Newton-Raphson Analysis in FEM

Finite Element Method [304.007]

25.05.2022

Contents

1 Explanation of existing code 1

2 Extension of existing classes 2

3 New Classes 3

4 Application and Interpretation 4

5 Diagrams 5

5.1 UML Diagramm for nsModel 5

5.2 UML Diagram for nsModel.nsAnalysis 6

You find the theoretical input for this lecture in the TeachCenter (video and transcript). After repeating the theory, download the zip-folder you find under 'Lecture Material > Material - Unit 5' and copy the contents of the directory 'soofeam' into your program directory.

1 Explanation of existing code

nsModel.TimeStamp In this lecture, we solve the quasistatic equilibrium of linear momentum, i.e. we do not consider inertial forces. Time may only appear in changing boundary conditions. The class `nsModel.TimeStamp` is used for handling the time stepping. It is very simple and hopefully needs no further explanation.

nsModel.nsDOF.DisplacementDOF The DOFs (Degrees Of Freedom) of the nonlinear analysis are the *incremental* node displacements. Therefore, this class is enhanced compared to the linear code:

- In addition to the property `displacement`, it has a property `increment`. Like `displacement`, it is initialized as a zero vector in the constructor.

- In analogy to the displacements, the class has the methods `getIncrement` and `setConstraintIncrement`.
- The class has a public method `resetDOF`. This method must put the DOF into its initial state, i.e. the increment vector must be set to zero and the constraint vector must be set to `false`. This method is called at the beginning of each *time step*.
- We need a public method `addIncrement`. This method writes the incremental displacements into the model and updates the overall displacement. It is called once per Newton-Raphson step after solving the linear equation system.

```
function addIncrement( self, coordinate_id, increment )
    self.displacement(coordinate_id) = ...
        self.displacement(coordinate_id) + increment;
    self.increment(coordinate_id) = increment;
end
```

In Section 5.1 you find an UML diagram of the extended class.

2 Extension of existing classes

Modify your code according to the following instructions.

nsModel.Node The node class has to be extended:

- In analogy to the method `setBCDisplacement`, create a method `setBCIncrement`. This method will be called in the `BCHandler` of an example to define the Dirichlet boundary conditions.
- Create the method `updateCoordinates`. It will be called once per Newton iteration. In this method, the `spatial_coordinates` (i.e. coordinates in the deformed state) are brought up to date.

```
function updateCoordinates( self )
    self.spatial_coordinates = self.undeformed_coordinates + ...
        self.dof.displacement;
end
```

This corresponds to updating the deformation mapping via

$$\chi_{k+1} = \chi_k + \Delta \boldsymbol{u}_{k+1} \qquad \text{respective}$$
$$\chi_{k+1} = \boldsymbol{X} + \boldsymbol{u}_{k+1}$$

In Section 5.1 you find an UML diagram of the extended class.

nsModel.BCHandler The base class for handling Boundary conditions must be extended for the nonlinear analysis:

- Create the public method `resetBC`. It will be called at the beginning of each *time step*. This method removes the boundary conditions from the previous time step.

```
function resetBC(self)
    for node = self.model.node_dict
        for coord_id = 1:self.model.dimension
            node.dof.resetDOF(coord_id);
        end
    end
end
```

- Create the public method `setPrescribedDOFZero`. It will be called at the end of the first Newton step. This method sets all Dirichlet boundary conditions to zero, since they are already fulfilled exactly after the first step of the Newton iteration. You find further explanation in the video of this unit.

```
function setPrescribedDOFZero(self)
    for node = self.model.node_dict
        for coord_id = 1:self.model.dimension
            if node.dof.getConstraint(coord_id)
                node.dof.setConstraintIncrement(coord_id, 0.0);
            end
        end
    end
end
```

In section 5.1 you find an UML diagram of the extended class.

3 New Classes

nsAnalyzer.nsAnalysis.NonlinearAnalysis This class performs the nonlinear analysis. You find it in the TeachCenter. In analogy to `LinearAnalysis`, it is derived from the base class `Analysis`. You find an UML diagram in Section 5.2.
The only public method of this class is `run`. It performs the following steps:

- The undeformed state is exported in vtk format
- The loop over the time steps starts
  - All Dirichlet boundary conditions are removed from the model
  - The Dirichlet boundary conditions of the current time step are stored in the model

- The loop over the Newton-Raphson steps starts
  - The linear equation system is created and solved. This method is the same we used for the linear analysis. It is implemented in the base class `nsAnalyzer.nsAnalysis.Analysis`.
  - In the first Newton step, the Dirichlet boundary conditions are set to zero
  - The convergence is checked
- The converged state is exported in vtk format *once per time step*

Exercise 1 Complete the method `nsAnalyzer.nsAnalysis.NonlinearAnalysis.run`. You will find instructions in comments inside the class file. In order to test your implementation, run example `plateAnalysisNonlinear` by executing the main script `soofeam`.

