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

Joseph Teran (UCLA) Jeffrey Hellrung (UCLA)

July 16, 2010

1 Neo-Hookean Elasticity with Backward Euler Evolution in Dimension 2

Let us again recall the elasticity equations:

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

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

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

With non-negligible inertia, a more sophisticated time-stepping scheme than quasistatics must be used. One such scheme is Backward Euler, which is desirable due to its unconditional stability. 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.

To formulate the Backward Euler time-stepping, we must introduce an auxiliary variable, $\mathbf{v}$ ("velocity"), to transform eqs (1) into a first-order system:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{v}; \tag{2a}$$

$$\rho_0 \frac{\partial \mathbf{v}}{\partial t} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}}. \tag{2b}$$

The Backward Euler time discretization thus gives

$$\frac{1}{\Delta t} \left(\mathbf{u}(\cdot, t + \Delta t) - \mathbf{u}(\cdot, t) \right) = \mathbf{v}(\cdot, t + \Delta t); \tag{3a}$$

$$\rho_0 \frac{1}{\Delta t} \left(\mathbf{v}(\cdot, t + \Delta t) - \mathbf{v}(\cdot, t) \right) = \left(\nabla^{\mathbf{X}} \cdot \mathbf{P} \right)_{t+\Delta t} + \mathbf{f}^{\text{ext}}(\cdot, t + \Delta t). \tag{3b}$$

Eliminating $\mathbf{v}(\cdot, t + \Delta t)$ from eqs (3) gives a (nonlinear) equation which must be solved for $\mathbf{u}(\cdot, t + \Delta t)$ at each time-step. For present notational purposes, let us refer to the unknown $\mathbf{u}(\cdot, t + \Delta t)$ as simply $\mathbf{u}$, and to the known $\mathbf{u}(\cdot, t)$ and $\mathbf{v}(\cdot, t)$ as $\mathbf{u}_0$ and $\mathbf{v}_0$, respectively. Thus, eqs (3) are equivalent to

$$\rho_0 \mathbf{u} - \Delta t^2 \nabla^{\mathbf{X}} \cdot \mathbf{P} = \rho_0 \left(\mathbf{u}_0 + \Delta t \mathbf{v}_0 \right) + \Delta t^2 \mathbf{f}^{\text{ext}}, \tag{4}$$

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 $t + \Delta t$.

Let us now derive the weak formulation of equation (4) 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 use index notation and implied summation. We will use subscripts to denote coordinates $(1,\ldots,d)$ and (later) superscripts to denote grid vertices.

$$\rho_0 u_i w_i - \Delta t^2 P_{ij,j} w_i = \left(\rho_0\left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) w_i \tag{5a}$$

$$\int_{\Omega_0} \rho_0 u_i w_i - \Delta t^2 P_{ij,j} w_i = \int_{\Omega_0} \left(\rho_0\left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) w_i \tag{5b}$$

$$\int_{\Omega_0} \rho_0 u_i w_i + \Delta t^2 P_{ij} w_{i,j} = \int_{\Omega_0} \left(\rho_0\left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) w_i + \Delta t^2 \int_{\partial\Omega_n} h_i w_i. \tag{5c}$$

For the finite element formulation, 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 (5c), 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:

$$q_i^a(\mathbf{u}) = \left(\int_{\Omega_0} \rho_0 N^a N^b\right) u_i^b + \Delta t^2 \left(\int_{\Omega_0} P_{ij} N_{,j}^a\right) - b_i^a = 0; \tag{6a}$$

$$b_i^a = \int_{\Omega_0} \left(\rho_0\left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) N^a + \Delta t^2 \int_{\partial\Omega_n} h_i N^a. \tag{6b}$$

It is worth noting that this is very similar to the nonlinear system of equations which arise in quasistatics. The differences are only in the right-hand side $\mathbf{b}$ and the additional identity-like block to (6a).

