Pressure Boundary Conditions

Finite Element Method [304.007]

31.05.2022

Contents

1 Theory

The theory is presented in a learning video on TUbe. The transcript of the video is provided in the TeachCenter. Details of derivations which are not covered in the video are found in the appendix of this sheet in Section 5.
Remark: It is not the goal of this lecture to completely understand the theory of pressure boundary conditions, but to use them.
Furthermore, it is a goal to understand why, in a nonlinear analysis, the external loads may appear in the stiffness matrix.

2 Extension of Existing classes

nsModel.RealIntegrationPoint

- Add the property `pressure`. Initialize the pressure with 0, otherwise the program will not work.
- Add an according set-method for the property `pressure`. This method will be called from the `BCHandler` of a concrete example.

If you don't know how a typical set-method looks like, you can have a look at other classes of the code, e.g. `nsModel.Element`.

nsAnalyzer.nsAnalysis.NonlinearAnalysis

- `calcGlobalStiffnessMatrix`: There is a loop over all elements calculating the stiffness matrix of the incremental internal virtual work and assembling it into the global stiffness matrix. Add a second loop over all faces of the model. In this loop, calculate the stiffness matrix of the incremental external virtual work of each face and assemble it into the global stiffness matrix.
- `calcGlobalLoadVector`: There is a loop over all elements calculating the force vector of the residual internal virtual work. Add a second loop over all faces of the model. In this loop, calculate the force vector of the external virtual work for each face and assemble it into the global force vector.
- In the method `NonlinearAnalysis.run`, change the line

```
1 self.model.bc_handler.incorporateBC();
```

to

```
1 self.model.bc_handler.incorporateBC(time_stamp);
```

if it's not already there.

3 New Classes

nsAnalyzer.nsJacobian.BoundaryJacobian This class is provided in the TeachCenter. It is derived from the class `Jacobian`. Using the boundary Jacobian, we can calculate the tangential vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in the integration points of the faces:

$$\boldsymbol{a} = \frac{\partial \boldsymbol{x}}{\partial \xi} \quad \boldsymbol{b} = \frac{\partial \boldsymbol{x}}{\partial \eta}$$

In the 3D case, those vectors are the first two columns of the Jacobian.
For obtaining the normal vector, we need to calculate the cross product of these two vectors, which is only defined in three dimensions. Therefore, we define the vector $\boldsymbol{b}$ as $\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$ in the 2D case.

nsAnalyzer.nsImplementation.NonlinearBoundaryImpl This class is provided in the TeachCenter. It calculates increment and residuum (i.e. stiffness matrix and force vector) according to the theory. If you want to understand the derivations, have a look at the appendix in Section 5
The class is implemented analogously to
`nsAnalyzer.nsImplementation.NonlinearElementImpl`.

4 Exercise

Simulate a square 2d body which is fixed at the bottom boundary. On the top boundary, a constant pressure is applied.

(a) Create an example folder `platePressure`. You can use the folder `plateAnalysisNonlinear` as a template. Don't forget to rename all files.

(b) Set the Dirichlet boundary condition on the bottom boundary (boundary number 1) in the class `platePressure/MyBCHandler`.

(c) Set the Pressure (i.e. Neumann) boundary condition on the top boundary (boundary number 3) in the class `platePressure/MyBCHandler`. The pressure should become a little larger at each time step. At the end, it should reach its final value $p = 3e4\,\mathrm{N/mm^2}$:

```
press = 3e4*time_stamp.time/self.model.time_bar(end).time;
```

The object `time_stamp` is the second input parameter of the method `incorporateBC`. Where is this function called? Set the pressure boundary using the method `nsModel.RealIntegrationPoint.setPressure` in each integration point of the top boundary.

(d) Don't forget to add the appropriate `NonlinearBoundaryImpl` to the boundary type in the example file `platePressure.m`:

```
mymodel.boundary_type.setImplementation(nsAnalyzer.nsImplementation.
    NonlinearBoundaryImpl());
```

Run the example and visualize the results.

Exercise

1. Play around with the number of time steps (platePressure.m). What do you observe? What is your interpretation?

2. Play around with the maximum value of the pressure. What do you observe here? What happens if you switch the sign (negative pressure)?

3. Bonus: In the provided course material, you find the introductory example `twist`. If you want, you can reproce the results now. Since our code is very slow in 3D, I added results for a large number of elements.