4 Application and Interpretation

For a nonlinear analysis, we need two additional simulation parameters:

- The convergence criterion (also called tolerance): For finding a lower bound (accuracy of Matlab), you can set the tolerance ridiculously low , e.g. `1e-100`. The solver will not converge, and you will see the accuracy of Matlab in the output (command window oder log file).
- The maximum number of Newton steps: A common value is 10. For hyperelasticity (without instability phenomena), quadratic convergence should appear after the first step. Depending on the tolerance, convergence should be achieved after a maximum of 5 steps.

Exercise 2

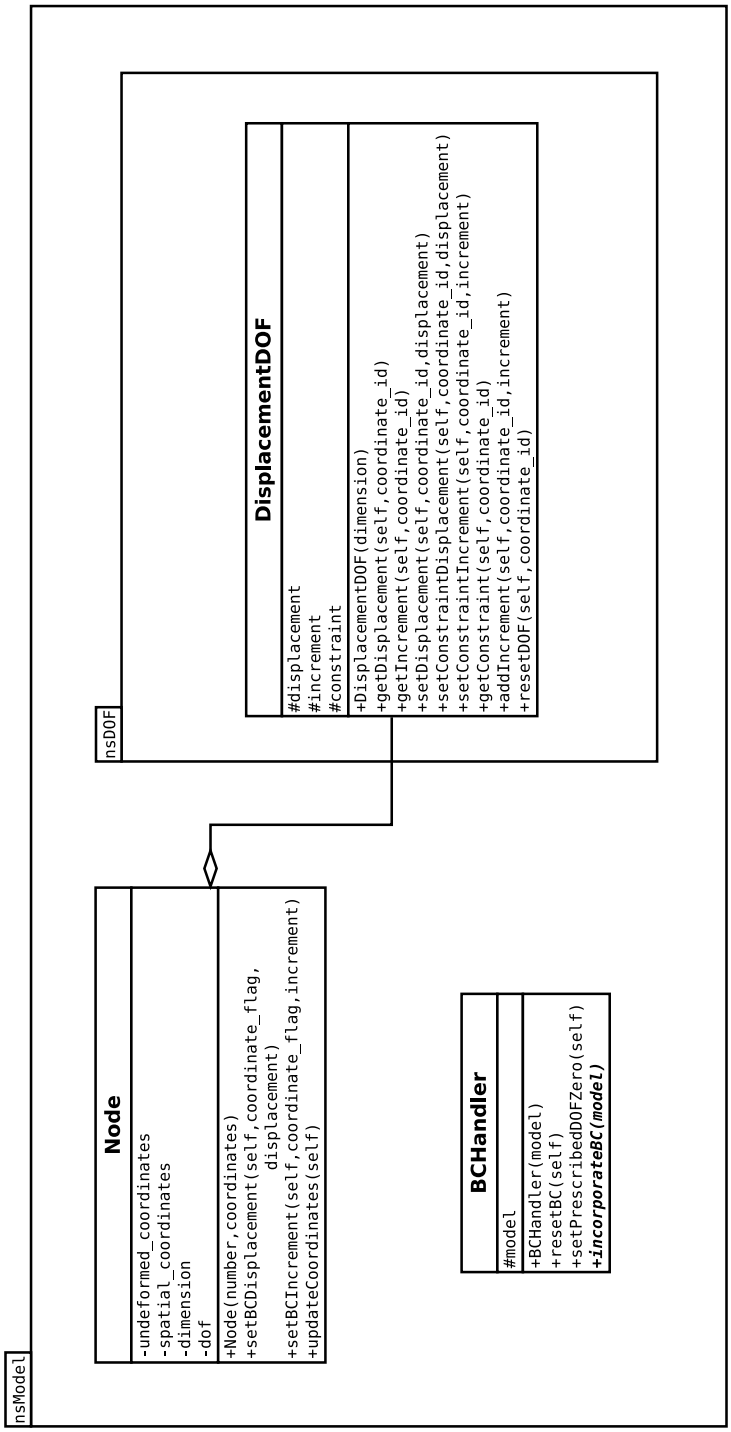
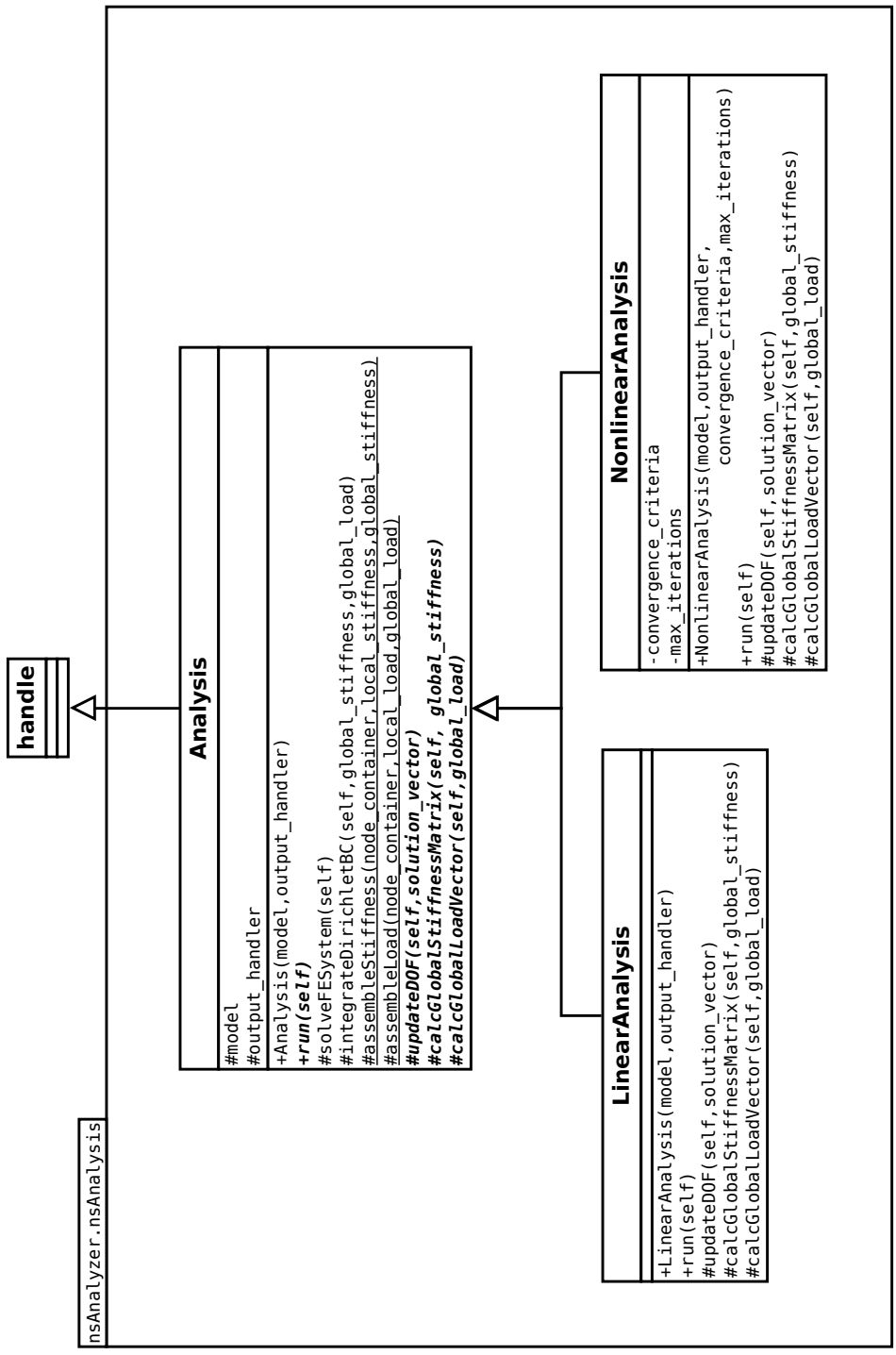
a) Run example `plateAnalysisNonlinear` with a convergence criterion of `conv_crit=1e-100`. You can set the convergence criterion in the file `plateAnalysisNonlinear.m`. What would you define as lower bound?

b) Run example `plateAnalysisNonlinear` with 2x2 elements and a) `conv_crit=1e-3` b) `conv_crit=1e-13`. Analyze the difference with regard to accuracy *and* number of Newton steps per time step. To do this, find the respective horizontal displacement of the top right corner. What is your interpretation?

Exercise 3 Run example `plateAnalysisNonlinear` with `conv_crit=1e-10` and a) 2x2 elements b) 10x10 elements. Analyze the difference with regard to accuracy *and* number of Newton steps per time step. To do this, find the respective horizontal displacement of the top right corner. What is your interpretation?

5 Diagrams

5.1 UML Diagramm for nsModel



5.2 UML Diagram for nsModel.nsAnalysis