Contents

Simulation of elasticity, biomechanics, and virtual surgery

Joseph M. Teran

IAS/Park City Mathematics Series
Volume XX, XXXX

Simulation of elasticity, biomechanics, and virtual surgery

Joseph M. Teran

UCLA Mathematics Department, Box 951555, Los Angeles, CA 90095-1555
E-mail address: jteran@math.ucla.edu

FIGURE 1. Surgical simulation will ideally be used to provide a virtual environment of prototyping procedures as well as for research and development of novel procedures. The images show a subject specific simulated cleft lip and palate repair. Elastic simulation of soft tissues is the primary algorithmic challenge in providing such technologies.

Introduction

This short lecture series will cover the simulation of elastic materials characteristic of biological soft tissues. The target application is virtual surgery. This relatively new field places some unique constraints on the types of algorithms needed for simulation. First and foremost, the simulation must go very fast, nearly unprecedentedly so in fact. We must run traditional scientific computing applications in real-time to provide the functionality needed for a surgical simulator. Specifically, we must update the state of the simulation every thirtieth of a second in order to provide an interactive environment. This is highly non-trivial as such steps usually require the solution of large linear systems of equations, a task that can be notoriously time consuming. Furthermore, these simulations must be abnormally robust to user input. Many of us have played video games. The presence of a bug or some source of unexpected behavior is unacceptable as it degrades the user experience and can render the environment non-interactive. Satisfying these two constraints as well as a tutorial in basic simulation of elasticity will be the primary focus of this lecture series.

Real-time computing

Real-time simulation refers to the ability to evolve the approximation to an initial boundary value problem in less than a thirtieth of a second. In other words, less than the time it would take to observe the solution on the screen. Traditionally, computation times required for a time step in such problems have been on the order of a few minutes or even a few hours, far short of the thirtieth of a second constraint. However, such performance, should it be allowed, would provide a controllable virtual environment where a user would have the ability to change the forcing and boundary conditions of a simulation in response to real-time observation of the solution. The application of such functionality for solid and fluid mechanics problems is engendering many new applications. For example, imagine an interactive virtual environment where the user interacts with a finite element simulation of biomechanical soft tissues. The governing equations dictate how the tissues respond to external influence from the users. This predictive ability would allow the user to push, pull or even cut/excise portions of the tissue in real time with full confidence that the real life counterpart would behave similarly. This ability could potentially revolutionize the process of training surgical residents and medics by providing a cost effective and scientific alternative to expensive cadaver based training. Imagine the ability to know the outcome of a surgical procedure before it is performed. Reconstructive surgery for severe trauma is unpredictable by the nature of the injuries. The surgeon must design the treatment procedure on a case-by case-basis. With the aid of a predictive simulator the surgeon could

perform hypothetical surgical repairs in advance to determine the most likely successful approach. This would significantly reduce complication rates and lead to drastically improved quality of life. See [**3**] for further discussion of the potential benefits of surgical simulation.

LECTURE 1

Introduction to Continuum Mechanics and Elasticity

Kinematics

I will go through the basic continuum mechanics for describing the *initial boundary value problems* (IBVPs) for deformable objects as quickly as possible with an emphasis on only those details necessary for implementing a basic *finite element method* (FEM) type simulation (see the text [**1**] for more details). We will quantify the deformation of the objects of interest in terms of the mapping $\boldsymbol{\phi}$ between an initial (or material) $\mathbf{B}_0 \in \mathbb{R}^2$ and current (or deformed) $\mathbf{B}_t \in \mathbb{R}^2$ configuration (we'll just be covering 2D for simplicity). Let's introduce the convention that points in $\mathbf{B}_0$ we label as $\mathbf{X}$ and points in $\mathbf{B}_t$ we label as $\mathbf{x}$ (see Figure 1). I.e. $\mathbf{B}_0 = \{\mathbf{X}\}$ and $\mathbf{B}_t = \{\mathbf{x}\}$ so

$$\boldsymbol{\phi}(\cdot, t)\colon \mathbf{B}_0 \to \mathbf{B}_t \quad \text{and} \quad \boldsymbol{\phi}(\mathbf{X}, t) = \mathbf{x}$$

It is also useful to talk about

$$\mathbf{u}(\cdot, t)\colon \mathbf{B}_0 \to \mathbb{R}^2 \quad \text{where} \quad \mathbf{u}(\mathbf{X}, t) = \boldsymbol{\phi}(\mathbf{X}, t) - \mathbf{X}.$$

We'll refer to this function $\mathbf{u}$ as the displacement mapping. We'll also introduce a little more notation here for the derivatives of these mappings. The deformation gradient refers to $\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}$ and is often denoted with $\mathbf{F}$.

$$\mathbf{F}(\cdot, t)\colon \mathbf{B}_0 \to \mathbb{R}^{2\times 2} \quad \text{and} \quad \mathbf{F} = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}.$$

Index notation for derivatives will be helpful for compactness of expressions. E.g. $F_{ij} = \frac{\partial \phi_i}{\partial X_j} = \phi_{i,j}$ and $\frac{\partial u_i}{\partial X_j} = u_{i,j}$.

Basic Balance Laws

Again, I will only present the essential details for getting started with elastodynamics. Our governing equations of motion arise from the basic principles of continuum mechanics, ultimately $F = ma$ applied over our continuum of material $\mathbf{B}_0$. This reads as:

$$\rho_0(\mathbf{X}) \frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}}.$$

Here, ρ_0 is the mass density of the material, $\mathbf{f}^{\text{ext}}$ is an externally applied force, and $\mathbf{P}$ is the first Piola-Kirchoff Stress which describes the forces of interaction

FIGURE 1. We will be solving for the mapping between the current configuration and a rest configuration. Stresses will arise via elasticity to resist changes in shape induced by this motion.

in the material (specifically, $\mathbf{P}$ relates the normal of an infinitesimal surface in the material to the forces applied by one side of the material to the other; see Figure 1). We will discuss the form of this stress in the coming sections. For now, note that it is a matrix valued function over the material:

$$\mathbf{P}\colon \mathbf{B}_0 \times [0,T] \to \mathbb{R}^{2\times 2}.$$

For elasticity, we will be relating this quantity to the deformation gradient $\mathbf{F}$. It will usually make more sense for us to consider $\mathbf{P}$ as a function of $\mathbf{F}$, but we will discuss this in the constitutive modeling section. This equation is hyperbolic and roughly like the second order linear wave equation (although the specific nature of $\mathbf{P}$ for elasticity will give rise to non-linear equations). It will be useful numerically to consider both the velocities of points of material and the displacements as unknowns. We will use $\mathbf{v}(\mathbf{X},t) = \frac{\partial \mathbf{u}}{\partial t}(\mathbf{X},t)$ to denote the velocity of material point $\mathbf{X}$. With this convention, we can rewrite our problem with the equivalent system:

$$\begin{pmatrix} \frac{\partial \mathbf{u}}{\partial t} \\ \rho_0 \frac{\partial \mathbf{v}}{\partial t} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \end{pmatrix}.$$

Elasticity and Constitutive Modeling

The equations are not yet closed because we still need to determine a relationship between the stress and the state of the material. For elastic materials, this is often referred to as the constitutive or material law. Here, we will focus on hyperelastic materials. The constitutive law for such materials arises from a so-called strain energy density function Ψ. Specifically, we define the stress as the derivative of this energy with respect to the deformation gradient $\mathbf{F}$:

$$\Psi\colon \mathbb{R}^{2\times 2} \to \mathbb{R} \quad \text{and} \quad \mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}.$$

In index notation this is $P_{ij} = \Psi_{,ij}$. In general, the indices following the comma refer to partial differentiation.

The most simplistic model for elasticity is isotropic linear elaticity. In this case, we have

$$\epsilon_{ij} = \frac{1}{2}\left(F_{ij} + F_{ji}\right) - \delta_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i}\right) \quad \text{and} \quad \Psi\left(\mathbf{F}\right) = \mu \epsilon_{ij}\epsilon_{ij} + \frac{\lambda}{2}\epsilon_{kk}^2.$$

We are using the convention that repeated indices are summed here and δ_{ij} is the identity matrix. Differentiating this expression with respect to $\mathbf{F}$ gives us the relationship between the stress and the state of the displaced material:

$$P_{ij} = \Psi_{,ij} = 2\mu\epsilon_{ij} + \lambda\epsilon_{kk}\delta_{ij} \quad \text{or} \quad \mathbf{P} = 2\mu\boldsymbol{\epsilon} + \lambda\,\mathrm{tr}(\boldsymbol{\epsilon})\mathbf{I}.$$

This relationship between the stress and the displacement mapping is enough to close our governing system of equations. I will not give too much motivation for these equations, but note that $\boldsymbol{\epsilon}$ is zero when $\mathbf{F}$ is skew-symmetric. Also, note that if the displacement is spatially constant, then $\frac{\partial \mathbf{u}}{\partial \mathbf{X}} = -\mathbf{I}$ and hence $\boldsymbol{\epsilon} = \mathbf{P} = \mathbf{0}$. $\boldsymbol{\epsilon}$ is a measure of the strain (or change of shape) in the material. Ideally, we'd like a measure that was invariant under rigid motions $\boldsymbol{\phi}$. Unfortunately, the motions just described are only approximately rigid. The approximation is valid if we assume that the deformation is very small. In fact, we can only say that linear elasticity is valid if the deformation is very small. Otherwise, it is not a very good model. That is, it does not accurately describe any elastic materials observed in nature unless the strain is very small. A more general expression of a rigid

FIGURE 2. Constitutive models for muscle, tendon, ligament and many other tissues must be anisotropic to accurately reflect the natural fibrous structure of these materials. In such cases, we must represent a fiber field defined over the material configuration of the material as shown above in a rectis femoris muscle.

