td>2.948e-02</td><td>9.802e-03</td><td>2.925e+06</td></tr>
</table>

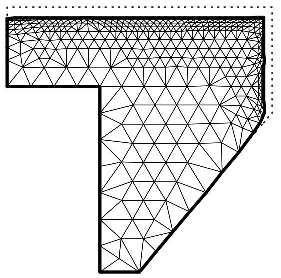
The next table of results are for adapted solutions with $\mathsf{TOL} = 20, 15, 10, 5$ and where we now start with an initial mesh of $h = 0.05$ (due to the slender nature of the component). Some of the meshes and plots of the stress surfaces are shown in the Figure 8.4.

<table>
<tr><th>ki</th><th>N elem</th><th>||u||_E</th><th>||uh||_E</th><th>|uh|+|e|</th><th>|uh|+est|e|</th><th>||u-uh||_E</th><th>est ||e||_E</th><th>inf|u-uh|</th><th>||u-uh||</th><th>||s-sh||</th></tr>
<tr><td>1.0000e+00</td><td>276</td><td>0.000e+00</td><td>2.082e+02</td><td>2.944e+02</td><td>2.091e+02</td><td>2.081676e+02</td><td>1.986391e+01</td><td>2.438e-02</td><td>7.869e-03</td><td>2.717e+06</td></tr>
<tr><td>1.0000e+00</td><td>829</td><td>0.000e+00</td><td>2.179e+02</td><td>3.082e+02</td><td>2.183e+02</td><td>2.179416e+02</td><td>1.306965e+01</td><td>2.758e-02</td><td>9.049e-03</td><td>2.859e+06</td></tr>
<tr><td>1.0000e+00</td><td>2487</td><td>0.000e+00</td><td>2.213e+02</td><td>3.130e+02</td><td>2.215e+02</td><td>2.213019e+02</td><td>9.014411e+00</td><td>2.882e-02</td><td>9.529e-03</td><td>2.903e+06</td></tr>
<tr><td>1.0000e+00</td><td>12232</td><td>0.000e+00</td><td>2.230e+02</td><td>3.154e+02</td><td>2.231e+02</td><td>2.230293e+02</td><td>4.942554e+00</td><td>2.945e-02</td><td>9.777e-03</td><td>2.925e+06</td></tr>
</table>

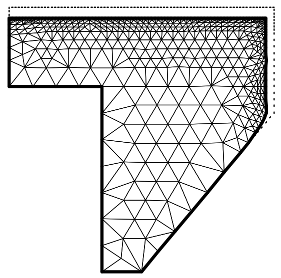
Here it requires over 25,000 elements to achieve a similar estimated error as the adaptive solution produces with only 12,232 elements.

Each of these examples clearly demonstrates how adaptive mesh refinement, guided by an *a posteriori* error estimate, can result in an acceptable solution requiring far fewer elements that would be needed when using uniform meshes.

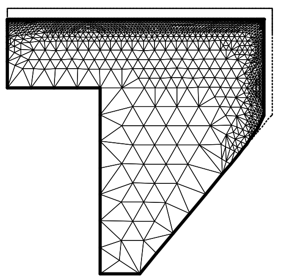
Figure 8.1: Adapted meshes for the exact solution with $\mathsf{TOL} = 15$, 10 and 5 (initial mesh: $h = 0.1$). Also shown are plots of the stress surfaces for $\mathsf{TOL} = 10$.



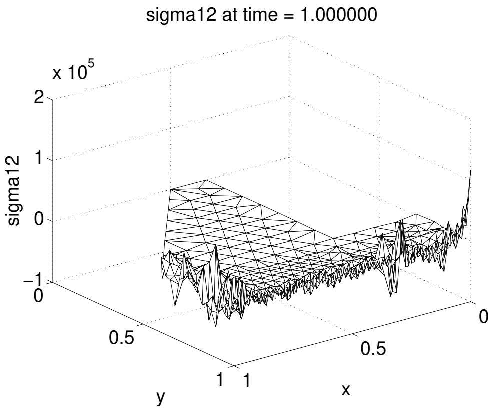
391 nodes, 695 elements
85 boundary sides during
times $t \in (0.000000, 1.000000)$.

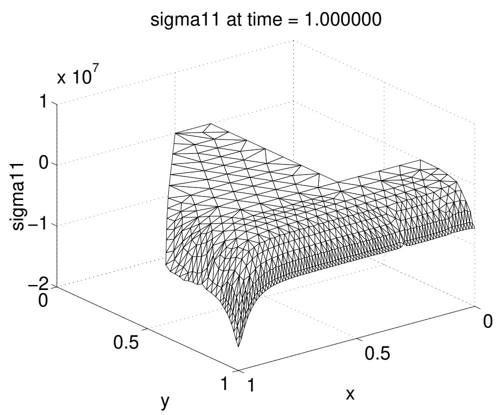


595 nodes, 1056 elements
132 boundary sides during
times $t \in (0.000000, 1.000000)$.



1656 nodes, 3064 elements
246 boundary sides during
times $t \in (0.000000, 1.000000)$.





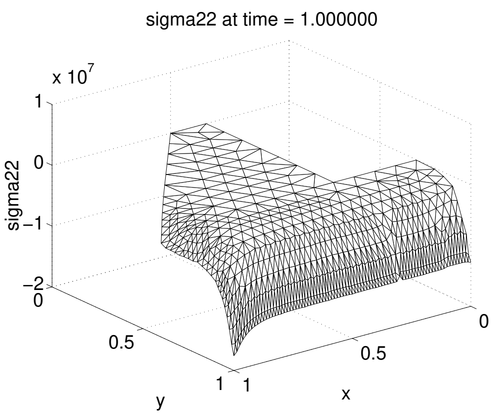
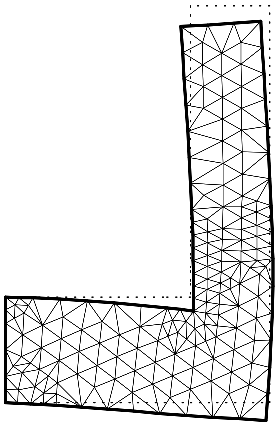
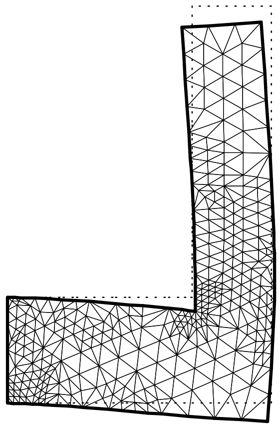


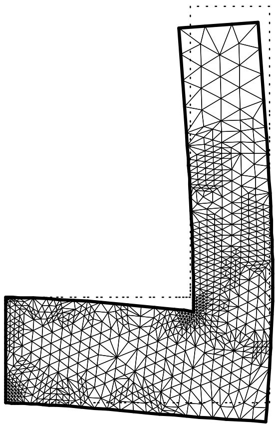
Figure 8.2: Adapted meshes for the L-shaped lever arm with $\mathsf{TOL} = 32$, 24 and 16 (initial mesh: $h = 0.1$). Also shown are plots of the stress surfaces for $\mathsf{TOL} = 8$.



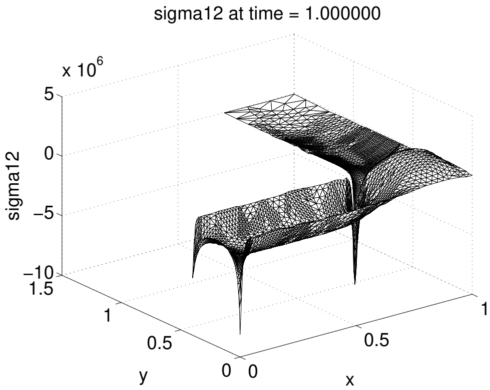
186 nodes, 308 elements
62 boundary sides during
times $t \in (0.000000, 1.000000)$.

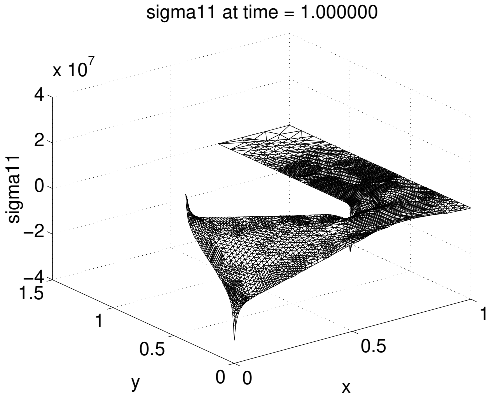


347 nodes, 608 elements
84 boundary sides during
times $t \in (0.000000, 1.000000)$.



836 nodes, 1542 elements
128 boundary sides during
times $t \in (0.000000, 1.000000)$.





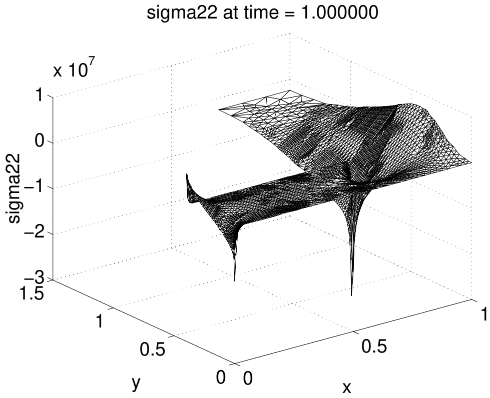
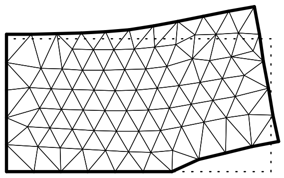
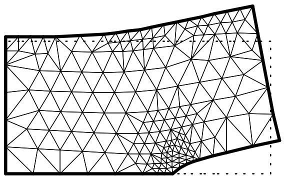


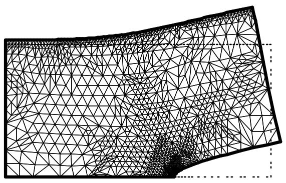
Figure 8.3: Adapted meshes for the simple symmetric crack with $\mathsf{TOL} = 24$, 32 and 8 (initial mesh: $h = 0.1$). Also shown are plots of the stress surfaces for $\mathsf{TOL} = 8$.



