MECH 570C-FSI: Coding Project 2 Fluid-structure interaction with nonlinear hyperleastic structure

Jincong Li
60539939

April 10, 2024

Introduction

The genesis of this project lies in the quest to extend our comprehension and simulation capabilities beyond rigid cylindrical structures interacting with fluid flows, as explored in our preliminary assignment. The focal point shifts towards the modeling and simulation of a flexible hyperelastic structure, such as a flag, subject to the dynamic forces of a surrounding fluid. This progression is not merely academic but a necessary leap towards capturing the real-world complexities encountered in engineering and design. Hyperelastic materials, characterized by their ability to undergo large deformations and return to their original state, necessitate a nonlinear structural analysis to accurately predict their behavior under load.

Methodology

The methodology adopted in this project is a multi-step process beginning with the discretization of the governing structural equations using the finite element method (FEM). This approach enables the approximation of the equations' solutions over a discretized model of the structure. Following the discretization, the nonlinear equations are coded into a MATLAB script, hyperelasticMaterial.m, akin to the format used in our initial exploration of Navier-Stokes equations for fluid flow simulation. Critical to solving the nonlinear structural equations is the Newton-Raphson technique, renowned for its efficiency in handling nonlinear systems through iterative approximation.

Boundary and Initial Conditions

Physical constraints and initial states are simulated by explicitly applying boundary and initial conditions:

```
bcFix = unique(BCBarFix(:));
velS(bcFix,1,1) = solid.DirichletUval;
velS(bcFix,2,1) = solid.DirichletVval;
```

Finite Element Analysis (FEA) Processing

Element-wise calculations form the core of the FEA process, with a focus on assembling stiffness and mass matrices from deformation and strain tensors:

```
    F11  = 1 + locgradXdispXPrev;
    F12  = locgradYdispXPrev;
    F21  = locgradXdispYPrev;
    F22  = 1 + locgradYdispYPrev;

    F11_new  = 1 + locgradXdispX;
    F12_new  = locgradYdispX;
    F21_new  = locgradXdispY;
    F22_new  = 1 + locgradYdispY;

    E11  = 0.5 .* (locgradXdispXPrev + locgradXdispXPrev + locgradXdispXPrev .*
    E12  = 0.5 .* (locgradYdispXPrev + locgradXdispYPrev + locgradXdispXPrev .*
    E21  = 0.5 .* (locgradXdispYPrev + locgradYdispXPrev + locgradYdispXPrev .*
    E22  = 0.5 .* (locgradYdispYPrev + locgradYdispYPrev + locgradYdispXPrev .*

    E11_new  = 0.5 .* (locgradXdispX + locgradXdispX + locgradXdispX .* locgradX
    E12_new  = 0.5 .* (locgradYdispX + locgradXdispY + locgradXdispX .* locgradX
    E21_new  = 0.5 .* (locgradXdispY + locgradYdispX + locgradYdispX .* locgradX
    E22_new  = 0.5 .* (locgradYdispY + locgradYdispY + locgradYdispX .* locgradX

    TraE = 0.5 .* (2 .* locgradXdispXPrev + 2.* locgradYdispYPrev + ...
                       locgradXdispXPrev    .* locgradXdispXPrev + ...
                       locgradXdispYPrev    .* locgradXdispYPrev + ...
                       locgradYdispXPrev    .* locgradYdispXPrev + ...
                       locgradYdispYPrev    .* locgradYdispYPrev);

    TraE_new = 0.5 .* (2 .* locgradXdispX + 2.* locgradYdispY + ...
                           locgradXdispX    .* locgradXdispX + ...
                           locgradXdispY    .* locgradXdispY + ...
                           locgradYdispX    .* locgradYdispX + ...
                           locgradYdispY    .* locgradYdispY);

    S11  = lambda_s .* TraE_new .* F11_new + 2 .* miu_s * (F11_new .* E11_new +
    S12  = lambda_s .* TraE_new .* F12_new + 2 .* miu_s * (F11_new .* E12_new +
    S21  = lambda_s .* TraE_new .* F21_new + 2 .* miu_s * (F21_new .* E11_new +
    S22  = lambda_s .* TraE_new .* F22_new + 2 .* miu_s * (F21_new .* E12_new +
```

Galerkin Terms Assembly

Inertia, convection, and source terms are integrated to capture the material's dynamic behavior:

```
Mij = gW(p) * (N(p,i) * N(p,j));
Mij = Mij .* solid.dens ;
Mij = Mij .* volume;
Aij_1 = ((lambda_s .* (E11 .* dt/2 + E22 .* dt/2 + F11.^2 * dt/2) + ...
2 .* nu_s .* (F11.^2 * dt/2 + F12.^2 * dt/4 + E11 .* dt/2)) .* C1 + ...
(lambda_s .* (F11 .* F12 * dt/2) + ...
2 .* nu_s .* (F11 .* F12 * dt/4 + E21 .* dt/2)) .* C2 + ...
(lambda_s .* (F11 .* F21 * dt/2) + ...
2 .* nu_s .* (F11 .* F21 * dt/2 + F12 .* F22 .* dt/4)) .* C3 + ...
(lambda_s .* (F12 .* F21 * dt/2) + ...
2 .* nu_s .* (F11 .* F22 * dt/4)) .* C4) .* volume;
```

Linear System Solution and Updates

Solving the assembled linear system for nodal displacements and velocities involves iterative updates to accommodate material nonlinearity:

```
Ms = [Ms ZeroF ;...
      ZeroF Ms ];

Ke = [A1 A2;
      A3 A4]

Src = [A5.*solid.gravity(1) ZeroF
       ZeroF A5.*solid.gravity(2)];

Src1 = [A6]*[conn(1*ndof,1)];
Src2 = [A7]*[conn(1*ndof,1)];

MS = (pmc.alphaM/(pmc.gamma*pmc.alpha*solver.dt))*Ms;

% Left-hand side matrix
LHS = Ke + MS;
Increment = LHS(freeNodes,freeNodes) \ RHS(freeNodes);
result(freeNodes) = result(freeNodes) + Increment;
```

Conclusion

The `"hyperelasticMaterial.m"` script was succesfully implemented by linearizing the elasticity equations and the Newton-Raphson technique. However, the debugging process was failed to be accomplished. It could be called from the main function but the output and how the data is transferring between the functions are not fullfilled correctly. Further and more systermetically inspections are required.