FIGURE 3. Simulation of elasticity for the musculoskeletal system.

body motion is that the deformation gradient is orthogonal. I.e., that $\mathbf{F}^T\mathbf{F} = \mathbf{I}$. In fact, more appropriate strain measures for large deformation elasticity are the right and left Cauchy-Green strain tensors: $\mathbf{G}_R = \mathbf{F}^T\mathbf{F} - \mathbf{I}$ and $\mathbf{G}_L = \mathbf{F}\mathbf{F}^T - \mathbf{I}$, respectfully. Note that this measure is quadratic in the deformation. This will lead to nonlinear governing equations and complicate considerably the process of running our simulation. In other words, if we want to simulate large tissues that undergo significant deformation (as we would need in virtual surgery), we'll be stuck with nonlinear governing equations.

Arguably, the most simplistic model for large deformation elasticity is the *Neo-Hookean* model. The hyperelastic strain energy density for such materials is given by the following equation

$$\Psi(\mathbf{F}) = \frac{\mu}{2}(F_{ij}F_{ji} - 2) - \mu\log(J) + \frac{\lambda}{2}\log(J)^2$$

$$\mathbf{P}(\mathbf{F}) = \mu\mathbf{F} + (\lambda\log(J) - \mu)\mathbf{F}^{-T}$$

where J is the determinant of the deformation gradient

$$J = \det(\mathbf{F}).$$

The log term in this equation is there to resist changes in volume. The λ and μ terms can be set as in linear elasticity. The most intuitive way to set these parameters is from the Young's modulus K and Poisson's ratio ν. This relationship is

$$\lambda = \frac{K\nu}{(1+\nu)(1-2\nu)} \quad \text{and} \quad \mu = \frac{K}{2(1+\nu)}.$$

The Young's modulus is a measure of the stiffness of the material and should be larger than zero (typically around 1000 for our exercises). The Poisson's ratio is a measure of the incompressibility of the material and should be between 0 and $\frac{1}{2}$ (with $\nu = \frac{1}{2}$ being the limit of an incompressible material; note that λ goes to ∞ in this case).

The models described so far are both isotropic. That is, the elastic response to deformation is the same regardless of which direction the material is stretched in. This is not very accurate for many fibrous tissues in the anatomy. In these cases, we need an auxiliary function $\mathbf{g}\colon \mathbf{B}_0 \to \mathbb{R}^2$ that describes the local fiber direction in the material configuration of the tissue (see Figure 2). The specialized response in the fiber direction can be incorporated by changing the original isotropic strain energy density to include a term based on the stretching in the fiber direction: $\mathbf{g}^T\mathbf{F}^T\mathbf{F}\mathbf{g}$. That is, the modification looks like

$$\Psi(\mathbf{F}) = \Psi_{\text{iso}}(\mathbf{F}) + \Psi_{\text{fiber}}\left(\mathbf{g}^T\mathbf{F}^T\mathbf{F}\mathbf{g}\right).$$

Equilibrium and Weak Form

We can get an idea for how to treat the spatial discretization of the partial differential equation by first considering elastic equilibrium (or quasistatic) problems where the inertial terms in the governing equations are negligible (see [**4**] for more details). In this case we can solve directly for the displacement (i.e., we don't need to store the velocities as well). Ignoring inertia, our governing equations are then

$$-\nabla^{\mathbf{X}} \cdot \mathbf{P} = \mathbf{f}^{\text{ext}} \quad \text{or} \quad -P_{ij,j} = f_i^{\text{ext}}.$$

This can be read as the row wise divergences of the matrix $\mathbf{P}$ balance the different components of the applied forces. We'll mainly be considering *finite element method* (FEM) type discretizations in these notes, so we'll look at the weak form here to get started with that. Taking a class of test functions $\mathbf{w}\colon \mathbf{B}_0 \to \mathbb{R}^2$, we take the dot product with both sides of the governing equations and integrate over $\mathbf{B}_0$ to get

$$\begin{aligned} -\int_{\mathbf{B}_0} w_i P_{ij,j} d\mathbf{X} &= \int_{\mathbf{B}_0} w_i f_i^{\text{ext}} d\mathbf{X} \\ -\int_{\mathbf{B}_0} \left(w_i P_{ij}\right)_{,j} - w_{i,j} P_{ij} d\mathbf{X} &= \int_{\mathbf{B}_0} w_i f_i^{\text{ext}} d\mathbf{X} \\ \int_{\mathbf{B}_0} w_{i,j} P_{ij} d\mathbf{X} &= \int_{\partial \mathbf{B}_0} w_i P_{ij} n_j d\mathbf{S}\left(\mathbf{X}\right) + \int_{\mathbf{B}_0} w_i f_i^{\text{ext}} d\mathbf{X}. \end{aligned}$$

We'd then just assume that the bottom equations hold for all suitable $\mathbf{w}$ for the weak form. Furthermore, the quantity $P_{ij}n_j$ would be supplied as a boundary condition (this would be the external loading on the material). A FEM discretization would then be done by setting up a space of interpolating functions over a mesh and considering the weak form over this finite dimensional space.

1D Elasticity

To provide an introductory example, I will cover the case of 1D elasticity (I will assume some basic knowledge of FEM). Here, we'll first assume the material is linearly elastic. In that case,

$$P(X) = (2\mu - \lambda) \frac{\partial u}{\partial X}$$

and the equilibrium equation is just Poisson's equation:

$$-(2\mu - \lambda) \frac{\partial^2 u}{\partial X^2} = f^{\text{ext}}.$$

Thus, we can just use a continuous piecewise linear interpolation space of a uniform discretization of our material (which I will assume for simplicity is just some interval (a, b)). In this case, we will ultimately just be solving a Poisson equation for the unknown deformation field. This is only the case in 1D. If we assume that each node in the discrete domain has an associated interpolating function N_i, then the FEM discretized linear system is

$$A_{ij} u_j = F_i^{\text{ext}} + g_i \quad \text{where} \quad A_{ij} = (2\mu - \lambda) \int_a^b \frac{\partial N_i}{\partial X} \frac{\partial N_j}{\partial X} dX,$$

$$F_i^{\text{ext}} = \int_a^b f^{\text{ext}} N_i dX, \quad g_i = N_i(b)P(b) - N_i(a)P(a)$$

FIGURE 4. The image at the left shows an acceptable solution for the deformation ϕ (namely, $\phi(X) = X^2$). The solution at the right shows one that would produce inverted material (namely, $\phi(X) = \sin(2\pi X)$).

(where we assume that $P(a)$ and $P(b)$ were already given). Now, let's consider the case of Neo-Hookean elasticity. In this case, our discretization of the weak form will give rise to a nonlinear system of equations for our displacements u_i:

$$q_i(\vec{u}) = \int_a^b \frac{\partial N_i}{\partial X} P(F(u(X)))dX - F_i^{\text{ext}} + g_i = 0$$

where $u(X) = u_i N_i(X)$ (again, where repeated indices imply sumation). For Neo-Hookean materials in 1D, we have

$$P(F(u(X))) = \mu\left(\frac{\partial u}{\partial X} + 1\right) + \left(\lambda \log\left(\frac{\partial u}{\partial X} + 1\right) - \mu\right)\frac{1}{\frac{\partial u}{\partial X} + 1}.$$

We can think of $\vec{q}\colon \mathbb{R}^n \to \mathbb{R}^n$ as a nonlinear function that we need to find a zero of. We'll use Newton's method to iteratively improve our approximate solution (here, I am assuming $\vec{u}^k \in \mathbb{R}^n$ is our current approximation to the vector of discrete displacements) as:

$$\frac{\partial q_i}{\partial u_j}\left(\vec{u}^k\right)\Delta u_j + q_i\left(\vec{u}^k\right) = 0,$$
$$u_i^{k+1} \leftarrow u_i^k + \Delta u_i.$$

The linearization of the discrete equilibrium equations can be a little tricky, so it will be good to first consider it in 1D. The general idea is to linearize the stress with respect to the deformation gradient, then linearize the deformation gradient with respect to discrete degrees of freedom.

$$\frac{\partial q_i}{\partial u_j}(\vec{u}) = \int_a^b \frac{\partial N_i}{\partial X}\frac{\partial P}{\partial F}(F(\vec{u}))\frac{\partial F}{\partial u_j}dX = \int_a^b \frac{\partial N_i}{\partial X}\frac{\partial P}{\partial F}(F(\vec{u}))\frac{\partial N_j}{\partial X}dX$$

since $u(X) = u_i N_i(X)$ and $F(X) = \frac{\partial u}{\partial X}(X) + 1 = u_i \frac{\partial N_i}{\partial X}(X) + 1$. Note that the linearized system is just a spatially varying Poisson equation. However, it should be noted that for such equations we must typically require the coefficients remain positive throughout the domain. This will not always happen and can lead to difficulties with robustness. We will discuss this issue in the next section.

