The co-energy for a domain Ω with a given electrical potential ϕ is

$$W^* = \frac{\kappa}{2} \int_\Omega (\nabla\phi)^T (\nabla\phi)\, dx$$

We can take variations with respect to ϕ in order to get a charge distribution; to get a force distribution, we must also take *minus* the variation with respect to changes in the geometry. We would not have this issue if we were using potential with force as our primary variables (or displacement and charge), but since displacement and potential are the conventional choice, we'll stick with that. See also section 6.4.2 in Senturia's *Microsystem Design*.

To compute variations with respect to geometry, we first express the energy integral with respect to a reference geometry:

$$W^* = \frac{\kappa}{2} \int_{\Omega_0} (\nabla\phi)^T (JC^{-1})(\nabla\phi)\, dX$$

where $J = \det(F)$ and $C = F^T F$ are the Jacobian of the mapping from reference to actual and the associated right Cauchy-Green tensor. We need both first and second variations of W^* with respect to the mapping (which we will represent by a displacement vector field u).

Taking first variations (δu), we have

$$\begin{aligned}
\delta(JC^{-1}) &= (\delta J)C^{-1} - JC^{-1}(\delta C)C^{-1} \\
\delta J &= J\,\mathrm{tr}((\delta F)F^{-1}) \\
\delta C &= (\delta F)^T F + F^T(\delta F)
\end{aligned}$$

Taking second variations (Δu), we have

$$\begin{aligned}
\Delta(\delta(JC^{-1})) &= (\Delta\delta J)C^{-1} - JC^{-1}(\Delta\delta C)C^{-1} \\
&\quad -(\delta J)C^{-1}(\Delta C)C^{-1} + JC^{-1}(\delta C)C^{-1}(\Delta C)C^{-1} \\
&\quad -(\Delta J)C^{-1}(\delta C)C^{-1} + JC^{-1}(\Delta C)C^{-1}(\delta C)C^{-1} \\
\Delta\delta J &= J\,\mathrm{tr}((\delta F)F^{-1})\mathrm{tr}((\Delta F)F^{-1}) - J\,\mathrm{tr}((\delta F)F^{-1}(\Delta F)F^{-1}) \\
\Delta\delta C &= (\delta F)^T(\Delta F) + (\Delta F)^T(\delta F)
\end{aligned}$$

