Simluation of Elasticity, Biomechanics, and Virtual Surgery Problem Session III

Joseph Teran (UCLA) Jeffrey Hellrung (UCLA)

July 15, 2010

1 Neo-Hookean Elasticity with Quasistatics Evolution in Dimension 1

Recall the elasticity equations from Problem Session II,

$$\rho_0(\mathbf{X})\frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \quad \in \Omega(t) \tag{1a}$$

$$\mathbf{u}(\cdot, t) = \mathbf{g}(\cdot, t) \quad \in \partial\Omega_d(t) \tag{1b}$$

$$(\mathbf{P} \cdot \hat{\mathbf{n}})(\cdot, t) = \mathbf{h}(\cdot, t) \quad \in \partial\Omega_n(t) \tag{1c}$$

and the final non-linear system one must solve if one uses quasistatics for the time evolution in dimension $d = 1$ (and assuming a uniform distribution of mass, i.e., $\rho_0 \equiv 1$):

$$q_i(u) = \int_a^b \frac{\partial N_i}{\partial X} P dX - b_i = 0; \tag{2a}$$

$$b_i = \int_a^b N_i f^{\text{ext}} dX + [\text{Neumann boundary terms}]. \tag{2b}$$

Here, our undeformed domain is $\Omega_0 = (a, b) \subset \mathbb{R}$; N_i is the nodal basis function at grid vertex i; f^{ext} is the external forcing term; and $u = u_i N_i$ is the unknown displacement we wish to determine at the current time-step. 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 $\partial\Omega_n$.

When using the (inversion-robust) Neo-Hookean model, P is related to u via

$$P(u) = \mu F + (\lambda r(F) - \mu)\, r'(F) \tag{3}$$

where $F = \partial u/\partial X + 1$ and

$$r(x + 1) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3, \tag{4a}$$

$$r'(x + 1) = 1 - x + x^2. \tag{4b}$$

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. (As much as it pains me to do so, we'll use 1-indexing to make the translation into Matlab more straightforward.)

We will solve the nonlinear system (2) via Newton iteration:

$$\frac{\partial q}{\partial u}\left(u^k\right)\Delta u + q\left(u^k\right) = 0; \tag{5a}$$

$$u^{k+1} = u^k + \Delta u. \tag{5b}$$

We break the implementation of such a solve into several pieces, outlined below and expounded upon in the subsequence subsections.

- Compute $b = (b_1, \ldots, b_n)$ (from (2b)).
- Implement a procedure to evaluate $q = (q_1, \ldots, q_n)$ (from (2a)).
- Implement a procedure to evaluate $\partial q / \partial u$.
- Solve for the Newton increment Δu.

1.1 Computing b

We begin by considering the computation of 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 b might be a vertex-based approach, where each b_i is computed explicitly, and we consider the boundary grid vertices specially. However, as outlined in the last exercise from Problem Session I, 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

$$\int_{x_i}^{x_{i+1}} N_j f^{\text{ext}} dX \tag{6}$$

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 = construct_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;
    b(i+1) += fext(i) * dX / 2;
end
b(1) += h1;
b(n) += hn;
```

1.2 Evaluating q

We turn now toward evaluating q via (2a) given the current Newton approximation u. Again, we'll use an element-based approach, meaning we need to compute the integrals

$$\int_{x_i}^{x_{i+1}} \frac{\partial N_j}{\partial X} P\left(\frac{\partial u}{\partial X}\right) dX \tag{7}$$

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

$$\frac{\partial N_i}{\partial X} = -\frac{1}{\Delta X} \tag{8}$$

$$\frac{\partial N_{i+1}}{\partial X} = \frac{1}{\Delta X} \tag{9}$$

$$P\left(\frac{\partial u}{\partial X}\right) = P\left(\frac{u_{i+1} - u_i}{x_{i+1} - x_i}\right). \tag{10}$$

This leads to the following procedure to evaluate 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;
    q(i+1) += P;
end
```

1.3 Evaluating $\partial q/\partial u$

The nex step is to consider the evaluation of $\partial q/\partial u$, which will be a sparse $n \times n$ matrix. The integrals we need to compute this time are

$$\int_{x_i}^{x_{i+1}} \frac{\partial N_j}{\partial X}\frac{\partial N_k}{\partial X}\frac{\partial P}{\partial F}\left(\frac{\partial u}{\partial X}\right) dX \tag{11}$$

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 (3):

$$\frac{\partial P}{\partial F} = \mu + \lambda r'(F)^2 + (\lambda r(F) - \mu)\, r''(F). \tag{12}$$

The procedure looks as follows:

```
function dq_du = eval_dq_du(x1, xn, n, mu, lambda, u)
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
```

1.4 Computing the Newton Increment Δu

With the above procedures in hand, computing the Newton increment Δu becomes very easy. There is one thing to note regarding Dirichlet boundary conditions. When computing Δu, we will assume that the current Newton approximation u agrees with its Dirichlet boundary conditions at the current time-step. Hence, we will want zeros in Δu 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_flag1
    dq_du(1,1) = 1;
    dq_du(1,2) = 0;
    dq_du(2,1) = 0;
    q(1) = 0;
end
if dirichlet_flagn
    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;
```

1.5 The Full Newton Method

We now have all the pieces to solve the nonlinear equation (2):

```
function u = solve( ...
    x1, xn, n, ...
    mu, lambda, ...
    dirichlet_flag1, g1, dirichlet_flagn, gn, ...
    h1, hn, ...
    fext, ...
    tol, u)
b = construct_b(x1, xn, n, fext, h1, hn);
if dirichlet_flag1
    u(1) = g1;
end
if dirichlet_flagn
    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_flag1
        dq_du(1,1) = 1;
        dq_du(1,2) = 0;
        dq_du(2,1) = 0;
        q(1) = 0;
    end
    if dirichlet_flagn
        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
```

1.6 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 $u(X, t) = \sin t = g(0, t)$.

Feel free to experiment with other combinations of parameters.