Inversion

The deformation mapping $\boldsymbol{\phi}$ must be a bijection (one-to-one and onto) if we are to strictly apply the principles of continuum mechanics. This precludes many solutions you might otherwise expect to see. For example, for the 1D case, the solution must never have a derivative less than or equal to zero. I.e. many functions we are used to solving for (see Figure 4) are not, strictly speaking, allowable. Unfortunately, this constraint will not always be naturally achieved by our discretization. For some models (e.g. linear elasticity) it is okay that the discrete solution does not satisfy the inversion constraint. For others like Neo-Hookean, we would end up taking the natural log of a negative number. This log term is there to prevent excessive compression. However, it has the consequence that our equations become

FIGURE 5. These images show the the algorithm of [**2**] in action. This algorithm is based on removing the bijection constraint from continuum mechanics in the discrete setting to provide robustness to large deformation.

FIGURE 6. The plots at the left are of the hyperelastic energy density of the 1D Neo-Hookean model. The plots at the right are of the 1D Neo-Hookean stresses. The blue curves are the unmodified Neo-Hookean model, the green curves result from replacing the log with it's cubic Taylor expansion around $J = 1$. These are preferable for the sake of robustness.

incredibly stiff under compression. Such behavior can be very problematic numerically and must be avoided in a surgical simulator. A general approach to improving robustness to discretely inverted (i.e., non-bijective) deformation mappings was developed in [**2**]. I will not go through all of the details for the sake of time. However, I will show how we can apply the principles developed in this work to our simple Neo-Hookean model.

The problematic term in the Neo-Hookean model is the natural log. If we replace the log with another function r, we can improve the robustness of our numerical simulation to inversion. This function should look very nearly like the natural log around $J = 1$ to prevent loss of accuracy, but should not have a singularity at $J = 0$. We can do this by taking r to be the cubic Taylor expansion of the log around $J = 1$ (see Figure 6). With this choice, we have the following expressions for the Neo-Hookean hyperelastic strain energy density and first Piola-Kirchoff stresses:

$$\Psi(\mathbf{F}) = \frac{\mu}{2}(F_{ij}F_{ji} - 2) - \mu r(J) + \frac{\lambda}{2}r(J)^2$$

$$\mathbf{P}(\mathbf{F}) = \mu\mathbf{F} + (\lambda r(J) - \mu)\, r'(J) \begin{pmatrix} F_{22} & -F_{12} \\ -F_{21} & F_{11} \end{pmatrix}.$$

In Figure 6, we see the effect of this modification in 1D. This new model is well-defined through inversion, but accurately reflects the original model when very near the undeformed configuration. We can make further adjustments to prevent problematic values of $\frac{\partial P}{\partial F}$ (e.g. we wanted this to be positive for our 1D example). The treatment of these terms is addressed in [**4**].

Time Stepping

In general, our simulator will run the following loop:

(1.6)

for $n = 0 : N_{\text{final}}$
- get user boundary condition input
- $t \leftarrow t + \Delta t$
- solve nonlinear system for displacements and velocities
- render solution to screen

end

The key factor is that the nonlinear system must be solved in less than one thirtieth of a second to maintain an interactive simulation rate. An example of a discretization of this system that allows for inertia is the following. Let $\vec{u}^m \in \mathbb{R}^n$ represent the spatially discrete displacements at time m and let $\vec{v}^m \in \mathbb{R}^n$ represent the spatially discrete velocities. We can implement backward Euler by saying

$$\vec{v}^{m+1} = \frac{1}{\Delta t}\left(\vec{u}^{m+1} - \vec{u}^m\right)$$

and

$$\rho_0 \frac{1}{\Delta t}\left(\frac{1}{\Delta t}\left(\vec{u}^{m+1} - \vec{u}^m\right) - \vec{v}^m\right) = \nabla^{\mathbf{X}} \cdot \mathbf{P}\left(\mathbf{F}\left(\vec{u}^{m+1}\right)\right) + \mathbf{f}^{\text{ext}}$$

where we solve this final system by following a similar FEM procedure to the quasistatic case for $\vec{u}^{m+1}$. Specifically, we'd get

$$\begin{aligned} q_i^a\left(\vec{u}^{m+1}\right) = {} & M_{ai;bj} u_{bj}^{m+1} \\ & + \Delta t^2 \int_{\mathbf{B}_0} N^a_{,j} P_{ij} d\mathbf{X} \\ & - M_{ai;bj}\left(u_{bj}^{m+1} + \Delta t v_{bj}^m\right) \\ & - \int_{\partial \mathbf{B}_0} N^a P_{ij} n_j d\mathbf{S}(\mathbf{X}) \\ & - \int_{\mathbf{B}_0} N^a f_i^{\text{ext}} d\mathbf{X} \end{aligned}$$

and $M_{ai;bj} = \int_{\mathbf{B}_0} N^a \rho_0 N^b \delta_{ij} d\mathbf{X}$.

LECTURE 2

Numerical Solutions of the Equations of Elasticity

This lecture provides details and exercises pertaining to the implementation of numerical solvers for the equations of elasticity. We begin with a review of the Finite Element Method (FEM) as it applies to Poisson's equation. We will then apply this framework to numerically solve the equations of elasicity, first in dimension $d = 1$ with quasistatic evolution, then in dimension $d = 2$ with inertial effects included.

1. Numerical Solution of Poisson's Equation via the Finite Element Method

Let us consider numerical solutions of (variable coefficient) Poisson's equation:

$$-\nabla \cdot (\beta \nabla u) = f \quad \in \Omega \tag{2.1a}$$

$$u = p \quad \in \partial\Omega_d \tag{2.1b}$$

$$\beta \nabla u \cdot \hat{n} = q \quad \in \partial\Omega_n \tag{2.1c}$$

Here, $\Omega \subset \mathbb{R}^d$ is open; $\partial\Omega_d$ and $\partial\Omega_n$ partition the boundary of Ω and define where *Dirichlet* (d) and *Neumann* (n) boundary conditions are applied, respectively; $\hat{n}$ is the outward-pointing unit normal on $\partial\Omega_n$; $\beta, f \colon \Omega \to \mathbb{R}$, $p \colon \partial\Omega_d \to \mathbb{R}$, and $q \colon \partial\Omega_q \to \mathbb{R}$ are given, and β is bounded below by some positive number; and the goal is to solve for the unknown function $u \colon \Omega \to \mathbb{R}$. For now, we'll ignore the details of the function spaces that u, β, f, p, and q are assumed to belong in.

For the numerical solution of (2.1a), we'll use a discretization based on the Finite Element Method (FEM). Briefly, an FEM discretization focuses on the weak formulation of the partial differential equation. The derivation involves multiplying the differential equation by a test function (chosen from some suitable function space), integrating by parts, and applying boundary conditions:

$$-\nabla \cdot (\beta \nabla u) = f \tag{2.2a}$$

$$\Rightarrow \int_\Omega -\nabla \cdot (\beta \nabla u)\, v = \int_\Omega f v \tag{2.2b}$$

$$\Rightarrow \int_\Omega \beta \nabla u \cdot \nabla v = \int_\Omega f v + \int_{\partial\Omega} (\beta \nabla u \cdot \hat{n})\, v \tag{2.2c}$$

$$\Rightarrow \int_\Omega \beta \nabla u \cdot \nabla v = \int_\Omega f v + \int_{\partial\Omega_n} q v \tag{2.2d}$$

where the derivation from (2.2c) to (2.2d) is possible by stipulating that $v \equiv 0$ on $\partial\Omega_d$. Clearly, then, if u satisfies (2.1a), then u also satisfies (2.2d) for all appropriate v. Under suitable conditions, it turns out to be the case that the converse holds as well. Since (2.2d) involves only first order derivatives in u, its analysis is often preferable to (2.1a).

We then suppose that u can be approximated by a function in a finite-dimensional function space (the *(discretized) solution space*) (e.g., continuous piecewise linear functions over some tessellation of Ω), i.e., $u \approx \tilde{u} = \sum_j u_j e_j$ for basis elements $\{e_j\}$. Likewise, v is selected from some other (often related) finite-dimensional function space (the *(discretized) test space*), thus giving a system of equations for the coefficients u_j. For example, if v is taken from the same space as $\tilde{u}$, and we let v be, in turn, each of the basis functions, then we obtain

$$\sum_j \left(\int_\Omega \beta \nabla e_j \cdot \nabla e_i \right) u_j = \int_\Omega f e_i + \int_{\partial \Omega_n} q e_i \tag{2.3}$$

for all i, which may be expressed as a linear system of equations:

$$A\vec{u} = \vec{b}, \tag{2.4}$$

where

$$A_{ij} = \int_\Omega \beta \nabla e_i \cdot \nabla e_j \tag{2.5a}$$

$$\vec{u}_i = u_i \tag{2.5b}$$

$$\vec{b}_i = \int_\Omega f e_i + \int_{\partial \Omega_n} q e_i \tag{2.5c}$$

Strictly speaking, the above is only correct if v varies *precisely* throughout the same function space which $\tilde{u}$ belongs to. This is generally not the case. For example, in the presence of Dirichlet boundary conditions, $\tilde{u}$ will generally be nonzero along Ω_d, however v will be chosen from a function space which vanishes along Ω_d (hence these functions spaces will only align under homogeneous Dirichlet boundary conditions). How this manifests itself in the linear system will be investigated in the exercises.