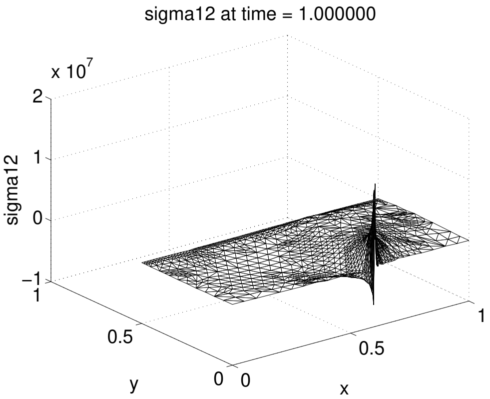
89 nodes, 146 elements
30 boundary sides during
times $t \in (0.000000, 1.000000)$.

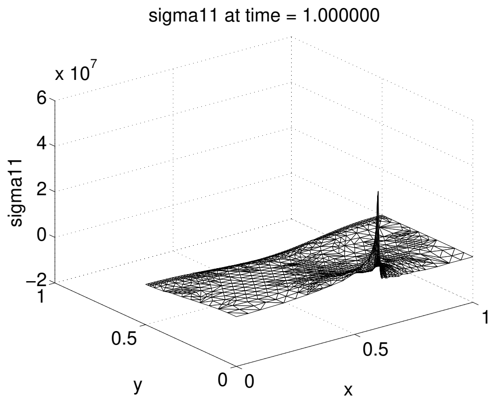


174 nodes, 301 elements
45 boundary sides during
times $t \in (0.000000, 1.000000)$.



1191 nodes, 2235 elements
145 boundary sides during
times $t \in (0.000000, 1.000000)$.





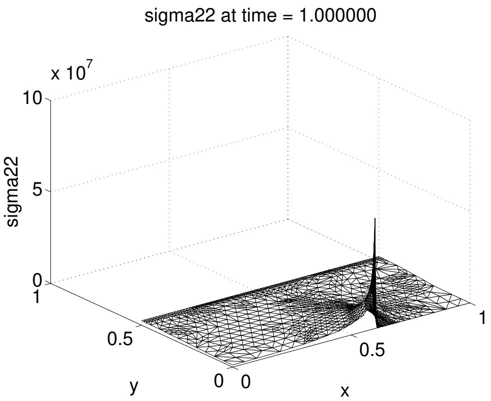
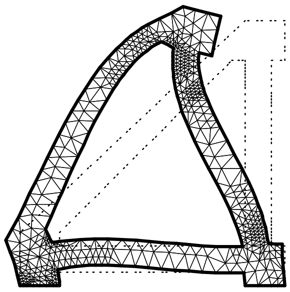
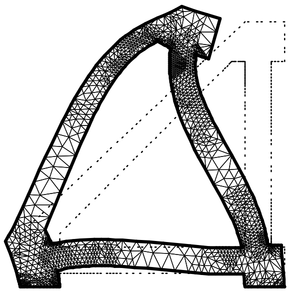


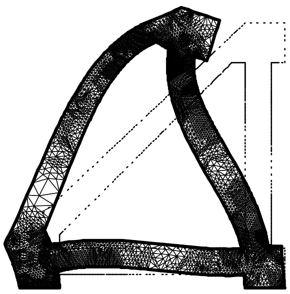
Figure 8.4: Adapted meshes for the webbed angle bracket with $\mathsf{TOL} = 15$, 10 and 5 (initial mesh: $h = 0.05$). Also shown are plots of the stress surfaces for $\mathsf{TOL} = 10$.



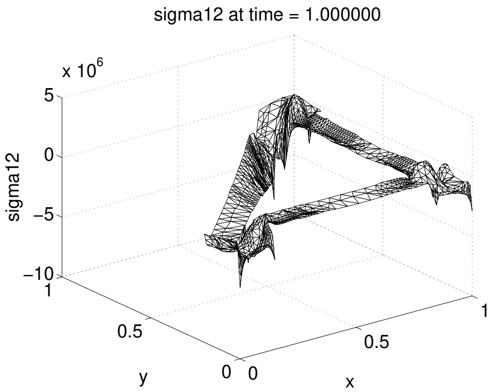
507 nodes, 829 elements
185 boundary sides during
times $t \in (0.000000, 1.000000)$.

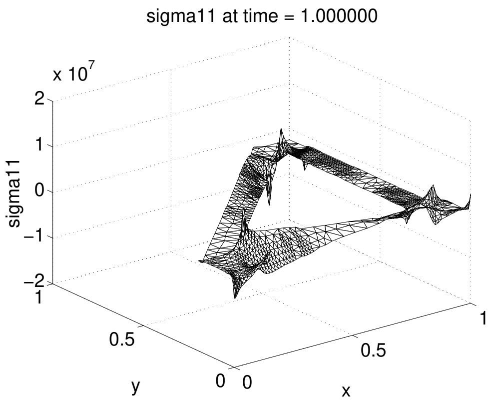


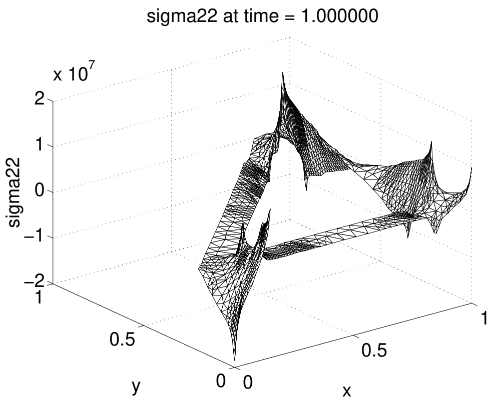
1393 nodes, 2487 elements
299 boundary sides during
times $t \in (0.000000, 1.000000)$.



6445 nodes, 12232 elements
658 boundary sides during
times $t \in (0.000000, 1.000000)$.







Part III

Closure

Chapter 9

Obtaining and using the software

9.1 Introduction

The C code used to generate the numerical solutions detailed in the previous chapters is available via `ftp`. This chapter describes how to retrieve the source files and compile and run them on an X-window Unix platform. We also give a short "manual" describing how the code can be set up to solve a specific problem. We do this by choosing a simple example problem and demonstrating how the input files should be generated to model the domain and boundary conditions. Since the software comes packaged with the example problems used earlier in this report, these together with this manual should provide all the necessary information.

NOTE: this software is what we term a "research code". It has been developed in a piecemeal fashion over a long period of time as and when new research results become available and are suitable for implementation. Consequently, no claims are made for it being efficient or robust, and it should **not** be used in a situation where its output may have a safety or financial implication.

9.2 Obtaining the software

Firstly, connect to the Brunel University `ftp` server via the command,

```
ftp ftp.brunel.ac.uk
```

When prompted logon as "`anonymous`" and enter your email address as a password. When you get the prompt issue the following commands:

```
cd icsrsss
bin
get seedproj.tar
quit
```

You should now have a binary file called `seedproj.tar` in your local working directory. This is a tape-archived record of the source codes created using the Unix `tar` utility.

The next step is to extract the directory structure. Do this with `tar` using the command,

```
tar xvf seedproj.tar
```

You now should have a directory called `seedproj` which contains the sub-directories holding the source codes for the mesh generator and finite element solver. At this point the archive file `seedproj.tar` can be deleted.

To compile the mesh generator issue the following commands:

```
cd seedproj/advance/source
make
```

and to compile the solver type,

```
cd ../../quasivis/native/source
make
```

There is also a very simple X-window plotting program called `Xwin` which will be useful later when generating the mesh. To compile this type,

```
cd ../../../Xwin
make
```

If this stage has gone well you should have three executable files, `advance`, `fem` and `Xwin` in the directory,

```
seedproj/bin
```

Now change to the directory `seedproj/quasivis/data` and list the contents. You will see subdirectories such as `leverarm`, `crack`, etc. These are the input files for the numerical examples shown in previous chapters. The source files are supplied and set up so as to solve the problem with an exact solution (see previously in Subsection 8.4.2) and to check that the installation was successful type,

```
adapt exact 15
```

This executes the `sh` script file `adapt` and solves the problem, with $\mathsf{TOL} = 15$, as specified by the files in the `exact` directory. Specifically the mesh generator `advance` builds the mesh according to the files in the `exact` directory, and then `adapt` launches the finite element solver `fem`. After the `make` (i.e. compilation) stage you should see an X-window popped to the screen showing a sequence of adapted meshes. You will also see many lines of text rolling around the terminal screen. The end of the textual output should look like

```
   ki        N elem  ||u||_E     ||uh||_E    |uh|+|e|    |uh|+est|e|  ||u-uh||_E     est ||e||_E   inf|u-uh|  ||u-uh||   ||s-sh||
 1.0000e+00  695     8.398e+01   8.331e+01   8.398e+01   8.415e+01   1.061322e+01   1.186403e+01  2.015e-05  3.930e-06  4.303e+05
Tue Dec  1 17:15:34 GMT 1998
```

(although you will certainly have a different time stamp showing). Note that these figures agree with those shown in the first line of the table of adapted solutions in Subsection 8.4.2. If all of this happens the installation went well, and you may now define your own domain and quasistatic viscoelasticity problem.