Like in dimension $d = 1$, we shall solve (6) via Newton iteration, which takes the form

$$\frac{\partial \mathbf{q}}{\partial \mathbf{u}}(\mathbf{u})\,\Delta\mathbf{u} + \mathbf{q}(\mathbf{u}) = 0; \tag{7a}$$

$$\mathbf{u} \leftarrow \mathbf{u} + \Delta\mathbf{u}. \tag{7b}$$

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 $t + \Delta t$.

1.1 Evaluating b

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

$$\int_{S^\alpha} \left(\rho_0\left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) N^a. \tag{8}$$

(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

$$
\begin{aligned}
&\int_{S^\alpha} \left(\rho_0\left((u_0)_i + \Delta t (v_0)_i\right) + \Delta t^2 f_i^{\text{ext}}\right) N^a \\
&\quad = \int_{S^\alpha} \left(\rho_0^\alpha \left((u_0)_i^b N^b + \Delta t (v_0)_i^b N^b\right) + \Delta t^2 f_i^{\text{ext},b} N^b\right) N^a \\
&\quad = \int_{S^\alpha} \left(\rho_0^\alpha \left((u_0)_i^b + \Delta t (v_0)_i^b\right) + \Delta t^2 f_i^{\text{ext},b}\right) N^a N^b \\
&\quad = \left(\rho_0^\alpha \left((u_0)_i^b + \Delta t (v_0)_i^b\right) + \Delta t^2 f_i^{\text{ext},b}\right) \int_{S^\alpha} N^a N^b.
\end{aligned} \tag{9}
$$

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

$$
\int_{S^\alpha} N^a N^a = \frac{1}{6} \operatorname{area}(S^\alpha), \tag{10a}
$$

$$
\int_{S^\alpha} N^a N^b = \frac{1}{12} \operatorname{area}(S^\alpha) \quad (a \neq b) \tag{10b}
$$

and we can compute area (S^α) via

$$
\operatorname{area}(S^\alpha) = \frac{1}{2} \begin{vmatrix} \mathbf{X}^{S_1^\alpha} & \mathbf{X}^{S_2^\alpha} & \mathbf{X}^{S_3^\alpha} \\ 1 & 1 & 1 \end{vmatrix}. \tag{11}
$$

In the implementation below, we assume area (S^α) 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
```

1.2 Evaluating q

Evaluation of $\mathbf{q}$ requires computing the integrals

$$\left(\int_{S^\alpha} \rho_0 N^a N^b\right) u_i^b + \Delta t^2 \left(\int_{S^\alpha} P_{ij} N^a_{,j}\right) - b_i^a \tag{12a}$$

$$= \rho_0^\alpha \left(\int_{S^\alpha} N^a N^b\right) u_i^b + \Delta t^2 \left(\int_{S^\alpha} P_{ij} N^a_{,j}\right) - b_i^a, \tag{12b}$$

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 subsection, 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

$$\begin{aligned}\mathbf{P} &= \mu\mathbf{F} + (\lambda r(J) - \mu)\, r'(J) \frac{\partial J}{\partial \mathbf{F}} \\ &= \mu\mathbf{F} + (\lambda r(J) - \mu)\, r'(J) J \mathbf{F}^{-T}\end{aligned} \tag{13a}$$

$$r(x+1) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 \tag{13b}$$

$$r'(x+1) = 1 - x + x^2. \tag{13c}$$

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

$$\begin{pmatrix} \mathbf{X}^{S_1^\alpha} & \mathbf{X}^{S_2^\alpha} & \mathbf{X}^{S_3^\alpha} \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \xi^{S_1^\alpha} \\ \xi^{S_2^\alpha} \\ \xi^{S_3^\alpha} \end{pmatrix} = \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix}. \tag{14}$$

It follows that

$$\mathbf{N}^{S^\alpha}(\mathbf{X}) = \begin{pmatrix} N^{S_1^\alpha}(\mathbf{X}) \\ N^{S_2^\alpha}(\mathbf{X}) \\ N^{S_3^\alpha}(\mathbf{X}) \end{pmatrix} = \begin{pmatrix} \mathbf{X}^{S_1^\alpha} & \mathbf{X}^{S_2^\alpha} & \mathbf{X}^{S_3^\alpha} \\ 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix} \tag{15}$$

and hence

$$\frac{\partial \mathbf{N}^{S^\alpha}}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial N^{S_1^\alpha}}{\partial \mathbf{X}} \\ \frac{\partial N^{S_2^\alpha}}{\partial \mathbf{X}} \\ \frac{\partial N^{S_3^\alpha}}{\partial \mathbf{X}} \end{pmatrix} = \begin{pmatrix} \mathbf{X}^{S_1^\alpha} & \mathbf{X}^{S_2^\alpha} & \mathbf{X}^{S_3^\alpha} \\ 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}. \tag{16}$$

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 $\mathbf{F}$:

$$\mathbf{F} = \frac{\partial \mathbf{u}}{\partial \mathbf{X}} + \mathbf{I} = \mathbf{u}^{S^\alpha} \frac{\partial \mathbf{N}^{S^\alpha}}{\partial \mathbf{X}} + \mathbf{I} \tag{17}$$

Using (17) in (13) 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 evaluating $\mathbf{q}$ 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
```