Note that the sparsity of A is directly related to the overlap of the supports of the e_j's. Typically we choose the e_j's such that the support of one basis element overlaps the supports of only a small constant number of other basis elements, in which case A will have a number of nonzero entries proportional to the number of basis elements.

Let's now focus on dimension $d = 1$ and use *linear finite elements*. Let Ω be an interval (a, b), and let $\{x_j\}$ be grid points in (a, b), with $a = x_0 < x_1 < \cdots < x_n = b$. Often times, the x_j's will be equally spaced, which makes the analysis simpler, but it's not necessary. We'll take $\tilde{u}$ (the discretized solution) and v (the discretized test function) to be continuous and *piecewise linear* on each segment (x_{j-1}, x_j), with $v(a)$ and/or $v(b)$ vanishing if a Dirichlet condition is specified at a and/or b, respectively. This space of continuous piecewise linear functions is spanned by the *nodal basis functions* (also called "hat" functions, because of the shape of their graphs) e_j, (uniquely) defined such that

$$e_j(x_k) = \delta_{jk} = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases} \tag{2.6}$$

1.1. Exercises

Unless otherwise noted, assume dimension $d = 1$ linear finite elements with the basis $\{e_j\}$ given by (2.6), although these results extend to higher order and higher dimensional finite elements.

(1) Here is one way to incorporate the Dirichlet boundary conditions into (2.5). For concreteness, suppose $\Omega_d = \{a\}$ (with $\Omega = (a, b)$). Then we still have $\tilde{u} = \sum_{j=0}^{n} u_j e_j$, but our space of test functions $\{v\}$ is slightly smaller, $\text{span}\{e_1, \ldots, e_n\}$, to ensure that $v(a) = 0$. Thus we obtain n equations of the form (2.3), plus one more equation from the Dirichlet boundary condition ($u_0 = u(a)$). This (ostensibly) gives enough equations to solve for the $n + 1$ coefficients $\{u_j\}$.

Another conceptual way to incorporate the Dirichlet boundary conditions is to consider the function $w = u - u(a)e_0$, which satisfies a similar Poisson problem as u but now with homogeneous Dirichlet boundary conditions. Thus, the space of test functions $\{v\}$ and the solution space of w coincide, and one may use eqs (2.5) directly.

In practice, one can actually just ignore the Dirichlet boundary conditions when initially computing the entries of A and $\vec{b}$ (i.e., just use (2.5a) and (2.5c)), then "correct" A and $\vec{b}$ to account for the Dirichlet boundary conditions afterwards. Describe a *simple* algorithm to effect this "correction" procedure. Try to keep A symmetric.

(2) Show that the linear system (2.4) has a unique solution if one specifies Dirichlet boundary conditions at either $x = a$ or $x = b$ or both (i.e., $u(a) = u_a$ and/or $u(b) = u_b$). For simplicity, you may assumed $\beta \equiv 1$. What happens if both boundary conditions are Neumann? Specifically address the conditions which $\vec{b}$ must satisfy, how this translates into a condition on f, and "how uniquely" $\tilde{u}$ is determined.

(3) Compute the integral in (2.5a) when $\beta \equiv 1$. If you wish, for simplicitly, you may assume the x_j's are equally spaced, i.e., $x_j = a + jh$, where $h = (b - a)/n$. What is the structure of the matrix A (e.g., describe the sparsity)?

(4) Give an expression to reasonably approximate the integrals in (2.5a) and (2.5c) for aribtrary β, f, and q (assume β and f are sufficiently "well-behaved"). You may have to consider boundary and interior grid vertices separately.

(5) Write some code (e.g., a Matlab script) to solve the following boundary value problem using linear finite elements:

$$\begin{aligned} -\left((1+x)\,u'(x)\right)' &= (1+x)^{-2} \quad x \in (0,1) \\ u(0) &= 1 \\ (1+(1))\,u'(1) &= -1/2 \end{aligned}$$

(6) The FEM terminology for the matrix A in (2.4) is the *(global) stiffness matrix* (the meaning of the term "global" will become obvious shortly). With dimension $d = 1$ linear finite elements, it is relatively straightforward to compute the entries of the stiffness matrix, even given that one has to account for differences between boundary and interior grid vertices. However, the situation becomes more complicated with higher order or higher dimensional finite elements, and it becomes tedious to consider all the special boundary cases. As such, it is common practice to assemble the global stiffness matrix element by element (i.e., cell by cell), with each element contributing partially to several entries of the global stiffness matrix. The collection of these contributions from a single element may

be put into a small matrix, the *local stiffness matrix* for that element. The computation of the local stiffness matrix is identical to that of the global stiffness matrix upon replacing the integrations over Ω with integrations localized over the element. For example, back to dimension $d = 1$ linear finite elements, the local stiffness matrix over $I_j = (x_{j-1}, x_j)$ would consist of 2^2 (nonzero) entries (since only 2 basis functions are supported on I_j). Each of these entries would be added to accumulated global stiffness matrix entries:

$$\begin{aligned} A_{j-1,j-1} &+= \int_{I_j} \beta \nabla e_{j-1} \cdot \nabla e_{j-1} \\ A_{j-1,j}, A_{j,j-1} &+= \int_{I_j} \beta \nabla e_{j-1} \cdot \nabla e_j \\ A_{jj} &+= \int_{I_j} \beta \nabla e_j \cdot \nabla e_j \end{aligned}$$

This suggests the following procedure to build the (global) stiffness matrix A. First, initialize the sparsity pattern of A (with zeros). Then iterate through each element and add the contributions of the element's local stiffness matrix to the corresponding entries in A.

Verify this procedure computes the same stiffness matrix as before, and modify your program from Exercise 5 to build the stiffness matrix in this fashion.

2. Neo-Hookean Elasticity with Quasistatics Evolution in Dimension 1

Let us now consider the numerical solution of the equations of elasticity with the Neo-Hookean constitutive model. To begin, for simplicity, we will consider using quasistatic evolution in dimension $d = 1$. The subsequent section will extend this into dimension $d = 2$ and add inertial effects.

2.1. Elasticity

Recall that the equations of elasticity are given by the boundary value problem

$$\rho_0(\mathbf{X}) \frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \quad \in \Omega(t) \tag{2.7a}$$

$$\mathbf{u}(t, \cdot) = \mathbf{g}(t, \cdot) \quad \in \partial\Omega_d(t) \tag{2.7b}$$

$$(\mathbf{P} \cdot \hat{\mathbf{n}})(t, \cdot) = \mathbf{h}(t, \cdot) \quad \in \partial\Omega_n(t) \tag{2.7c}$$

where $\rho_0 \colon \Omega_0 \to \mathbb{R}$ is the mass density (as a function of $\mathbf{X}$, the undeformed coordinates); $\mathbf{u} \colon \Omega_0 \to \Omega(t), \mathbf{X} \mapsto \boldsymbol{\phi}(\mathbf{X}) - \mathbf{X}$ is the unknown displacement; $\mathbf{P}$ is the first Piola-Kirchoff stress (which takes a specific form for Neo-Hookean, to be given later); $\mathbf{f}^{\text{ext}}$ is the given (external) force; and $\mathbf{g}$ and $\mathbf{h}$ specify the Dirichlet and Neumann boundary conditions, respectively. For simplicity, we'll assume a uniform mass density, i.e., $\rho_0 \equiv 1$.

2.2. Neo-Hookean Constitutive Model

The Neo-Hookean constitutive model relates the stress $\mathbf{P}$ to the deformation gradient $\mathbf{F} := \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}$ via

$$\Psi(\mathbf{F}) := \frac{\mu}{2}\left(F_{ij}F_{ji} - 2\right) - \mu \log J + \frac{\lambda}{2}\log^2 J, \tag{2.8a}$$

$$\mathbf{P}(\mathbf{F}) := \frac{\partial \Psi}{\partial \mathbf{F}} = \mu \mathbf{F} + \left(\lambda \log J - \mu\right)\mathbf{F}^{-T}. \tag{2.8b}$$

(Recall that $J := \det \mathbf{F}$.) This manifests itself in dimension $d = 1$ in terms of the displacement u as

$$P(u) = \mu\left(\frac{du}{dX} + 1\right) + \left(\lambda \log\left(\frac{du}{dX} + 1\right) - \mu\right)\frac{1}{\frac{du}{dX} + 1}.$$

2.3. Inversion-Robust Neo-Hookean

Neo-Hookean as formulated above will not be robust to element inversions, due to the $\log J$ terms. To remedy this, we replace the logarithm in (2.8a) with a cubic Taylor approximation around 1:

$$\begin{aligned} r(x) &= (x-1) - \frac{1}{2}(x-1)^2 + \frac{1}{3}(x-1)^3 \\ &= -\frac{11}{6} + 3x - \frac{3}{2}x^2 + \frac{1}{3}x^3, \end{aligned} \tag{2.9a}$$

$$\begin{aligned} r'(x) &= 1 - (x-1) + (x-1)^2 \\ &= 3 - 3x + x^2. \end{aligned} \tag{2.9b}$$

$$\begin{aligned} r''(x) &= -1 + 2(x-1) \\ &= -3 + 2x. \end{aligned} \tag{2.9c}$$