9.3 Defining the domain

As the name `advance` suggests, the mesh generator is a (simple) implementation of the **advancing front** technique. For this it is necessary only to define the boundary of the domain according to a simple **orientation rule**, and provide a function $h = h(x, y)$ that describes the desired mesh size variation over the domain. In our implementation $h = h(x, y)$ is specified as a constant h_0, which we term the **basic mesh size**, and local mesh features and size control are effected through **source points** (in the file `srcpnts.dat`), **source lines** (in the file `srcline.dat`), **source circles** (in the file `srccirc.dat`) and **source discs** (in the file `srcdisc.dat`). We'll get to these later but for now note that in this way an *a priori* graded mesh can be created by suitably editing one of these input files rather than hard coding a C function, `double h(double x, double y)`, and then having to re-compile each time.

At the moment only two-dimensional polygonal domains are supported by `advance`, with the boundary defined piecewise by the endpoints of straight-line segments. The domain can however be multiply-connected (i.e. have an arbitrary number of "holes"), and it would not be difficult in the future to incorporate curved boundaries.

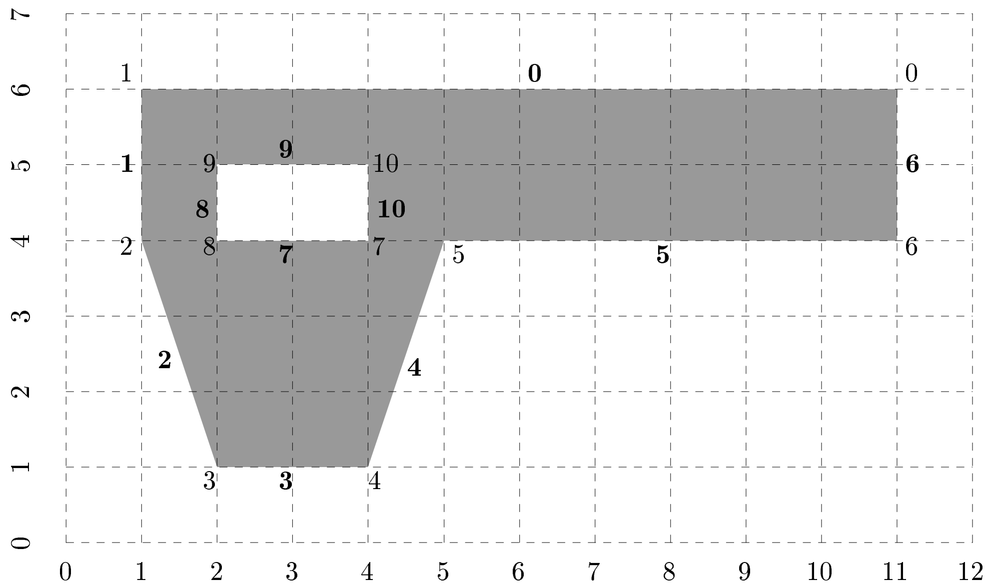
To illustrate how to set up a domain in the way required for `advance` we will consider the case of a trestle-shaped structure with—for illustration purposes only—a hole in it. The domain is shown in Figure 9.1 and is supposed to symmetrical about the vertical edge at $x = 11$cm. The lowest horizontal edge is constrained to have no vertical displacement and, to impose the symmetry, the right-most vertical edge is constrained to have no horizontal displacement. We assume that a load (i.e. surface traction) of $g_2 = -1\,\mathrm{MPa}$ acts vertically downward on the uppermost horizontal edge.

Each corner of the trestle has a numbered node associated with it and in between each node we identify a numbered (shown in bold) boundary edge. Note that the node and edge numbering **must** begin at zero (this is because all arrays begin with a zero subscript in the C language). Also, although the node and edge numbering follow a simple counter-clockwise pattern in this example, there is absolutely no need for this. The crucial thing is that the **orientation rule** which we describe below is observed when defining the edge connectivity.

Before constructing the input files we need to assign the correct **boundary condition codes** to the nodes and edges. These integer values describe the type of constraint (if any) that the node or edge is subject to, and are given by:

0 — unfixed (for interior nodes only).

Figure 9.1: The left half of the trestle domain. (Measurements are in cm.)



1 — Non-homogeneous Dirichlet condition - unused at the moment.

2 — Homogeneous Dirichlet condition (zero displacement).

3 — Non-homogeneous Neumann condition (prescribed traction).

4 — Homogeneous Neumann condition (zero traction).

Each node and each edge is now assigned a triple of integers made up from the above codes (except for 0 — unfixed, this is reserved for interior nodes created by the mesh generator). The first two elements of the triple correspond to displacements or tractions in the x and y direction respectively, while the third element is unused at present (and so its value is irrelevant).

This works in the following way. Nodes 3 and 4, and edge 3, at the foot of the trestle are to be constrained so as to have no vertical displacement and this is accomplished by setting a 2 in the second element of the triple. Also, we'll assume that there is no traction acting horizontally along edge 3 and so the first element of the triple is set as 4. For this part of the boundary we then get,

boundary codes for node 3 are: 4 2 0
boundary codes for node 4 are: 4 2 0
boundary codes for edge 3 are: 4 2 0

Now, for edge 0, defined by nodes 0 and 1, we assume zero horizontal traction and this leads to us setting the first element of the triple as 4. On the other hand we have a non-zero vertical traction acting on this edge and so the second element is set as 3. This gives:

boundary codes for edge 0 are: 4 3 0

By symmetry node 0 is fixed horizontally – code 2 – but is subject to vertical traction – code 3. Thus:

boundary codes for node 0 are: 2 3 0
boundary codes for node 1 are: 4 3 0

since node 1 is free and unforced in the x direction.

Now let us take a look at edge 6 as defined by nodes 6 and 0. Due to symmetry this edge may not move horizontally – code 2 – and must be free of vertical traction – code 4. We get the same for node 6 and so:

boundary codes for node 6 are: 2 4 0
boundary codes for edge 6 are: 2 4 0

Every other node and edge is free to move (so we cannot use either of codes 1 or 2), but is also traction-free. Thus for all other nodes and edges we set the boundary code as: 4 4 0.

Equipped with this information we now create the first input files for the mesh generator. Inside the `seedproj/quasivis/data` directory create a new subdirectory called `trestle`. Change into this new directory and edit the files `defstart.inf`, `defnodes.inf` and `defsides.inf` as shown below (do not include the underlined headings).

```
                    The input file ‘defstart.inf’
                    -----------------------------
Number_of_nodes: 11
Number_of_sides: 11
bounding_box_for_graphics   0.0  0.12  -0.03  0.09
```

The file `defstart.inf` simply shows the number of nodes and (necessarily the same for polygonal domains) edges used to define the domain—in this case 11. The file also defines a **bounding box** to control the graphical display of the domain. This box is defined by a line of the form,

`bounding_box_for_graphics` x_{low} x_{high} y_{low} y_{high}

which describes the box by the diagonal running from $(x_{\mathrm{low}}, y_{\mathrm{low}})$ to $(x_{\mathrm{high}}, y_{\mathrm{high}})$. The bounding box is assumed to be square by all of the graphics routines and so it is necessary to ensure that

$$x_{\mathrm{high}} - x_{\mathrm{low}} = y_{\mathrm{high}} - y_{\mathrm{low}}.$$

If this does not hold then the display will be distorted by a relative scale difference in the horizontal and vertical directions. For our example we will specify all lengths in metres and so we choose a bounding box with the line,

```
bounding_box_for_graphics   0.0  0.12  -0.03  0.09
```

The domain fits inside this bounding box and so should appear centered in the graphical displays. By altering the bounding box one can scale the x and y dimensions of the domain in different ways or place it off-center, as well as reducing its apparent distance from the eye.

```
                     The input file 'defnodes.inf'
                     -----------------------------

0    2 3 0   0.11  0.06
1    4 3 0   0.01  0.06
2    4 4 0   0.01  0.04
3    4 2 0   0.02  0.01
4    4 2 0   0.04  0.01
5    4 4 0   0.05  0.04
6    2 4 0   0.11  0.04
7    4 4 0   0.04  0.04
8    4 4 0   0.02  0.04
9    4 4 0   0.02  0.05
10   4 4 0   0.04  0.05
```

The file `defnodes.inf` contains the numbered (in order) list of nodes defining the domain, with each integer label followed by the boundary code triple and the node coordinates. There is nothing special about the formatting or spacing of the fields on each line, so long as at least one whitespace character separates each.

```
                     The input file 'defsides.inf'
                     -----------------------------

0  4 3 0   0  1
1  4 4 0   1  2
2  4 4 0   2  3
3  4 2 0   3  4
4  4 4 0   4  5
5  4 4 0   5  6
6  2 4 0   6  0
7  4 4 0   7  8
8  4 4 0   8  9
9  4 4 0   9  10
10 4 4 0   10 7
```

The file `defsides.inf` is similar to `defnodes.inf`; it contains the numbered list of edges defining the domain. On each line the integer edge label is followed by the boundary condition triple, and then by the pair of nodes that define the edge. For example, the line

```
3  4 2 0   3  4
```

shows that edge 3 takes the boundary codes 4 2 0 and is defined by nodes 3 and 4. The order in which these two nodes appear is crucial and must obey the **orientation rule**. This rule is simple enough: it says that in travelling along the edge from the first node to the second node the domain must be on the left. For this reason all the edges on the outer boundary (i.e. 0 to 6) are described by a counter-clockwise node ordering, while all the edges on the interior boundary (i.e. edges 7 to 10) are described by a clockwise ordering.