If we are linearizing about $u = 0$ ($F = I$), this gives us

$$\begin{aligned}
\delta F &= \frac{\partial(\delta u)}{\partial x} \\
\delta J &= \mathrm{div}(\delta u) \\
\delta C &= 2\nabla^s(\delta u) \\
\delta(JC^{-1}) &= \mathrm{div}(\delta u) - 2\nabla^s\delta u \\
\Delta\delta(JC^{-1}) &= (\mathrm{div}(\delta u)\mathrm{div}(\Delta u) - \mathrm{tr}\,((\delta F)(\Delta F)))\, I \\
&\quad -(\delta F)^T(\Delta F) - (\Delta F)^T(\delta F) \\
&\quad -\mathrm{div}(\delta u)(\Delta C) + (\delta C)(\Delta C) \\
&\quad -\mathrm{div}(\Delta u)(\delta C) + (\Delta C)(\delta C)
\end{aligned}$$

```
inline void localfunc()
{
    /* <generator matexpr>
       //
       // Evaluate variations of C = F'*F, J = det(F), and J*inv(C)
       // at F = I.
       //
       function tr(F) = F(1,1)+F(2,2);
       function dC(dF) = dF + dF';
       function dJCinv(dF) = tr(dF)*eye(2) - dC(dF);
       function DdJCinv(DF,dF) =
           ( tr(DF)*tr(dF) - tr(DF*dF) )*eye(2)
           - ( tr(DF)*dC(dF) + tr(dF)*dC(DF) )
           + ( dC(DF)*dC(dF) + dC(dF)*dC(DF) )
           - ( dF'*DF + DF'*dF );
    */
}


ME::ME(double kappa) :
    Element(3), kappa(kappa)
{
}


ME::~ME()
{
}


void ME::assemble_dR(Mesh* mesh, int eltid, QAssembler* K,
                     double cx, double cv, double ca)
{
    nen = mesh->get_nen(eltid);
    QMatrix1<int,   3,MAXNEN> id;
    QMatrix1<double,3,MAXNEN> nodeu;
    set_local_id(mesh, eltid, id.data);
    set_local_u (mesh, eltid, nodeu.data);
    Quad2d shape(mesh, eltid, nen);
    for (int i = 0; i < nen; ++i) {
        shape.nx(0,i) += nodeu(0,i);
        shape.nx(1,i) += nodeu(1,i);
    }
```

```
    QMatrix<double> Ke(NULL, 3*nen, 3*nen);
    for (Quad2dInt iter(nen); !iter.end(); iter.next()) {
        shape.eval(iter.X());

        // Form grad phi in spatial configuration
        double E[2] = {0,0};
        for (int k = 0; k < nen; ++k)
            for (int j = 0; j < 2; ++j)
                E[j] += shape.dNdx(j,k) * nodeu(2,k);

        double W = cx*kappa*iter.W()*shape.J();
        for (int j = 0; j < nen; ++j) {
            for (int i = 0; i < nen; ++i) {
                int nen3 = 3*nen;
                double* dNdxi = &(shape.dNdx(0,i));
                double* dNdxj = &(shape.dNdx(0,j));
                double* Kij = &(Ke(3*i,3*j));
                /* <generator matexpr>
                   input dNdxi(2), dNdxj(2), E(2), W;
                   DFx = [dNdxi'; 0,0]; DFy = [0,0; dNdxi'];
                   dFx = [dNdxj'; 0,0]; dFy = [0,0; dNdxj'];

                   Kmm = [E'*DdJCinv(DFx,dFx)*E, E'*DdJCinv(DFx,dFx)*E;
                          E'*DdJCinv(DFy,dFx)*E, E'*DdJCinv(DFy,dFx)*E]/2;

                   Kme = [E'*dJCinv(DFx)*dNdxj;
                          E'*dJCinv(DFy)*dNdxj];

                   Kem = [E'*dJCinv(dFx)*dNdxi,
                          E'*dJCinv(dFy)*dNdxi];

                   Kee = dNdxi'*dNdxj;

                   inout Kij[nen3](3,3) += [-Kmm, -Kme; Kem, Kee]*W;
                 */
            }
        }
    }
    K->add(id.data, 3*nen, Ke.data);
}


void ME::assemble_R(Mesh* mesh, int eltid)
{
    nen = mesh->get_nen(eltid);
    QMatrix1<double,3,MAXNEN> nodeu;
```

```
    set_local_u (mesh, eltid, nodeu.data);
    Quad2d shape(mesh, eltid, nen);
    for (int i = 0; i < nen; ++i) {
        shape.nx(0,i) += nodeu(0,i);
        shape.nx(1,i) += nodeu(1,i);
    }

    QMatrix<double> Fe(NULL, 3*nen, 1);
    for (Quad2dInt iter(nen); !iter.end(); iter.next()) {
        shape.eval(iter.X());

        // Form grad phi in spatial configuration
        double E[2] = {0,0};
        for (int k = 0; k < nen; ++k)
            for (int j = 0; j < 2; ++j)
                E[j] += shape.dNdx(j,k) * nodeu(2,k);

        double W = kappa*iter.W()*shape.J();
        for (int i = 0; i < nen; ++i) {
            int nen3 = 3*nen;
            double* dNdxi = &(shape.dNdx(0,i));
            double* Fi = &(Fe(3*i));
            /* <generator matexpr>
               input dNdxi(2), E(2), W;
               dFx = [dNdxi'; 0,0];
               dFy = [0,0; dNdxi'];
               inout Fi(3) += [-E'*dJCinv(dFx)*E/2;
                               -E'*dJCinv(dFy)*E/2;
                               E'*dNdxi]*W;
            */
        }
    }
    add_local_f(mesh, eltid, Fe.data);
}
```