This gives

$$\begin{aligned} \Psi(\mathbf{F}) &= \frac{\mu}{2}\left(F_{ij}F_{ji} - 2\right) - \mu r(J) + \frac{\lambda}{2}r(J)^2, \\ \mathbf{P}(\mathbf{F}) &= \frac{\partial \Psi}{\partial \mathbf{F}} = \mu \mathbf{F} + \left(\lambda r(J) - \mu\right) r'(J)\frac{\partial J}{\partial \mathbf{F}}. \end{aligned}$$

Again, specializing this for $d = 1$ dimension, we obtain

$$P(u) = \mu\left(\frac{du}{dX} + 1\right) + \left(\lambda r\left(\frac{du}{dX} + 1\right) - \mu\right) r'\left(\frac{du}{dX} + 1\right). \tag{2.10}$$

2.4. Quasistatic Evolution

We begin by studying (2.7) at equilibrium, which is the basis for quasistatic evolution. This reduces the equations to

$$-\nabla^{\mathbf{X}} \cdot \mathbf{P} = \mathbf{f}^{\text{ext}}. \tag{2.11}$$

The equivalent weak formulation is

$$\int_{\Omega_0} w_{i,j} P_{ij} d\mathbf{X} = \int_{\partial \Omega_n} w_i h_i dS(\mathbf{X}) + \int_{\Omega_0} w_i f_i^{\text{ext}} d\mathbf{X}$$

for all test functions $\mathbf{w}$. Recall that summation over repeated indices is implied, and comma'ed indices indicate differentiation.

As for Poisson, we let the coordinates of $\mathbf{w}$ vary over the nodal basis functions N_i. In dimension $d = 1$, this reduces to the system of equations

$$0 = q_i(\vec{u}) := \int_a^b \frac{\partial N_i}{\partial X} P(F(u(X)))dX - b_i; \tag{2.12a}$$

$$b_i := \int_a^b N_i f^{\text{ext}} dX + [\text{Neumann boundary terms}] \tag{2.12b}$$

for each grid vertex i, where $u := u_i N_i$. The Neumann boundary terms consist of neither, one, or both of $N_i(b)h(b)$ and/or $-N_i(a)h(a)$, depending on whether b and/or a, respectively, belong to $\partial\Omega_n$.

For Neo-Hookean, P depends *non-linearly* on the displacement u, hence one must use a non-linear solver, such as Newton iteration, to solve (2.12) for $\vec{u} := (u_i)$. The Newton step looks like

$$\frac{\partial q_i}{\partial \vec{u}}(\vec{u})\,\Delta\vec{u} + q_i(\vec{u}) = 0;$$

$$\vec{u} \leftarrow \vec{u} + \Delta\vec{u}$$

where

$$\frac{\partial q_i}{\partial u_j}(\vec{u}) = \int_a^b \frac{\partial N_i}{\partial X}\frac{\partial P}{\partial F}(F(u))\frac{\partial N_j}{\partial X} dX$$

Thus, the computation of $\Delta\vec{u}$ in each Newton iteration amounts to solving a variable coefficient Poisson problem. The coefficient is $\partial P/\partial F$, which we can express via (2.10) as

$$\frac{\partial P}{\partial F} = \mu + \lambda r'\left(\frac{du}{dX} + 1\right)^2 + \left(\lambda r\left(\frac{du}{dX} + 1\right) - \mu\right) r''\left(\frac{du}{dX} + 1\right).$$

This gives all the necessary pieces to implement a quasistatic evolution of the equations of elasticity (2.7).

2.5. Implementation

Let us assume a regular grid on (a, b), such that we have n grid vertices $x_i := a + (i-1)\Delta x$, $\Delta x := (b-a)/(n-1)$, and i ranges from 1 to n, inclusive. We break the implementation of such a solve into several steps, outlined below and expounded upon in the subsequence subsubsections.

- Compute $\vec{b} := (b_1, \ldots, b_n)$ (from (2.12b)).
- Implement a procedure to compute $\vec{q} := (q_1, \ldots, q_n)$ (from (2.12a)).
- Implement a procedure to compute $\partial q/\partial\vec{u}$.
- Solve for the Newton increment $\Delta\vec{u}$.

We provide psuedocode for all steps in a syntax similar to Matlab or Octave.

2.5.1. *Compute $\vec{b}$.* We begin by considering the computation of $\vec{b}$. This is a natural place to start because, first, it remains constant throughout the Newton iterations within a single time step; and second, its computation is identical to that for the right-hand side in Poisson's equation. One natural algorithm to compute $\vec{b}$ might be a vertex-based approach, where each b_i is computed explicitly and in isolation, and we consider the boundary grid vertices specially. However, as outlined in the last exercise from the previous section, an element-based approach scales better with dimension and order, hence we will focus on this approach for this and subsequent computations.

The implementation thus boils down to computing the integrals

$$\int_{x_i}^{x_{i+1}} N_j f^{\text{ext}} dX$$

where $j \in \{i, i+1\}$ (all other values of j integrate to zero). The result of this integral is then added to a running accumulation of b_j. We'll assume we've been given the values of f^{ext} at the midpoints of each interval (x_i, x_{i+1}), and approximate the integral via the midpoint rule, leading to the following implementation:

```
function b = eval_b(x1, xn, n, fext, h1, hn)
b = zeros([n 1]);
dX = (xn - x1) / (n - 1);
for i = 1:n-1
    b(i  ) += fext(i) * dX / 2; % fext(i) is fext evaluated at
    b(i+1) += fext(i) * dX / 2; %     x_{i+1/2}
end
b(1) += h1; % add Neumann boundary terms
b(n) += hn; % add Neumann boundary terms
```

2.5.2. *Computing $\vec{q}$.* We turn now toward computing $\vec{q}$ via (2.12a) given the current Newton approximation u. Again, we'll use an element-based approach, meaning we need to compute the integrals

$$\int_{x_i}^{x_{i+1}} \frac{\partial N_j}{\partial X} P\left(\frac{\partial u}{\partial X}\right) dX$$

where, again, $j \in \{i, i+1\}$. Note the components of the integrand are constant; indeed, on (x_i, x_{i+1}),

$$\begin{aligned} \frac{\partial N_i}{\partial X} &= -\frac{1}{\Delta X} \\ \frac{\partial N_{i+1}}{\partial X} &= \frac{1}{\Delta X} \\ P\left(\frac{\partial u}{\partial X}\right) &= P\left(\frac{u_{i+1} - u_i}{x_{i+1} - x_i}\right). \end{aligned}$$

It follows that

$$\int_{x_i}^{x_{i+1}} \frac{\partial N_j}{\partial X} P\left(\frac{\partial u}{\partial X}\right) dX = \pm P\left(\frac{u_{i+1} - u_i}{x_{i+1} - x_i}\right).$$

This leads to the following procedure to compute $\vec{q}$:

```
function q = eval_q(x1, xn, n, mu, lambda, b, u)
q = -b;
dX = (xn - x1) / (n - 1);
for i = 1:n-1
    du_dx = (u(i+1) - u(i)) / dX;
    r     = du_dx - du_dx^2 / 2 + du_dx^3 / 3;
    dr    = 1 - du_dx + du_dx^2;
    P     = mu * (du_dx + 1) + (lambda * r - mu) * dr;
    q(i  ) -= P; % (-) since dN_{i  }/dX = -1/dX
    q(i+1) += P; % (+) since dN_{i+1}/dX = +1/dX
end
```

2.5.3. *Computing $\partial q/\partial \vec{u}$.* The nex step is to consider the computation of $\partial q/\partial \vec{u}$, which will be a sparse $n \times n$ matrix. The integrals we need to compute this time are

$$\int_{x_i}^{x_{i+1}} \frac{\partial N_j}{\partial X} \frac{\partial N_k}{\partial X} \frac{\partial P}{\partial F} \left(\frac{\partial u}{\partial X} \right) dX$$

where $j, k \in \{i, i+1\}$. Again, the integrand is constant, so the integration is trivial. The expression for $\partial P/\partial F$ comes from differentiating (2.10):

$$\frac{\partial P}{\partial F} = \mu + \lambda r'(F)^2 + (\lambda r(F) - \mu)\, r''(F).$$

The procedure looks as follows:

```
function dq_du = eval_dq_du(x1, xn, n, mu, lambda, u)
% create an empty tri-diagonal sparse matrix
dq_du = sparse([1:n 2:n 1:n-1], [1:n 1:n-1 2:n], 0);
dX = (xn - x1) / (n - 1);
for i = 1:n-1
    du_dx = (u(i+1) - u(i)) / dX;
    r     = du_dx - du_dx^2 / 2 + du_dx^3 / 3;
    dr    = 1 - du_dx + du_dx^2;
    d2r   = -1 + 2 * du_dx;
    dP_dF = mu + lambda * dr^2 + (lambda * r - mu) * d2r;
    dq_du(i  ,i  ) += dP_dF / dX;
    dq_du(i  ,i+1) -= dP_dF / dX;
    dq_du(i+1,i  ) -= dP_dF / dX;
    dq_du(i+1,i+1) += dP_dF / dX;
end
```