9.4 Grading and generating the mesh

As far as the mesh generator `advance` is concerned the domain is now completely defined, but we cannot yet use it to create the mesh because we have to specify the mesh size function $h = h(x, y)$. This function should return the desired size of the triangles in the vicinity of the point (x, y). To do this we need the files `srcpnts.dat`, `srcline.dat`, `srccirc.dat` and `srcdisc.dat`. These files can be used to specify the basic value of h throughout the domain, as well as any local refinements that are *a priori* required. You may find it easier to simply copy these files over from another directory (e.g. `../crack`) since they will contain helpful annotations at the bottom that describe the meaning of the information contained in them.

Once you have copied them across edit the files `srcline.dat`, `srccirc.dat` and `srcdisc.dat` and make sure that the very first entry on the first line is the integer 0. We'll return to these files later but for now concentrate on the most important one: `srcpnts.dat`.

The first entry in `srcpnts.dat` is h_0, the basic value of h to be used throughout the domain. Looking back at Figure 9.1 it seems reasonable to require an initial mesh consisting of triangles with side length $h_0 = 0.005$ metres. To obtain this we edit `srcpnts.dat` so that the first line contains the value `0.005`, and ensure that the second line contains the integer `0`. We may now generate the mesh. To do this issue the command (in the directory `trestle`),

```
../../../bin/advance | ../../../bin/Xwin
```

This initiates the mesh generator `advance` and pipes the output into the simple X-window utility `Xwin`. In this case an X-window should pop to the screen and the element edges are drawn in this window as they are created. When the mesh is complete place the mouse cursor in the X-window and type the `q` button. The programs will quit and the prompt will return in the terminal window. (Note that `Xwin` is a very primitive X-window application and so the terminal may display "junk" output while it is running.)

If you now list the files in the `trestle` subdirectory you should see something like:

```
bdystart.fem    defsides.inf    meshinfo.fem    srccirc.dat
boundary.fem    defstart.inf    mon.out         srcdisc.dat
config.inf      element.fem     neighbor.fem    srcline.dat
defnodes.inf    memory.dat      nodes.fem       srcpnts.dat
```

All except the `def*.inf` and `src*.dat` files that we created earlier are disposable (in that they can be regenerated), and so can eventually be removed with a command line like,

```
rm *.fem memory.dat mon.out scratch.dat
```

(Note that the files `mon.out` and `scratch.dat` may or may not be present, depending on how `advance` terminated. These files are always disposable and may use up a lot of disk space—it is advisable therefore to always remove them.) The files `*.fem` are used by the finite element code to determine the initial mesh. Now we have these `*.fem` files we can set up and execute the finite element calculation in the next section, but for the remainder of this section we'll explain a little more about how to use the `src*.dat` files to control *a priori* mesh grading.

The mesh that appeared on the screen should have looked like that on the left in Figure 9.2 and, one might think, is perfectly acceptable for the initial mesh in an adaptive calculation. However, it is also true that an adaptive calculation can be much improved by a sensible choice of initial mesh. For example, for the trestle it is likely that high stresses will be present at the re-entrant corner at $(0.05, 0.04)$, and so it is sensible to build this information into the initial mesh by asking for a finer mesh grading at this point. This can done by introducing a **source point** at this corner by making an entry in the file `srcpnts.dat`. Edit this file so that the first three lines are as shown below.

```
0.005
1
0.05     0.04      0.02     0.002
```

Recall that the first entry is no more than $h_0 = 0.005$m. The second entry states that the mesh is to contain one source point, and the third line then defines this source point to be located at $(0.05, 0.04)$, to have a radius of influence of 0.02m and to have a local mesh size at $(0.05, 0.04)$ of 0.002m. The mesh generator will now attempt to create a mesh with this local size at the re-entrant corner, blending smoothly to the basic mesh size throughout a circle of radius 0.02m. Run the mesh generator again with the command,

```
../../../bin/advance | ../../../bin/Xwin
```

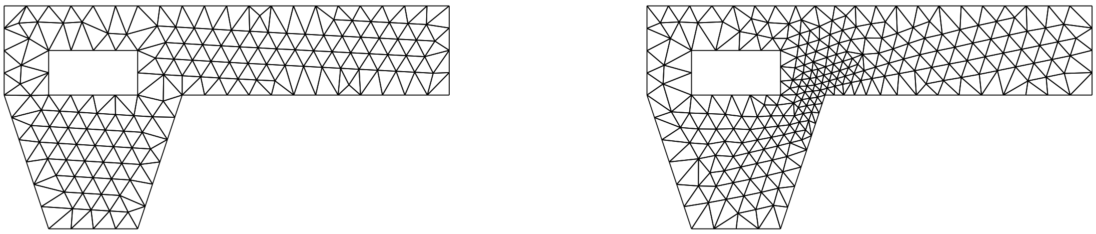
and you should now get the mesh shown on the right of Figure 9.2.

On an intuitive level this seems a much more sensible choice for the initial mesh, and it is simple to specify through the use of the input file `srcpnts.dat`. Any number of source points can be added in this way, with the total number given in the second line of the file, so long as two simple rules are followed. The local mesh size should always be smaller than the basic mesh size, and the radius of influence should be at least twice the basic mesh size.

Source lines, source circles and source discs can be specified in a similar way through the input files `srcline.dat`, `srccirc.dat` and `srcdisc.dat`. To illustrate this edit the file `srcline.dat` so that the first five lines are given by,

```
4
```

Figure 9.2: Trestle mesh generated by `advance`. The mesh on the left has a basic value of $h_0 = 0.005$m throughout while the one on the right has a source point at $(0.05, 0.04)$.



```
0.04 0.04 0.02 0.04  0.005 0.002
0.02 0.04 0.02 0.05  0.015 0.002
0.02 0.05 0.04 0.05  0.015 0.002
0.04 0.05 0.04 0.04  0.005 0.002
```

This now specifies a source line for each edge of the rectangular "hole" in the trestle. The annotations in the file explain what the entries on each line actually mean, and running the mesh generator again gives the mesh shown on the left of Figure 9.3.

The mesh now appears to be suitable for input to the finite element code and indeed this is the one we will use below. However, we will give one more example of how to achieve local mesh grading. To create a source circle of radius 0.008m at the point $(0.08, 0.05)$, with a local mesh size 0.001m and radius of influence 0.01m, edit the file `srccirc.dat` so that the first two lines are:

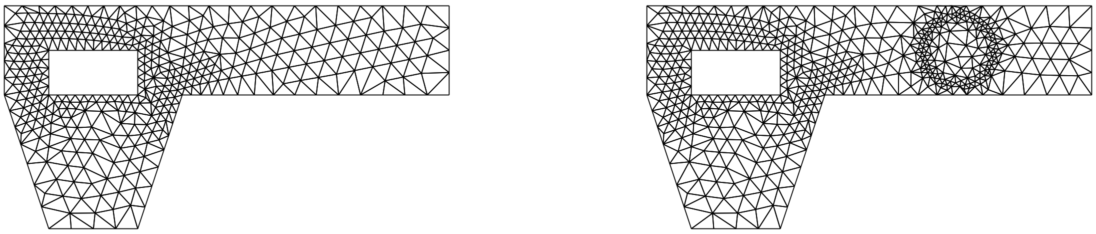
```
1
0.08    0.05    0.008    0.01     0.001
```

The first line specifies how many source circles there are while the subsequent lines (in this only one) specify the circles themselves. The resulting mesh is shown on the right of Figure 9.3. We have included this source circle for demonstration purposes only, it will be of no further use so change the `1` in the first line in the file back to `0` to switch the feature off.

9.5 The finite element calculation

We now have our initial mesh, the one on the left of Figure 9.3, and so we are ready to feed it in to the finite element code. However, to drive the solver we must first provide the file `config.inf` in the `trestle` directory. This file contains basic configuration parameters

Figure 9.3: Trestle mesh generated by `advance`. The mesh on the left has source lines around the perimeter of the square hole while the one on the right shows the effect of a source circle.



for the code and, again, is best created by first copying an existing version over from another directory (e.g. `../crack`) and then editing it so that it appears as below.

```
1.0    = k_init (used only if not set on command line)
1.0    = T, the final time in the time interval (0,T)
1.0e-7   Conjugate gradient residual tolerance
1.0e-2   Error tolerance, non-specific at the moment
5 5 400 400  graphics window: x, y, width, height
2.0    magnification scale for graphical output
15     (or 25) minimum permitted angle in the mesh (in degrees)
150    (or 120) maximum permitted angle in the mesh (in degrees)
5      "memscale" controls memory allocation for the mesh



This file contains algorithm parameters.
```

Note that only the numeric data appearing on the left of each line are used, the text to the right being a short description of what the data are. We'll describe the meaning of each of these in a little more detail.

`k_init`: This is the initial time step—it can be overridden by a command line option.

`T`: The final time to calculate to where the time interval is assumed finite: $\mathcal{J} := [0, T]$.

Since the code is not yet ready to perform adaptive time stepping we'll assume a calculation over the short time interval $[0, 1]$ in a single time step $k_{\text{init}} = T = 1$. The remaining items in this file are described below.

`1.0e-7`: The residual tolerance to use as a stopping condition in the conjugate gradient solver.

`1.0e-2`: This field is unused

`5 5 400 400`: These integers control the location and size of the X-window that pops onto the screen. In this example the window will have its top and left hand edges each 5 pixels from the edge of the screen, and will be 400 pixels square.