5 Appendix: Derivation of stiffness matrix and force vector for a pressure boundary condition

The following derivations are not part of the lecture, and are only provided for the interested reader. The virtual work induced by the pressure boundary condition reads

$$\begin{aligned} W^p(\chi, \boldsymbol{v}) &= \int\limits_{\partial\Omega^K} p\boldsymbol{v}\cdot\boldsymbol{n}dA \\ &= \int\limits_{\partial\Omega^{\text{ref}}} p\boldsymbol{v}\cdot(\boldsymbol{a}\times\boldsymbol{b})\,dA \\ &= \int\limits_{\partial\Omega^{\text{ref}}} pv^m\varepsilon_{mkl}a^k b^l dA \end{aligned} \tag{1}$$

with the Levi-Civita tensor

$$\varepsilon_{ijk} \coloneqq \begin{cases} 1 & \text{for } (i,j,k)\in\{(1,2,3),(2,3,1),(3,1,2)\} \\ -1 & \text{for } (i,j,k)\in\{(1,3,2),(2,1,3),(3,2,1)\} \\ 0 & \text{else} \end{cases}. \tag{2}$$

The normal vector in the spatial configuration depends on the deformation, i.e. the nonlinear dependence of W^p on χ is introduced via $\boldsymbol{n}(\chi)$. The discretization of the test functions reads

$$v^m \quad\rightarrow\quad \delta_i{}^m N^j\,. \tag{3}$$

You find an explanation of this discretization in the exercise sheet of unit 4, near Equation (6) of that document. Plugging in (3) into (1) gives the discretization of the external virtual work, i.e. the residuum:

$$-W^{\text{p}} \quad\rightarrow\quad F^p_{ij} = -\int\limits_{\partial\Omega^{\text{ref}}} pN^j\varepsilon_{ikl}a^k b^l dA \tag{4}$$

You find the implementation of this formula in the method `nsAnalyzer.nsImplementation.NonlinearBoundaryImpl.pressureForcesIntegrator`. The negative sign is implemented in `NonlinearBoundaryImpl.calcLoad`.
So all that's left is linearizing the virtual work. The only parts of Equation (1) which are not constant are the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. Their respective linearizations and discretizations with shape functions and nodal incremental displacements read

$$\begin{aligned} \boldsymbol{a} &= \frac{\partial\boldsymbol{x}}{\partial\xi} & \boldsymbol{b} &= \frac{\partial\boldsymbol{x}}{\partial\eta} \\ D\boldsymbol{a}[\Delta\boldsymbol{u}] &= \frac{\partial\boldsymbol{\Delta u}}{\partial\xi} & D\boldsymbol{b}[\Delta\boldsymbol{u}] &= \frac{\partial\boldsymbol{\Delta u}}{\partial\eta} \\ (D\boldsymbol{a}[\Delta\boldsymbol{u}])_i &= \Delta\hat{u}_{ij}\frac{\partial N^j}{\partial\xi} = \Delta\hat{u}_{ij}N^j{}_{,1} & (D\boldsymbol{b}[\Delta\boldsymbol{u}])_i &= \Delta\hat{u}_{ij}\frac{\partial N^j}{\partial\eta} = \Delta\hat{u}_{ij}N^j{}_{,2} \end{aligned} \tag{5}$$

The linearization of Equation (1) with subsequent discretization of displacement increment and test function gives us the tangent stiffness matrix for pressure boundary conditions:

$$\begin{aligned} DW^p\left[\Delta\boldsymbol{u}\right] &= \int\limits_{\partial\Omega^{\mathrm{ref}}} pv^o\varepsilon_{omn}\left\{\left(D\boldsymbol{a}\left[\Delta\boldsymbol{u}\right]\right)_m b_n + a_m\left(D\boldsymbol{b}\left[\Delta\boldsymbol{u}\right]\right)_n\right\} dA \\ &\rightarrow \int\limits_{\partial\Omega^{\mathrm{ref}}} pN^j\varepsilon_{imn}\left\{\Delta\hat{u}_{ml}N^l{}_{,1}b_n + a_m\Delta\hat{u}_{nl}N^l{}_{,2}\right\} dA \\ &= \underbrace{\int\limits_{\partial\Omega^{\mathrm{ref}}} pN^j\varepsilon_{imn}\left\{N^l{}_{,1}b_n\delta_{km} + a_mN^l{}_{,2}\delta_{kn}\right\} dA}_{A^p_{ijkl}}\ \Delta\hat{u}_{kl} \end{aligned} \tag{6}$$