2.5.4. *Computing the Newton Increment $\Delta \vec{u}$.* With the above procedures in hand, computing the Newton increment $\Delta \vec{u}$ becomes very easy. There is one thing to note regarding Dirichlet boundary conditions. When computing $\Delta \vec{u}$, we will assume that the current Newton approximation $\vec{u}$ agrees with its Dirichlet boundary conditions at the current time-step. Hence, we will want zeros in $\Delta \vec{u}$ in those entries corresponding to Dirichlet boundary conditions, which can be effected by slightly altering the matrix and right-hand side:

```
dq_du = eval_dq_du(x1, xn, n, mu, lambda, u);
q     = eval_q    (x1, xn, n, mu, lambda, b, u);
if dirichlet_bc_at_1
    dq_du(1,1) = 1;
    dq_du(1,2) = 0;
    dq_du(2,1) = 0;
    q(1) = 0;
end
if dirichlet_bc_at_n
    dq_du(n,n) = 1;
    dq_du(n,n-1) = 0;
    dq_du(n-1,n) = 0;
    q(n) = 0;
end
delta_u = dq_du \ -q;
```

2.5.5. *The Full Newton Method.* We now have all the pieces to solve the nonlinear equation (2.12):

```
function u = solve( ...
    x1, xn, n, ...
    mu, lambda, ...
    dirichlet_bc_at_1, g1, dirichlet_bc_at_n, gn, ...
    h1, hn, ... % Neumann boundary conditions
    fext, ...
    tol, u)
b = construct_b(x1, xn, n, fext, h1, hn);
if dirichlet_bc_at_1
    u(1) = g1;
end
if dirichlet_bc_at_n
    u(n) = gn;
end
do
    dq_du = eval_dq_du(x1, xn, n, mu, lambda, u);
    q     = eval_q    (x1, xn, n, mu, lambda, b, u);
    if dirichlet_bc_at_1
        dq_du(1,1) = 1;
        dq_du(1,2) = 0;
        dq_du(2,1) = 0;
        q(1) = 0;
    end
    if dirichlet_bc_at_n
        dq_du(n,n) = 1;
        dq_du(n,n-1) = 0;
        dq_du(n-1,n) = 0;
        q(n) = 0;
    end
    delta_u = dq_du \ -q;
    u += delta_u;
while max(abs(delta_u)) < tol
```

2.5.6. *Example Problem.* We can test the code with the following example problem.

- $\Omega_0 = (0, 1)$ (so $a = 0$ and $b = 1$)
- Dirichlet boundary condition at $x = 0$: $u(0, t) = g(0, t) = \sin t$
- Neumann boundary condition at $x = 1$: $P(1, t) = 0$
- $f^{\text{ext}} \equiv 0$
- $E = 1000$ and $\nu = 0.3$; so $\mu = E/(2(1 + \nu)) = 384.$ and $\lambda = E\nu/((1 + \nu)(1 - 2\nu)) = 577.$

Since a quasistatics evolution has no inertial terms, we expect the solution displacement to simply be $u(X, t) = \sin t = g(0, t)$.

This is a pretty simple test, so feel free to experiment with other combinations of parameters.

3. Neo-Hookean Elasticity with Backward Euler Evolution in Dimension 2

Let us again recall the elasticity equations:

$$\rho_0 \frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \quad \in \Omega(t) \tag{2.14a}$$

$$\mathbf{u}(t,\cdot) = \mathbf{g}(t,\cdot) \quad \in \partial\Omega_d(t) \tag{2.14b}$$

$$(\mathbf{P} \cdot \hat{\mathbf{n}})(t,\cdot) = \mathbf{h}(t,\cdot) \quad \in \partial\Omega_n(t) \tag{2.14c}$$

With non-negligible inertial effects, a more sophisticated temporal evolution than quasistatic evolution must be used. One such scheme is Backward Euler, which is desirable due to its unconditional stability and dissipation of oscillatory errors. We'll also see that it is not significantly more complex than quasistatics. However, the transition from dimension $d = 1$ to dimension $d = 2$ does offer a fair amount of complexity, which we shall investigate.

3.1. Backward Euler

To formulate the Backward Euler time-stepping, we introduce an auxiliary variable, $\mathbf{v}$ ("velocity"), to transform (2.14) into a first-order system (in time):

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{v};$$
$$\rho_0 \frac{\partial \mathbf{v}}{\partial t} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}}.$$

The Backward Euler time discretization thus gives

$$\frac{1}{\Delta t}\left(\mathbf{u}(t+\Delta t,\cdot) - \mathbf{u}(t,\cdot)\right) = \mathbf{v}(t+\Delta t,\cdot); \tag{2.15a}$$

$$\rho_0 \frac{1}{\Delta t}\left(\mathbf{v}(t+\Delta t,\cdot) - \mathbf{v}(t,\cdot)\right) = \left(\nabla^{\mathbf{X}} \cdot \mathbf{P}\right)_{t+\Delta t} + \mathbf{f}^{\text{ext}}(t+\Delta t,\cdot). \tag{2.15b}$$

Eliminating $\mathbf{v}(t+\Delta t,\cdot)$ from (2.15) gives a (nonlinear) equation which must be solved for $\mathbf{u}(t+\Delta t,\cdot)$ at each time-step. For present notational purposes, let us refer to the unknown $\mathbf{u}(t+\Delta t,\cdot)$ as simply $\mathbf{u}$, and to the known $\mathbf{u}(t,\cdot)$ and $\mathbf{v}(t,\cdot)$ as $\mathbf{u}_0$ and $\mathbf{v}_0$, respectively. Thus, (2.15) is equivalent to

$$\rho_0 \mathbf{u} - \Delta t^2 \nabla^{\mathbf{X}} \cdot \mathbf{P} = \rho_0(\mathbf{u}_0 + \Delta t \mathbf{v}_0) + \Delta t^2 \mathbf{f}^{\text{ext}}, \tag{2.16}$$

where $\nabla^{\mathbf{X}} \cdot \mathbf{P}$ is evaluated at $t + \Delta t$ (hence depends on the uknown $\mathbf{u}$) and $\mathbf{f}^{\text{ext}}$ is also evaluated at $t + \Delta t$.

3.2. Weak Formulation and Finite Element Discretization

Let us now derive the weak formulation of equation (2.16) to obtain a finite element discretization in space. We dot product by a test function $\mathbf{w}$, integrate over Ω_0, apply integration by parts, and simplify the integrals over $\partial\Omega_0$ by stipulating that $\mathbf{w} \equiv 0$ on $\partial\Omega_d := \partial\Omega_d(t + \Delta t)$ and substituting the Neumann condition over $\partial\Omega_n := \partial\Omega_n(t + \Delta t)$. It will be notationally convenient, at this point, to once again use index notation and implied summation. We will use subscripts to denote coordinates $(1, \ldots, d)$ and (later) superscripts to denote grid vertices.

(2.17a)

$$\rho_0 u_i w_i - \Delta t^2 P_{ij,j} w_i = \left(\rho_0 \left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) w_i$$

(2.17b)

$$\int_{\Omega_0} \rho_0 u_i w_i - \Delta t^2 P_{ij,j} w_i = \int_{\Omega_0} \left(\rho_0 \left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) w_i$$

(2.17c)

$$\int_{\Omega_0} \rho_0 u_i w_i + \Delta t^2 P_{ij} w_{i,j} = \int_{\Omega_0} \left(\rho_0 \left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) w_i + \Delta t^2 \int_{\partial\Omega_n} h_i w_i.$$

For the finite element discretization, we'll suppose that $\Omega_0 \subset \mathbb{R}^d$ is tesselated by simplices $S^1, \ldots, S^N$ with grid vertices $\mathbf{X}^1, \ldots, \mathbf{X}^n \in \mathbb{R}^d$ and with S_i^α denoting the i^{th} grid vertex of simplex S^α (we shall let "$a \in S^\alpha$" denote the relation that grid vertex a is one of the grid vertices of simplex S^α). Our finite element space will consist of continuous $\mathbb{R}^d$-valued functions which are affine over each simplex S^α. This space is spanned by the nodal basis ("hat") functions $\{N^a\}$; N^a takes the value 1 at $\mathbf{X}^a$ and the value 0 at all other grid vertices. (Strictly speaking, of course, it's really the *projection* of this function space on each coordinate that is spanned by the nodal basis functions.)

We thus take $\mathbf{w} = (w_i) = (\delta_{ij} N^a)$ in (2.17c), where j ranges over $1, \ldots, d$ and a ranges over $1, \ldots, n$, to obtain dn equations for $\mathbf{u}$. Likewise, we discretize each coordinate of $\mathbf{u}$ as $u_i = u_i^b N^b$, giving a (nonlinear) system of equations for the coefficients u_i^b:

$$0 = q_i^a(\mathbf{u}) := \left(\int_{\Omega_0} \rho_0 N^a N^b\right) u_i^b + \Delta t^2 \left(\int_{\Omega_0} P_{ij} N^a_{,j}\right) - b_i^a; \tag{2.18a}$$

$$b_i^a := \int_{\Omega_0} \left(\rho_0 \left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) N^a + \Delta t^2 \int_{\partial\Omega_n} h_i N^a. \tag{2.18b}$$

It is worth noting that this is very similar to the nonlinear system of equations which arise in a discretization of quasistatic evolution. The differences are only in the right-hand side $\vec{b}$ and the additional identity-like block to (2.18a).