`2.0`: In the graphical display of results the displacements are shown on the mesh magnified by this factor.

`15`: The minimum tolerable interior angle in a triangle—used by the mesh adapting routines to determine whether a triangle should undergo a "red" or "green" refinement.

`150`: The maximum tolerable interior angle in a triangle—as above.

`5 "memscale"`: When the solver begins to execute it needs to set aside enough memory to store the mesh. However, in an adaptive calculation it is not known *a priori* how much memory the final mesh will require. Thus a mesh `memscale` times denser than the initial mesh is assumed. This field can be overridden on the command line as we show below, and in future versions of the code we will probably implement a mesh reallocation routine so as to exploit the ability of C to perform dynamic memory allocation. This parameter will then be redundant.

Now we come to the least user-friendly part of the set-up, specifying the loads. Change into the directory `seedproj/quasivis/native/source`. Here you will find all the C source code for the finite element solver. The file that we need to edit to specify the loads is `lesol.c`. Unfortunately a moderate amount of C coding is required here, and below we'll go over the main points. Load `lesol.c` into an editor and look at the top of the file...

```
#include "always.h"
#include "femdata.h"

extern Prony lambda, mu;

/*/
/*  Test exact solutions for a linear elasticity problem
/*  Possible solutions are selected by an appropriate #define.
/*  The options are:
/*
/*  #define EXACT       : trigonometric displacements - see notes.
/*  #define LEVERARM    : actual problem, nothing known
/*  #define CRACK       : actual problem, nothing known
/*  #define WEB         : actual problem, nothing known
/*  #define TRESTLE     : actual problem, nothing known
/*/

#define EXACT  /* ... MAKE THE CHOICE */
 ...
```

The first `#include` statements are directives to the compiler to include the contents of the **header files** `always.h` and `femdata.h`. These can also be found in the directory `native/source` and themselves contain other `#include` directives. They need not worry us. The `extern` statement that follows next is there merely to tell the compiler that two variables, `lambda` and `mu`, of type `Prony` are referenced in this file but are actually defined elsewhere (in the file `viscous.c` in fact). This need not trouble us either. The next few lines, those beginning with `/*` are comments and are ignored by the compiler. This is because any material enclosed between an opening `/*` and a closing `*/` is a comment in C and therefore plays no role in the executable code. Thus, the next non-trivial line in the file, and the one of concern to us, is

```
#define EXACT  /* ... MAKE THE CHOICE */
```

This line tells the compiler that the string `EXACT` has a special significance in the source code that follows. As the comments above this line suggest, and with regard to the numerical results presented earlier, it is also possible to substitute `EXACT` with:

`EXACT` for the exact solution;

`LEVERARM` for the L-shaped domain;

`CRACK` for the simple crack problem;

`WEB` for the webbed bracket.

`TRESTLE` for our trestle problem.

In fact, once the role this `#define`d string plays in the following code is appreciated, you may `#define` any string of your choice in order to specify your problem. To solve our example trestle problem change this line to:

```
#define TRESTLE  /* ... MAKE THE CHOICE */
```

The next part of the file `lesol.c` can be ignored now until you come to the lines

```
/*******************************************************************/
/*                          Displacements                          */
/*******************************************************************/
/* horizontal displacements */

real uxyt(x,y,t)
real x,y,t;
{
    real total = ZERO;
#if defined(EXACT)
    total = X(x,y) * TIME(t);
#elif defined(LEVERARM)
    total = (real)0.0;
#elif defined(CRACK)
    total = (real)0.0;
```

```
#elif defined(WEB)
    total = (real)0.0;
#elif defined(TRESTLE)
    total = (real)0.0;
#else
    fprintf(stderr,"\n\t No solution defined\n\n");
#endif

    return total;
}
```

This declares `uxyt` to be a C function that takes three real numbers x, y and t, giving position and time, and returns, via the `return` statement, a real number—held in the `real` variable `total`. (C programmers note: the type `real` is used here, via `typedef`, as a synonym for `double`.) This function `uxyt` should return the horizontal displacement $u(x, y, t)$ when it is known (for the `EXACT` solution only) or zero (for all the others: `LEVERARM`, `CRACK`, `WEB`, `TRESTLE`). Notice how the `#define`d token `EXACT` is used by the compiler here in conjunction with the "if defined" (`#if defined`) and "else if defined" (`#elif defined`) directives. The meaning and use of these is intuitive: since we had already `#define`d the token `EXACT` then the compiler only built the line

```
total = X(x,y) * TIME(t);
```

into the executable. (The symbols `X` and `TIME` are themselves defined further up the code and need not worry us). The variable `total` is then set to a real value and `return`ed as such every time the function is called. The important point to realize here is that the choice of which branch of the `#if` — `#elif` to build in is made at **compile time**, and so if the `#define`d token is changed the code must be re-compiled to reflect this. Thus our newly `#define`d token `TRESTLE` requires that `lesol.c` be re-compiled and re-linked to produce an updated executable—this will happen automatically when we execute the `adapt` script below.

As we look further down the source code we will see many other functions, each constructed in a similar way. The "listing" of the function `uxyt()` given above shows how we include a new `TRESTLE` branch into the `#if` — `#elif` directive and this must be repeated for every other function defined the following list.

`uxyt` returns `real` value of horizontal displacement $u(x, y, t)$ or zero if this is unknown;

`uxyt_x` returns `real` value of $u_x(x, y, t)$ or zero if this is unknown;

`uxyt_y` returns `real` value of $u_y(x, y, t)$ or zero if this is unknown;

`vxyt` returns `real` value of vertical displacement $v(x, y, t)$ or zero if this is unknown;

`vxyt_x` returns `real` value of $v_x(x, y, t)$ or zero if this is unknown;

`vxyt_y` returns `real` value of $v_y(x, y, t)$ or zero if this is unknown;

`f1xyt` returns `real` value of horizontal body force $f_1(x, y, t)$;

`f2xyt` returns `real` value of vertical body force $f_2(x, y, t)$;

`Sigma11xyt` returns `real` value of direct stress $\sigma_{11}(x, y, t)$ or zero if unknown;

`Sigma22xyt` returns `real` value of direct stress $\sigma_{22}(x, y, t)$ or zero if unknown;

`Sigma12xyt` returns `real` value of shear stress $\sigma_{12}(x, y, t)$ or zero if unknown;

`Epsilon11xyt` returns `real` value of direct strain $\varepsilon_{11}(x, y, t)$ or zero if unknown;

`Epsilon22xyt` returns `real` value of direct strain $\varepsilon_{22}(x, y, t)$ or zero if unknown;

`Epsilon12xyt` returns `real` value of shear strain $\varepsilon_{12}(x, y, t)$ or zero if unknown;

`g1xyt` returns `real` value of horizontal traction $g_1(x, y, t)$;

`g2xyt` returns `real` value of vertical traction $g_2(x, y, t)$;

The other functions in the source code are connected with the relaxation functions λ and μ and can be accepted as they stand.

In our trestle example the only non-zero traction is vertical, of of $-1\,\mathrm{MPa}$ in magnitude, and applied along only one edge. Thus we make sure that the function `g1xyt` contains the lines,

```
...
#elif defined(TRESTLE)
    total = (real)0.0;
#else
...
```

since there are no horizontal tractions, and also that the function `g2xyt` contains

```
...
#elif defined(TRESTLE)
    total = -(real)1.0e6;
#else
...
```

(The type cast `(real)` is not mandatory in these statements—it can be ignored and left out if desired.) Notice that we do not have to specify that g_2 is only non-zero for the specific coordinates (x, y) occurring on edge 0—this was taken care of with the boundary condition codes supplied to the mesh generator. Since this is the only edge with the non-zero traction boundary code 3, this C function is only invoked for element edges occurring on this boundary edge.

The remaining task is to define the material properties. At the moment the code is still set to use the values given by (**7**.13) to (**7**.16). However, to use the more realistic values for Maranyl given in Section 8.5 we need to make a small change to another `#define` directive. This one is near the top of the source file `seedproj/quasivis/native/source/viscous.c`. Edit this file and change the line

```
#define EXACT
```

to

```
#define MARANYL
```

Upon compilation, the declarations at the head of this code now set up two structure-type variables that describe the Maranyl Nylon 6.6 relaxation function.

We are now ready to run the code. Change to the `seedproj/quasivis/data` directory and list the files. You should see the directories `exact`, `crack`, `leverarm`, `web` and our new one `trestle`, and also two `/sbin/sh` shell scripts `adapt` and `run`. Do not attempt to use the latter, it was written (rather badly) in order to automate the generation of the numerical results given earlier. Instead we shall use the simpler `adapt` script. Type,

```
adapt trestle 1.5
```

This moves all working files into the `dump` directory; sets the `$PATH` variable so the shell can find the executables `advance`, `Xwin` and `fem`; generates the mesh according to the `*.inf` and `*.dat` files in the `trestle` directory; `make`s and links any updated source code; and, finally launches the finite element solver `fem` with the command,

```
fem -X -memscale 100 -adapt $TOL 1.0
```

Here the argument `-X` requests the X-window graphics, while `-memscale 100` allocates memory for a mesh a hundred times more dense that the starting mesh (the one given earlier on the left of Figure 9.3). The argument `-adapt` tells the code to solve in the adaptive mode with tolerance TOL given by the third command line argument to the `adapt` script. In this case the environment variable `$TOL` has the value $\mathsf{TOL} = 1.5$. The final value `1.0` tells the code to apply 100% of this tolerance to spatial error control and none to temporal error control. Eventually, when some form of temporal adaptivity is implemented, it will be possible to have a command-line option to `fem` taking the form,

```
-adapt 1.2 0.4
```

This would specify a global tolerance of $\mathsf{TOL} = 1.2$ with only 40% (i.e. $\mathsf{TOL}_\Omega = 0.48$) going toward controlling the spatial errors via the residual term $\mathcal{E}_\Omega$. However, until temporal adaptivity is implemented the second value for the `-adapt` flag should always be `1.0`.