1.3 Evaluating $\partial\mathbf{q}/\partial\mathbf{u}$

The most complicated computation is evaluating $\partial\mathbf{q}/\partial\mathbf{u}$. To begin, we use (6a) and differentiate with respect to u_j^b:

$$\frac{\partial q_i^a}{\partial u_j^b} = \delta_{ij}\int_{\Omega_0}\rho_0 N^a N^b + \Delta t^2\int_{\Omega_0}\frac{\partial P_{ik}}{\partial F_{\ell m}}\frac{\partial F_{\ell m}}{\partial u_j^b}N^a_{,k}. \tag{18}$$

Given that $\mathbf{F} = \mathbf{u}^{S^\alpha}\left(\partial\mathbf{N}^{S^\alpha}/\partial\mathbf{X}\right)+\mathbf{I}$ (equation (17)), 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

$$\frac{\partial q_i^a}{\partial u_j^b} = \delta_{ij}\int_{\Omega_0}\rho_0 N^a N^b + \Delta t^2\int_{\Omega_0}\frac{\partial P_{ik}}{\partial F_{jm}}N^a_{,k}N^b_{,m}. \tag{19}$$

We can already compute the first integral via (10), so the challenge remains to evaluate $\partial P_{ik}/\partial F_{jm}$. Using (13),

$$\begin{aligned}\frac{\partial P_{ik}}{\partial F_{jm}} &= \delta_{ij}\delta_{km}\mu \\ &\qquad + \left(\lambda r'(J)^2 + \left(\lambda r(J) - \mu\right)r''(J)\right)\frac{\partial J}{\partial F_{ik}}\frac{\partial J}{\partial F_{jm}} \\ &\qquad + \left(\lambda r(J) - \mu\right)r'(J)\frac{\partial^2 J}{\partial F_{ik}\partial F_{jm}}.\end{aligned} \tag{20}$$

This gives all the information necessary to evaluate $\partial\mathbf{q}/\partial\mathbf{u}$, 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 $\partial\mathbf{q}/\partial\mathbf{u}$ 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 $\partial\mathbf{q}/\partial\mathbf{u}$, one needs to "flatten" the right-hand side and then "unflatten" the solution (which will turn out to be $\Delta\mathbf{u}$ 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);
```

1.4 Evaluating the Newton Increment $\Delta\mathbf{u}$

As alluded to in the previous subsection, the stored matrix representing $\partial\mathbf{q}/\partial\mathbf{u}$ 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 illustrated as follows.

```
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])';
```

1.5 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
```

1.6 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 $\lambda = E\nu/((1 + \nu)(1 - 2\nu)) = 577.$