Like in dimension $d = 1$, we shall solve (2.18) via Newton iteration:

$$\frac{\partial \vec{q}}{\partial \vec{u}}(\vec{u})\, \Delta\vec{u} + \vec{q}(\vec{u}) = 0;$$
$$\vec{u} \leftarrow \vec{u} + \Delta\vec{u}.$$

3.3. Implementation Details in Dimension 2

We will now consider specifically dimension $d = 2$ and go through the implementation details of the various computational steps necessary to advance one time step, from time t to time $t + \Delta t$.

3.3.1. *Computing $\vec{b}$.* Like in dimension $d = 1$, we will evaluate $\vec{b}$ via an element-based loop, requiring the evaluation of the integrals

$$\int_{S^\alpha} \left(\rho_0 \left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) N^a.$$

(We'll address the Neumann terms later.) Let us suppose we are given the value of $\mathbf{f}^{\text{ext}}$ at each grid vertex $\mathbf{X}^b$ and the value of ρ_0 for each element S^α. Then

we can expand each of $(u_0)_i$, $(v_0)_i$, and f_i^{ext} as $(u_0)_i^b N^b$, $(v_0)_i^b N^b$, and $f_i^{\text{ext},b} N^b$, respectively, giving

$$\begin{aligned}
&\int_{S^\alpha} \left(\rho_0 \left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) N^a \\
&\quad = \int_{S^\alpha} \left(\rho_0^\alpha \left((u_0)_i^b N^b + \Delta t (v_0)_i^b N^b\right) + \Delta t^2 f_i^{\text{ext},b} N^b\right) N^a \\
&\quad = \int_{S^\alpha} \left(\rho_0^\alpha \left((u_0)_i^b + \Delta t (v_0)_i^b\right) + \Delta t^2 f_i^{\text{ext},b}\right) N^a N^b \\
&\quad = \left(\rho_0^\alpha \left((u_0)_i^b + \Delta t (v_0)_i^b\right) + \Delta t^2 f_i^{\text{ext},b}\right) \int_{S^\alpha} N^a N^b.
\end{aligned}$$

Note that this will be nonzero only for $a, b \in S^\alpha$. It thus suffices to evaluate $\int_{S^\alpha} N^a N^b$ for $a, b \in S^\alpha$. These may be computed by a change of coordinates to the standard $\{(0,0),(1,0),(0,1)\}$ triangle, ultimately giving

$$\int_{S^\alpha} N^a N^a = \frac{1}{6} \operatorname{area}\left(S^\alpha\right), \tag{2.20a}$$

$$\int_{S^\alpha} N^a N^b = \frac{1}{12} \operatorname{area}\left(S^\alpha\right) \quad (a \neq b) \tag{2.20b}$$

and we can compute $\operatorname{area}\left(S^\alpha\right)$ via

$$\operatorname{area}\left(S^\alpha\right) = \frac{1}{2} \begin{vmatrix} \mathbf{X}^{S_1^\alpha} & \mathbf{X}^{S_2^\alpha} & \mathbf{X}^{S_3^\alpha} \\ 1 & 1 & 1 \end{vmatrix}.$$

In the implementation below, we assume $\operatorname{area}\left(S^\alpha\right)$ and $\int_{\partial\Omega_n} \mathbf{h} N^a$ have been precomputed for each α and a, respectively.

```
function b = eval_b(tris, X, tri_areas, rho, fext, h, u0, v0, dt)
% tris(k,:)    = the 3 grid vertices within triangle k
% X(a,:)       = the coordinates of grid vertex a
% tri_areas(k) = the area of triangle k
% rho(k)       = the mass density within triangle k
% fext(a,:)    = the external force on grid vertex a
% h(a,:)       = the Neumann boundary condition on grid vertex a
% u0(a,:)      = the previous time-step's displacement at
%                      grid vertex a
% v0(a,:)      = the previous time step's velocity at grid vertex a
% dt           = the time step increment
b = dt^2 * h;
for k = 1:size(tris,1)
    tri = tris(k,:);
    b(tri,:) += tri_areas(k) ...
              * [2 1 1;1 2 1;1 1 2]/12 ...
              * (rho(k) * (u0(tri,:) + dt * v0(tri,:)) ...
               + dt^2 * fext(tri,:));
end
```

3.3.2. *Computing $\vec{q}$.* Computation of $\vec{q}$ requires computing the integrals

$$\begin{aligned}&\left(\int_{S^\alpha} \rho_0 N^a N^b\right) u_i^b + \Delta t^2 \left(\int_{S^\alpha} P_{ij} N^a_{,j}\right) - b_i^a \\ &\quad = \rho_0^\alpha \left(\int_{S^\alpha} N^a N^b\right) u_i^b + \Delta t^2 \left(\int_{S^\alpha} P_{ij} N^a_{,j}\right) - b_i^a,\end{aligned}$$

where we have used the fact that ρ_0 is given element-wise. We already know the value of $\int N^a N^b$ from the previous subsubsection, so we need only address $\int P_{ij} N^a_{,j}$.

To this end, recall that inversion-robust Neo-Hookean defines $\mathbf{P}$ in terms of $\mathbf{F}$ by

$$\begin{aligned}\mathbf{P} &= \mu\mathbf{F} + (\lambda r(J) - \mu)\, r'(J) \frac{\partial J}{\partial \mathbf{F}} \\ &= \mu\mathbf{F} + (\lambda r(J) - \mu)\, r'(J) J \mathbf{F}^{-T}\end{aligned} \tag{2.21a}$$

$$r(x+1) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 \tag{2.21b}$$

$$r'(x+1) = 1 - x + x^2. \tag{2.21c}$$

We thus must compute $\mathbf{F}$ from $\mathbf{u}$. First, notice that $N^a(\mathbf{X})$ may be interpreted as the barycentric coordinate of $\mathbf{X}$ with respect to $\mathbf{X}^a$ within S^α (assuming that $a \in S^\alpha$). In other words, $N^a(\mathbf{X})$ is equal to ξ^a, where

$$\begin{pmatrix} \mathbf{X}^{S_1^\alpha} & \mathbf{X}^{S_2^\alpha} & \mathbf{X}^{S_3^\alpha} \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \xi^{S_1^\alpha} \\ \xi^{S_2^\alpha} \\ \xi^{S_3^\alpha} \end{pmatrix} = \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix}.$$

It follows that

$$\mathbf{N}^{S^\alpha}(\mathbf{X}) = \begin{pmatrix} N^{S_1^\alpha}(\mathbf{X}) \\ N^{S_2^\alpha}(\mathbf{X}) \\ N^{S_3^\alpha}(\mathbf{X}) \end{pmatrix} = \begin{pmatrix} \mathbf{X}^{S_1^\alpha} & \mathbf{X}^{S_2^\alpha} & \mathbf{X}^{S_3^\alpha} \\ 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix}$$

and hence

$$\frac{\partial \mathbf{N}^{S^\alpha}}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial N^{S_1^\alpha}}{\partial \mathbf{X}} \\ \frac{\partial N^{S_2^\alpha}}{\partial \mathbf{X}} \\ \frac{\partial N^{S_3^\alpha}}{\partial \mathbf{X}} \end{pmatrix} = \begin{pmatrix} \mathbf{X}^{S_1^\alpha} & \mathbf{X}^{S_2^\alpha} & \mathbf{X}^{S_3^\alpha} \\ 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

With the ability to compute $\partial \mathbf{N}^{S^\alpha} / \partial \mathbf{X}$ in hand, and using the fact that $\mathbf{u} = \mathbf{u}^a N^a$, we can now compute $\mathbf{F}$:

$$\mathbf{F} = \frac{\partial \mathbf{u}}{\partial \mathbf{X}} + \mathbf{I} = \mathbf{u}^{S^\alpha} \frac{\partial \mathbf{N}^{S^\alpha}}{\partial \mathbf{X}} + \mathbf{I} \tag{2.22}$$

Using (2.22) in (2.21) will thus give $\mathbf{P}$. Finally, we can collect all $P_{ij} N^a_{,j}$ terms into a single (constant) 2×3 matrix as $\mathbf{P}\left(\partial \mathbf{N}^{S^\alpha} / \partial \mathbf{X}\right)^T$ (the rows are indexed by i, the columns by a). The implementation for computing $\vec{q}$ is then as follows.

```
function q = eval_q(tris, X, tri_areas, rho, mu, lambda, u, b, dt)
% tris(k,:)     = the 3 grid vertices within triangle k
% X(a,:)        = the coordinates of grid vertex a
% tri_areas(k)  = the area of triangle k
% rho(k)        = the mass density within triangle k
% mu            = mu
% lambda        = lambda
```

```
% u(a,:)       = the vector of u_i's at grid vertex a
% b(a,:)       = the vector of b_i's at grid vertex a
% dt           = the time step increment
q = -b;
for k = 1:size(tris,1)
    tri = tris(k,:);
    dN_dX = [X(tri,:)';1 1 1] \ [1 0;0 1;0 0];
    F     = u(tri,:)' * dN_dX + [1 0;0 1];
    J     = det(F);
    dJ_dF = [F(2,2) -F(2,1);-F(1,2) F(1,1)];
    rJ    = (J-1) - (J-1)^2 / 2 + (J-1)^3 / 3;
    drJ   = 1 - (J-1) + (J-1)^2;
    P     = mu * F + (lambda * rJ - mu) * drJ * dJ_dF;
    q(tri,:) += tri_areas(k) ...
              * (rho(k) * [2 1 1;1 2 1;1 1 2]/12 * u(tri,:) ...
               + dt^2 * (dN_dX * P'));
end
```