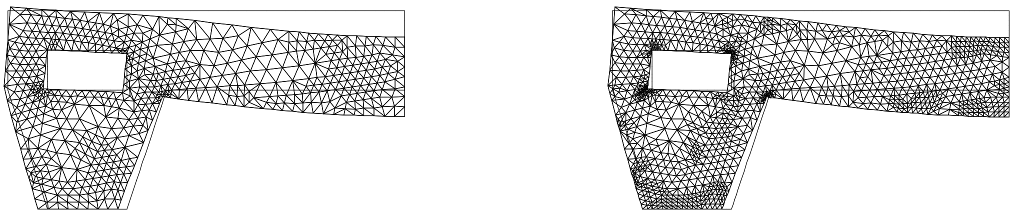
Executing the command

```
adapt trestle 1.5
```

should produce the output lines

```
    ki       N elem  ||u||_E     ||uh||_E    |uh|+|e|    |uh|+est|e|  ||u-uh||_E     est ||e||_E   inf|u-uh|  ||u-uh||   ||s-sh||
1.0000e+00   1123    0.000e+00   1.936e+01   2.738e+01   1.939e+01    1.935795e+01   1.138819e+00  3.343e-03  9.364e-05  2.670e+05
```

Figure 9.4: Adapted meshes for the trestle problem with $\mathsf{TOL}_\Omega = 1.5$ on the left and $\mathsf{TOL}_\Omega = 1.0$ on the right.



(although you'll need a wide terminal to see this clearly), as well as the adapted mesh shown on the left of Figure 9.4.

To see the effect of the adaptivity reduce the tolerance to 1.0 and execute again with the line

```
adapt trestle 1.0
```

This produces the output lines,

```
    ki        N elem  ||u||_E     ||uh||_E    |uh|+|e|    |uh|+est|e| ||u-uh||_E     est ||e||_E   inf|u-uh|  ||u-uh||   ||s-sh||
 1.0000e+00  2097    0.000e+00   1.955e+01   2.764e+01   1.957e+01   1.954769e+01   8.495796e-01  3.411e-03  9.558e-05  2.695e+05
```

and the adapted mesh shown on the right of Figure 9.4.

All of the program execution takes place in the `../dump/` directory, and here you will find four files of interest. The first is `mesh0.tex` which is an example of the input files used to illustrate the displaced meshes in this document. It is a mixture of LaTeX 2_ε source which uses the TeXdraw graphics package. The other three files are `sigma11_0.m`, `sigma22_0.m` and `sigma12_0.m`. These are Matlab script files which may be executed within Matlab as they stand and will produce the surface plots of the stresses. These were also used earlier in this document to illustrate the numerical solutions.

Note that the directory `dump` is named deliberately. All files in it are disposable, in that they can regenerated, and are potentially large and so can eat up disk space. It is recommended that this directory be cleaned out once you have finished using the code. However, leave the (empty) `dump` directory in place since its existence is assumed and required by the script files `adapt` and `run`.

9.6 Some comments

There is clearly room for a great deal of improvement in the way the code is currently constructed and configured. A possible way forward here would be to provide a platform-independent Java GUI which could invoke the existing `native` C code through a `native` Java method of a larger object. This is however not research within the scope of the seed project and so we suggest it only as a possibility for the future.

Chapter 10

Suggestions for further work

10.1 Overview

In this short concluding chapter we give brief details of the code development work carried out between ourselves and Dr. A.R. Johnson during his visit to BICOM in March 1998. We then give a list of research areas that could follow naturally from the work detailed in this report.

10.2 A.R. Johnson's visit to BICOM

During the first week of March 1998 Dr. A.R. Johnson (ARL, VTC, NASA, Langley, VA, USA) visited BICOM with the aim of investigating how adaptivity could be built in to his existing work on internal variables (see for example [24, 25]). During this short time we and Dr. Johnson were able to generate a Fortran `FUNCTION` that was easily added on to one of his existing finite element codes, in this case for solving viscoelastic beam problems. This `FUNCTION` adaptively and automatically solves the internal variable equation over each of the time steps taken by the larger finite element solver. Its modular nature means that it can also be easily transplanted to function with codes written to solve problems involving structures other than beams. In this section we will briefly outline the nature of the algorithm and the type of error control produced, and also give a "listing" of the `FUNCTION` to illustrate how this process can be easily encapsulated in a simple module.

In [24, Equation 20.38] Johnson and Tessler show how a finite element discretization of a quasistatic viscoelasticity problem can be written in the form,

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{F}_{\text{mech}} - \boldsymbol{F}_{\text{visc}} \qquad \text{at time } t. \tag{\textbf{10}.1}$$

Here: $\boldsymbol{u}$ is the vector of unknown nodal displacements; $\boldsymbol{K}$ is the finite element stiffness matrix; $\boldsymbol{F}_{\text{mech}}$ the vector of external loads; and, $\boldsymbol{F}_{\text{visc}}$ a vector of viscous forces containing the viscoelastic effects. The vector $\boldsymbol{F}_{\text{visc}}$ is obtained by assembling the local element level viscous forces via,

$$\boldsymbol{F}_{\text{visc}} = \sum {}^{*}\boldsymbol{k}\, {}^{*}\boldsymbol{u},$$

where $^*\boldsymbol{k}$ is an element-level viscous-stiffness matrix, and $^*\boldsymbol{u}$ a vector of unknown internal displacements defined at the element level.

As shown by Johnson and Tessler in [24, Equation 20.37] these internal displacements are given on each element by the solution of the evolution equation,

$$\frac{d\,^*\boldsymbol{u}}{dt} + \frac{^*\boldsymbol{u}}{\tau} = \frac{d\boldsymbol{u}}{dt}. \tag{10.2}$$

Thus, to find the displacement $\boldsymbol{u}(t)$ at the given time t one has only to solve (**10**.1) to give,

$$\boldsymbol{u}(t) = \boldsymbol{K}^{-1}\Big(\boldsymbol{F}_{\text{mech}} - \boldsymbol{F}_{\text{visc}}\Big).$$

It is natural to assume that $\boldsymbol{F}_{\text{mech}}$ is known precisely and so the time discretization error in this equation is effectively present in the term $\boldsymbol{F}_{\text{visc}}$. It is therefore important to have an adaptive solution algorithm for the evolution equation (**10**.2) so that the viscous loads can be built accurately and cheaply. It was the derivation of such an algorithm, and its realization and modular implementation as a Fortran `FUNCTION`, that formed the focus of the collaborative work carried out at BICOM during Dr. Johnson's visit.

We give brief details of this work below, and we finish be suggesting ways in which this work could be extended.

The internal variable equation can be considered as a collection of scalar ODE problems, each having the form: find u such that,

$$\frac{du}{dt} + \frac{u}{\tau} = \frac{g}{\tau} \qquad \text{in } [0,T], \qquad \text{with } u(0) = u_0, \tag{10.3}$$

and where $T > 0$, u_0, g and $\tau > 0$ are given data. Here, of course, $[0,T]$ represents a single time interval within the time stepping loop of the overall finite element solver, and g is derived from the global displacement vector $\boldsymbol{u}$. Below we set,

$$\frac{\boldsymbol{g}}{\tau} = \frac{d\boldsymbol{u}}{dt},$$

although we could also work in terms of reversed internal variables with essentially no change to the adaptive algorithm.

Our approach to deriving a numerical scheme is based on discretizing the single time interval $[0,T]$ into subintervals in the usual way,

$$0 = t_0 < t_1 < \cdots < t_q < \cdots < t_N = T,$$

and then defining the time steps $k_q = t_q - t_{q-1}$, and subintervals $\mathcal{J}_q := (t_{q-1}, t_q)$. Before proceeding we note that it is possible to "solve" (**10**.3) explicitly by introducing an integrating factor, but one would then need to evaluate the resulting integral with "sufficient" accuracy. The approach described below could thus also be described as an adaptive quadrature algorithm, and in particular can still be applied with essentially no changes in the nonlinear case where $\tau = \tau(\boldsymbol{u})$.

Our discretization of the internal variable equation (**10**.3) is based on the finite element method. We introduce a piecewise constant (i.e. constant on each $\mathcal{J}_q$) approximation U to u and, in particular, denote by U_q the constant approximation to u during times $t \in \mathcal{J}_q$. Then, the Galerkin finite element approximation of (**10**.3) can be written as,

$$(1 + k_q/\tau)U_q = U_{q-1} + \int_{t_{q-1}}^{t_q} \frac{g}{\tau}\,dt \qquad \text{with } U_0 = u_0.$$

One obtains this time stepping scheme by taking the scalar product of (**10**.3) with a piecewise constant test function, replacing u with U (interpreting the derivative in a distributional sense) and then applying standard finite element methodology.

Using duality arguments we then arrive at the *a posteriori* error estimate,

$$|u(t_N) - U_N| \leq (1 - e^{-t_N/\tau}) \max_{1 \leq q \leq N} \left\{ \frac{k_q}{\tau} \|g - U\|_{L_\infty(\mathcal{J}_q)} + |U_q - U_{q-1}| \right\}.$$

Note that the quantities on the right are completely determined by the given data and the computed solution, and so this bound can form the basis of an adaptive algorithm. We describe this more fully below.

The drawback with the approach just taken is that the piecewise constant approximation to u is not particularly accurate. To address this we also developed and implemented a continuous piecewise linear finite element approximation to (**10**.3), where we retained the piecewise constant test function. In this case the time stepping scheme reduces to,

$$(1 + k_q/2\tau)U_q = (1 - k_q/2\tau)U_{q-1} + \int_{t_{q-1}}^{t_q} \frac{g}{\tau}\,dt.$$

For this scheme we derived the *a posteriori* error estimate,

$$|u(t_N) - U(t_N)| \leq (1 - e^{-t_N/\tau}) \max_{1 \leq q \leq N} \left\{ k_q \|r\|_{L_\infty(\mathcal{J}_q)} \right\},$$

where,

$$r := \frac{g}{\tau} - U_t - \frac{U}{\tau}$$

is the **residual** and is computable.

We use this *a posteriori* error estimate to generate an adaptive time stepping scheme in the following way. Suppose we want to compute a numerical solution for which,

$$|u(t_N) - U(t_N)| \leq \mathsf{TOL}$$

is guaranteed, where $\mathsf{TOL} > 0$ is a user-specified tolerance level, then it is sufficient to ensure that,

$$(1 - e^{-t_N/\tau}) \max_{1 \leq q \leq N} \left\{ k_q \|r\|_{L_\infty(\mathcal{J}_q)} \right\} = \mathsf{TOL}.$$

This in turn is also guaranteed if we ensure that,

$$k_q \|r\|_{L_\infty(\mathcal{J}_q)} = \mathsf{TOL}(1 - e^{-t_N/\tau})^{-1}$$

for every q. Rearranging this leads to a simple rule for selecting the time steps in an iterative, or **adaptive**, manner:

$$k_q^{\mathrm{new}} = \frac{\mathsf{TOL}(1 - e^{-t_N/\tau})^{-1}}{\|r\|_{L_\infty(\mathcal{J}_q)}}.$$

This adaptive solver for (**10**.3) can be encapsulated as a Fortran `FUNCTION` as listed below.

```
C*******************************************************
c **********              C O N T I N U O U S             **********
c ********** P I E C E W I S E   L I N E A R   A P P R O X **********
c --------------     MUST be declared as REAL*8      --------------------
c usage:
c  ustar = ivadapt1(du/dt, ic, tau_n, t_{m-1}, t_m, TOL, steps, mk)
c  minimum predicted time step may be returned in mk if mk is larger
c  on entry. Otherwise the value in mk is not altered
c  Note that "steps" return the number of steps taken in the
c  time interval (tlo, thi).
c
      real*8 function ivadapt1(rhs,ic,tau,tlo,thi,TOL,steps,mk)

      real*8 rhs, ic, tau, tlo, thi, TOL, t, mk
c     time step, new time step, previous and current iterates
      real*8 k, new_k, prevu, curru
      real*8 numer, denom           ! for time step control
      real*8 denom1, denom2         ! for time step control
      integer iter, steps

      steps = 0            ! record the number of steps taken
      prevu = ic           ! set initial condition
      k = thi - tlo        ! guess initial time step
      t = tlo              ! current time level is (t, t+k)
c     NOTE: next quantity can be supplied to save "exp" calculations
      numer = TOL / ( 1.0d0 - exp(-(thi-tlo)/tau) ) ! for adaptivity
      do iter = 1, 1000    ! no idea how many steps are needed!
        steps = steps + 1
c       get next solution
        curru = (1 - k/2.0d0/tau)*prevu + rhs*k
        curru = curru / (1 + k/2.0d0/tau)
c       test error condition by computing a new time step,
c       first find the max. value in the denominator
        denom1 = abs( rhs - prevu/tau - (curru-prevu)/k )
        denom2 = abs( rhs - curru/tau - (curru-prevu)/k )
        denom  = max(denom1, denom2) ! next line can throw an exception
        new_k = numer / denom        ! predict a new time step
                                     ! and act accordingly
        mk = min(mk, new_k)          ! remember minimum predicted time step
```

```
        if( new_k .GE. k ) then      ! advance to next time level
          t = t + k
          k = MIN( new_k, thi - t )
          prevu = curru
          if( t .GE. thi ) goto 1 ! break out if we have reached final time
        else                      ! recompute this time level
          k = new_k               ! reduce k and try again at this level
        end if
      enddo
      write(*,*) '........... ERROR in ivadapt1() .............'
 1    ivadapt1 = curru  ! return solution at thi
      end
c**************************************************************************
```

The initial condition is given by `ic` and the right hand side g/τ is supplied as the constant (on the time step) **`rhs`**. The interval $[0, T]$ is supplied through the argument list in the more general form $[t_{m-1}, t_m]$ and the `FUNCTION` returns the approximation $U(t_m)$ to $u(t_m)$.

This approach could easily be modified to develop higher order solvers for the internal variable equation, and then applied to more or less arbitrary viscoelasticity problems that can be formulated with such evolution equations. The approach is also highly suited to constitutively nonlinear problems modelled with a reduced time. In such a case $\tau = \tau(\boldsymbol{u})$, but this will introduce no essential complications into the adaptive scheme.

10.3 Additional research areas

This section is deliberately short and "bullet pointed". We want only to suggest areas that could be fruitfully investigated on the back of this work, and not pre-empt the fine detail of the topics.

- In a time dependent problem it is crucial for an adaptive algorithm to be able to selectively de-refine as well as refine the space mesh as features in the solution evolve or decay over time. As we explained earlier in Chapter 8, our current *a posteriori* error estimates (given in detail in [55, 56]) do not allow de-refinement without the inclusion of a complicated and expensive residual term. We believe the internal variable algorithm developed in this report will allow remove this expensive term for good and therefore allow mesh de-refinement with only a modest amount of additional expense. We plan to investigate this further at a later date.

- Temporal error control should be achievable by measuring the energy error in an appropriately weak norm. Results for a prototype problem have already been given in [53], and the extension of this work to the space-time quasistatic problem is underway in [56].

- Ultimately the goal must be solve physically realistic nonlinear problems. We feel that extending our results to constitutively nonlinear problems, characterized by a **reduced time**, would be fairly straightforward, and it may even be true that the stability estimates in [57] will hold without modification. However, nonlinearity arising from large deformations is a much more challenging problem and as yet we have no theoretical insights that will help in the construction of adaptive software.

- Quasistatic linear plate or shell problems could also be addressed in essentially the same way as described in this report for the "standard" two- and three-dimensional problem. In particular, the stability estimates in [57] should apply directly.

- The space-time finite element method as illustrated in this report can also be used to generate *a posteriori* error estimates and adaptive algorithms for dynamic viscoelasticity problems. We hope to begin to look at this subject soon.

Part IV

References

Bibliography

[1] J. T. Bendler, B. Noble, and M. A. Hussain. Solution of an equation for creep in solid polymers. In *Proc. Computers in Engineering*, number 3 in 1, pages 365—368, 1988.

[2] J. G. Blom and H. Brunner. The numerical solution of nonlinear Volterra integral equations of the second kind by collocation and iterated collocation methods. *SIAM J. Sci. Stat. Comput.*, 8:806—830, 1987.

[3] J. R. Cannon and Y. Lin. *A priori* L^2 error estimates for finite-element methods for nonlinear diffusion equations with memory. *SIAM J. Numer. Anal.*, 27:595—607, 1990.

[4] J. T. Chern. *Finite element modeling of viscoelastic materials on the theory of fractional calculus.* PhD thesis, Penn. State Uni., USA, 1993.

[5] D. S. Cohen and A. B. White Jr. Sharp fronts due to diffusion and stress at the glass transition in polymers. *J. Polymer Sci. B: Polymer Physics*, 27:1731—1747, 1989.

[6] D. S. Cohen and A. B. White Jr. Sharp fronts due to diffusion and viscoelastic relaxation in polymers. *SIAM J. Appl. Math.*, 51:472—483, 1991.

[7] D. S. Cohen, A. B. White Jr., and T. P. Witelski. Shock formation in a multidimensional viscoelastic diffusive system. *SIAM J. Appl. Math.*, 55:348—368, 1995.

[8] R. W. Cox. Shocks in a model for stress-driven diffusion. *SIAM J. Appl. Math.*, 50:1284—1299, 1990.

[9] R. W. Cox and D. S. Cohen. A mathematical model for stress driven diffusion in polymers. *J. Polymer Sci. B: Polymer Physics*, 27:589—602, 1989.

[10] C. M. Dafermos. An abstract Volterra equation with applications to linear viscoelasticity. *J. Diff. Eqns.*, 7:554—569, 1970.

[11] C. M. Dafermos and J. A. Nohel. Energy methods for nonlinear hyperbolic Volterra type equations. *Comm. Part. Diff. Eqns.*, 4:219—278, 1970.

[12] Kai Diethelm. An algorithm for the numerical solution of differential equations of fractional order. *Electronic Transactions on Numerical Analysis*, 5:1—6, 1997.

[13] J. Douglas and B. F. Jones. Numerical methods for integro-differential equations of parabolic and hyperbolic types. *Numer. Math.*, 4:96—102, 1962.

[14] C. J. Durning. Differential sorption in viscoelastic fluids. *J. Polymer Sci. B: Polymer Physics*, 23:1831—1855, 1985.

[15] David A. Edwards. Constant front speed in weakly diffusive non-Fickian systems. *SIAM J. Appl. Math.*, 55:1039—1058, 1995.

[16] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. Introduction to adaptive methods for differential equations. *Acta Numerica*, pages 105—158, 1995.