3.3.3. *Computing $\partial\vec{q}/\partial\vec{u}$.* It is probably not too surprising that the most complicated computation is evaluating $\partial\vec{q}/\partial\vec{u}$. To begin, we use (2.18a) and differentiate with respect to u_j^b:

$$\frac{\partial q_i^a}{\partial u_j^b} = \delta_{ij}\int_{\Omega_0}\rho_0 N^a N^b + \Delta t^2 \int_{\Omega_0}\frac{\partial P_{ik}}{\partial F_{\ell m}}\frac{\partial F_{\ell m}}{\partial u_j^b}N^a_{,k}.$$

Given that $\mathbf{F} = \mathbf{u}^{S^\alpha}\left(\partial \mathbf{N}^{S^\alpha}/\partial\mathbf{X}\right) + \mathbf{I}$ (from (2.22)), one can show that $\partial F_{\ell m}/\partial u_j^b = \delta_{j\ell}N^b_{,m}$. Substituting this into the equation above yields

$$\frac{\partial q_i^a}{\partial u_j^b} = \delta_{ij}\int_{\Omega_0}\rho_0 N^a N^b + \Delta t^2 \int_{\Omega_0}\frac{\partial P_{ik}}{\partial F_{jm}}N^a_{,k}N^b_{,m}. \tag{2.23}$$

We can already compute the first integral via (2.20), so the challenge remains to evaluate $\partial P_{ik}/\partial F_{jm}$. Using (2.21),

$$\begin{aligned}\frac{\partial P_{ik}}{\partial F_{jm}} = {}& \delta_{ij}\delta_{km}\mu \\ &+ \left(\lambda r'(J)^2 + (\lambda r(J) - \mu)\, r''(J)\right)\frac{\partial J}{\partial F_{ik}}\frac{\partial J}{\partial F_{jm}} \\ &+ (\lambda r(J) - \mu)\, r'(J)\frac{\partial^2 J}{\partial F_{ik}\partial F_{jm}}.\end{aligned}$$

This gives all the information necessary to evaluate $\partial\vec{q}/\partial\vec{u}$, though the implementation is still relatively complex from all the implicitly summed indices (as a sanity check, one sees the symmetry when interchanging $a, i \leftrightarrow b, j$). We note that $\partial\vec{q}/\partial\vec{u}$ is stored as a $2n \times 2n$ matrix. Row r within the matrix refers to grid vertex $\lceil r/2 \rceil$ and coordinate $2 - r \bmod 2$. Thus, in practice, when inverting $\partial\vec{q}/\partial\vec{u}$, one needs to "flatten" the right-hand side and then "unflatten" the solution (which will turn out to be $\Delta\vec{u}$ in the Newton iteration).

```
function dq_du = eval_dq_du( ...
    tris, X, tri_areas, rho, mu, lambda, u, dt)
% tris(k,:)    = the 3 grid vertices within triangle k
% X(a,:)       = the coordinates of grid vertex a
```

```
% tri_areas(k) = the area of triangle k
% rho(k)       = the mass density within triangle k
% mu           = mu
% lambda       = lambda
% u(a,:)       = the vector of u_i's at grid vertex a
% dt           = the time step increment
n = size(X,1);
dq_du_vals = zeros([4 * 9 * size(tris,1) 1]);
dq_du_rows = dq_du_vals;
dq_du_cols = dq_du_vals;
index = 1;
for k = 1:size(tris,1)
    tri = tris(k,:);
    dN_dX = [X(tri,:)';1 1 1] \ [1 0;0 1;0 0];
    F     = u(tri,:)' * dN_dX + [1 0;0 1];
    J     = det(F);
    dJ_dF = [F(2,2) -F(2,1);-F(1,2) F(1,1)];
    rJ    = (J-1) - (J-1)^2 / 2 + (J-1)^3 / 3;
    drJ   = 1 - (J-1) + (J-1)^2;
    d2rJ  = -1 + 2 * (J-1);
    c1    = lambda * drJ^2 + (lambda * rJ - mu) * d2rJ;
    c2    = (lambda * rJ - mu) * drJ;
    for i = 1:2
        for j = 1:2
            dPik_dFjm = c1 * dJ_dF(i,:)' * dJ_dF(:,j)';
            if(i == j)
                dPik_dFjm(i,i) += mu;
            else
                dPik_dFjm(i,j) += c2;
                dPik_dFjm(j,i) -= c2;
            end
            local_dq_du = dt^2 * dN_dX * dPik_dFjm * dN_dX';
            if(i == j)
                local_dq_du += rho(k) * [2 1 1;1 2 1;1 1 2]/12;
            end
            local_dq_du *= tri_areas(k);
            for a = 1:3
                for b = 1:3
                    r = 2 * tri(a) + i - 2;
                    c = 2 * tri(b) + j - 2;
                    dq_du_vals(index) = local_dq_du(a,b);
                    dq_du_rows(index) = r;
                    dq_du_cols(index) = c;
                    ++index;
                end
            end
        end
    end
```

```
end
dq_du = sparse(dq_du_rows, dq_du_cols, dq_du_vals, 2*n, 2*n);
```

3.3.4. *Computing the Newton Increment $\Delta\vec{u}$.* As alluded to in the previous subsubsection, the stored matrix representing $\partial\vec{q}/\partial\vec{u}$ is $2n\times 2n$, requiring some "flattening" and "unflattening" (alternatively, one can use flattened vectors throughout, though the index manipulation would be somewhat more complicated). This is effected via the *reshape* function used below.

```
n = size(X,1);
dq_du = eval_dq_du(tris, X, tri_areas, rho, mu, lambda, u, dt);
q     = eval_q    (tris, X, tri_areas, rho, mu, lambda, u, b, dt);
for i = 1:length(dirichlet_vertices)
    a = dirichlet_vertices(i);
    dq_du(2*a-1,:) = 0;
    dq_du(2*a  ,:) = 0;
    dq_du(:,2*a-1) = 0;
    dq_du(:,2*a  ) = 0;
    dq_du(2*a-1,2*a-1) = 1;
    dq_du(2*a  ,2*a  ) = 1;
    q(a,:) = [0 0];
end
delta_u = reshape(dq_du \ reshape(-q', [2*n 1]), [2 n])';
```

3.3.5. *The Full Newton Method.* The entire solve procedure for a single time step looks as follows.

```
function u = solve( ...
    tris, X, tri_areas, ...
    rho, fext, mu, lambda, ...
    h, ...
    dirichlet_vertices, dirichlet_values, ...
    u0, v0, ...
    dt, ...
    tol, u)
b = eval_b(tris, X, tri_areas, rho, fext, h, u0, v0, dt);
for i = 1:length(dirichlet_vertices)
    u(dirichlet_vertices(i),:) = dirichlet_values(i);
end
n = size(X,1);
delta_u = inf([n 2]);
while max(max(abs(delta_u))) > tol
    dq_du = eval_dq_du( ...
                tris, X, tri_areas, rho, mu, lambda, u, dt);
    q     = eval_q( ...
                tris, X, tri_areas, rho, mu, lambda, u, b, dt);
    for i = 1:length(dirichlet_vertices)
        a = dirichlet_vertices(i);
        dq_du(2*a-1,:) = 0;
        dq_du(2*a  ,:) = 0;
        dq_du(:,2*a-1) = 0;
        dq_du(:,2*   ) = 0;
```

```
        dq_du(2*a-1,2*a-1) = 1;
        dq_du(2*a  ,2*a  ) = 1;
        q(a,:) = [0 0];
    end
    delta_u = reshape(dq_du \ reshape(-q', [2*n 1]), [2 n])';
    u += delta_u;
end
```

3.3.6. *Example Problem.* We can test the code with the following example problem.

- $\Omega_0 = (-1, 1)^2$
- Dirichlet boundary conditions at $\mathbf{X}_1 = \pm 1$: $u(\mathbf{X}, t) = \sin|\mathbf{X}_1|t$
- Zero Neumann boundary conditions at $\mathbf{X}_2 = \pm 1$
- $\rho_0 \equiv 1$
- $\mathbf{f}^{\text{ext}} = \mathbf{0}$
- $E = 1000$ and $\nu = 0.3$; so $\mu = E/(2(1+\nu)) = 384.$ and $\lambda = E\nu/((1+\nu)(1-2\nu)) = 577.$

LECTURE 3

Techniques to Deal with Element Inversion

Bibliography

1. J. Bonet and R.D. Wood, *Nonlinear continuum mechanics for finite element analysis*, Cambridge University Press, 1997.
2. G. Irving, J. Teran, and R. Fedkiw, *Tetrahedral and hexahedral invertible finite elements*, Graph. Models **68** (2006), no. 2, 66–89.
3. E. Sifakis, J. Hellrung, J. Teran, A. Oliker, and C. Cutting, *Local flaps: a real-time finite element based solution to the plastic surgery defect puzzle.*, Stud Health Technol Inform **142** (2009), 313–8 (eng).
4. J. Teran, E. Sifakis, G. Irving, and R. Fedkiw, *Robust quasistatic finite elements and flesh simulation*, ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA) (2005), 181–190.