[17] Kenneth Eriksson and Claes Johnson. Adaptive finite element methods for parabolic problems. I: a linear model problem. *SIAM J. Numer. Anal.*, 28:43—77, 1991.

[18] J. D. Ferry. *Viscoelastic properties of polymers.* John Wiley and Sons Inc., 1970.

[19] E. Hairer, CH. Lubich, and M. Schlichte. Fast numerical solution of nonlinear Volterra convolution equations. *SIAM J. Sci. Stat. Comput.*, 6:532—541, 1985.

[20] C. K. Hayes and D. S. Cohen. The evolution of steep fronts in non-Fickian polymer-penetrant systems. *J. Polymer Sci. B: Polymer Physics*, 30:145—161, 1992.

[21] Melvin L. Heard. An abstract parabolic Volterra integrodifferential equation. *SIAM J. Math. Anal.*, 13:81—105, 1982.

[22] V. Janovsky, Simon Shaw, M. K. Warby, and J. R. Whiteman. Numerical methods for treating problems of viscoelastic isotropic solid deformation. *J. Comput. Appl. Math.*, 63:91—107, 1995.

[23] A. R. Johnson and C. J. Quigley. A viscohyperelastic Maxwell model for rubber viscoelasticity. *Rubber Chem. Technology*, 65:137—153, 1992.

[24] A. R. Johnson and A. Tessler. A viscoelastic high order beam finite element. In J. R. Whiteman, editor, *The Mathematics of Finite Elements and Applications.* MAFELAP *1996*, pages 333—345. Wiley, Chichester, 1997.

[25] A. R. Johnson, A. Tessler, and M. Dambach. Dynamics of thick viscoelastic beams. *Journal of Engineering Materials and Technology*, 119:273—278, 1997.

[26] Arthur R. Johnson and Ross G. Stacer. Rubber viscoelasticity using the physically constrained system's stretches as internal variables. *Rubber Chem. Technology*, 66:567—577, 1993.

[27] C. Johnson and P. Hansbo. Adaptive finite element methods in computational mechanics. *Comput. Methods Appl. Mech. Engrg.*, 101:143—181, 1992.

[28] H. M. Jones and S. McKee. Variable step size predictor-corrector schemes for second kind Volterra integral equations. *Math. Comp.*, 44:391—404, 1985.

[29] M. Křížek and P. Neittaanmäki. *Finite element approximation of variational problems and applications*, volume 50 of *Pitman Monographs and Surveys in Pure and Applied Mathematics.* Longman Scientific & Technical, 1990.

[30] F. J. Lockett. *Nonlinear viscoelastic solids.* Academic Press, 1972.

[31] J. C. López-Marcos. A difference scheme for a nonlinear partial integrodifferential equation. *SIAM J. Numer. Anal.*, 27:20—31, 1990.

[32] CH. Lubich, I. H. Sloan, and V. Thomée. Nonsmooth data error estimates for approximations of an evolution equation with a positive-type memory term. *Math. Comp.*, 65:1—17, 1996.

[33] Marcus J. Ludwig. *Finite element error estimation and adaptivity for problems of elasticity.* PhD thesis, Brunel University, England, Submitted September 1998. (See `http://www.brunel.ac.uk/~icsrbicm`).

[34] R. C. MacCamy. A model for one-dimensional, nonlinear viscoelasticity. *Q. Appl. Math.*, 35:21—33, 1977.

[35] W. Mclean and V. Thomée. Numerical solution of an evolution equation with a positive-type memory term. *J. Austral. Math. Soc.*, 35:23—70, 1993.

[36] R. K. Miller. An integrodifferential equation for rigid heat conductors with memory. *J. Math. Anal. Appl.*, 66:313—332, 1978.

[37] J. A. Nohel. A nonlinear hyperbolic Volterra equation. In *Volterra equations*, volume 737 of *Lecture Notes in Mathematics*, pages 220—235. Spinger-Verlag, 1979.

[38] J. W. Nunziato. On heat conduction in materials with memory. *Quart. Appl. Maths.*, 29:187—204, 1971.

[39] A. K. Pani, V. Thomée, and L. B. Wahlbin. Numerical methods for hyperbolic and parabolic integro-differential equations. *J. Integral Equations Appl.*, 4:533—584, 1992.

[40] Amiya K. Pani and Todd E. Peterson. Finite element methods with numerical quadrature for parabolic integrodifferential equations. *SIAM J. Numer. Anal.*, 33:1084—1105, 1996.

[41] J. M. Sanz-Serna. A numerical method for a partial integro-differential equation. *SIAM J. Numer. Anal.*, 25:319—327, 1988.

[42] Simon Shaw, M. K. Warby, and J. R. Whiteman. An error bound via the Ritz-Volterra projection for a fully discrete approximation to a hyperbolic integrodifferential equation. Technical report, 94/3, BICOM, Brunel University, Uxbridge, U.K., 1994. (`http://www.brunel.ac.uk/~icsrbicm`).

[43] Simon Shaw, M. K. Warby, and J. R. Whiteman. A comparison of hereditary integral and internal variable approaches to numerical linear solid viscoelasticity. In *Proceedings of the XIII Polish Conference on Computer Methods in Mechanics*, 1997. Poznan, May 1997 (BICOM Tech. Rep. 97/2, see `http://www.brunel.ac.uk/~icsrbicm`).

[44] Simon Shaw, M. K. Warby, and J. R. Whiteman. Error estimates with sharp constants for a fading memory Volterra problem in linear solid viscoelasticity. *SIAM J. Numer. Anal.*, 34:1237—1254, 1997. (See also, `http://www.brunel.ac.uk/~icsrbicm`).

[45] Simon Shaw, M. K. Warby, J. R. Whiteman, C. Dawson, and M. F. Wheeler. Numerical techniques for the treatment of quasistatic viscoelastic stress problems in linear isotropic solids. *Comput. Methods Appl. Mech. Engrg.*, 118:211—237, 1994.

[46] Simon Shaw and J. R. Whiteman. Backward Euler and Crank-Nicolson finite element variants with rational adaptivity and *a posteriori* error estimates for an integrodifferential equation. Submitted to Math. Comp. (see `http://www.brunel.ac.uk/~icsrbicm`), 1996.

[47] Simon Shaw and J. R. Whiteman. Discontinuous Galerkin method with *a posteriori* $L_p(0, t_i)$ error estimate for second-kind Volterra problems. *Numer. Math.*, 74:361—383, 1996.

[48] Simon Shaw and J. R. Whiteman. Towards adaptive finite element schemes for partial differential Volterra equation solvers. *Advances in Computational Mathematics*, 6:309—323, 1996.

[49] Simon Shaw and J. R. Whiteman. Applications and numerical analysis of partial differential Volterra equations: a brief survey. *Comput. Methods Appl. Mech. Engrg.*, 150:397—409, 1997.

[50] Simon Shaw and J. R. Whiteman. Some partial differential Volterra equation problems arising in viscoelasticity. Technical report, BICOM, Brunel University, Uxbridge, England, 1997. TR97/9 (Proc. Equadiff, August 1997).

[51] Simon Shaw and J. R. Whiteman. Space-time finite element method with *a posteriori* Galerkin energy-error estimate for linear quasistatic viscoelasticity problems. BICOM Technical Report 97/7, see `http://www.brunel.ac.uk/~icsrbicm`, 1997.

[52] Simon Shaw and J. R. Whiteman. Towards robust adaptive finite element methods for partial differential Volterra equation problems arising in viscoelasticity theory. In J. R. Whiteman, editor, *The Mathematics of Finite Elements and Applications.* MAFELAP *1996*, pages 55—80. Wiley, Chichester, 1997.

[53] Simon Shaw and J. R. Whiteman. Negative norm error control for second-kind convolution Volterra equations. Submitted to *Numer. Math*; BICOM Tech. Rep. 98/6, see `http://www.brunel.ac.uk/~icsrbicm`, 1998.

[54] Simon Shaw and J. R. Whiteman. Numerical solution of linear quasistatic hereditary viscoelasticity problems I: *a priori* estimates. BICOM Tech. Rep. 98/2, see `http://www.brunel.ac.uk/~icsrbicm`, 1998.

[55] Simon Shaw and J. R. Whiteman. Numerical solution of linear quasistatic hereditary viscoelasticity problems II: *a posteriori* estimates. BICOM Tech. Rep. 98/3, see `http://www.brunel.ac.uk/~icsrbicm`, 1998.

[56] Simon Shaw and J. R. Whiteman. Numerical solution of linear quasistatic hereditary viscoelasticity problems III: *a posteriori* estimates in a weak norm. BICOM Technical Report 98/4,
see `http://www.brunel.ac.uk/~icsrbicm`, 1998.

[57] Simon Shaw and J. R. Whiteman. Optimal long-time $L_p(0, T)$ data stability and semidiscrete error estimates for the Volterra formulation of the linear quasistatic viscoelasticity problem. Submitted to *Numer. Math*; BICOM Tech. Rep. 98/7 see: `http://www.brunel.ac.uk/~icsrbicm.`, 1998.

[58] I. H. Sloan and V. Thomée. Time discretization of an integro-differential equation of parabolic type. *SIAM J. Numer. Anal.*, 23:1052—1061, 1986.

[59] V. Thomée and L. B. Wahlbin. Long-time numerical solution of a parabolic equation with memory. *Math. Comp.*, 62:477—496, 1994.

[60] E. G. Yanik and G. Fairweather. Finite element methods for parabolic and hyperbolic partial integro-differential equations. *Nonlinear Analysis, Theory, Methods & Applications*, 12:785—809, 1988.