CMIS18 - HW5

Isabela Blucher

May 24, 2018

Introduction

This week we introduce the concept of elasticity to our setup for last week's 2-dimensional problem. This means that our mesh still deals with triangular elements, area weighted coordinates for shape functions and point wise boundary conditions. The big difference is the equation that is being used.

The equation used this week is the Cauchy momentum equation, which is a vector partial derivative equation that describes the non-relativistic momentum transport in any continuum. The formula is shown below

$$\int_{\Omega^e} (\delta\varepsilon)^T \sigma \, d\Omega = \int_{\Gamma^e} (\delta u)^T f \, d\Gamma$$

Where σ is the Cauchy stress tensor, $\delta\varepsilon$ is the virtual Cauchy strain tensor, δu^e is the virtual displacement vector, f is the external body force, Ω^e is the element-wise domain and Γ^e is the element-wise boundary. We start from this equation to derive formulas for K^e and f^e and later implement the 2D elastic assembly process.

Derivation of the formulas

To start deriving formulas we have to define substitutions for the terms in the equation. For the virtual displacement vectors we can use the shape functions and then we have $\delta u = N^e \delta u^e$, with N begin a 2×6 matrix with the barycentric coordinates introduced last week. For the strain tensor, since strain is a function of the displacement field $\delta\varepsilon = S\delta u = SN^e\delta u^e = B\delta u^e$, where S is the differential operator being of dimensions 3×2 and B is of dimensions 3×6. And finally, we can relate the stress and strain tensors by the linear relation $\sigma = D\varepsilon = DBu^e$ where D is a 3×3 elasticity matrix.

Since the given formula is already in the weak form, there is no need to integrate by parts, which means that the next step is plugging in our approximations in the equation.

$$\begin{aligned}
\int_{\Omega^e} (B\delta u^e)^T DBu^e \, d\Omega &= \int_{\Gamma^e} (N^e \delta u^e)^T f \, d\Gamma \\
\int_{\Omega^e} (\delta u^e)^T B^T DBu^e \, d\Omega &= \int_{\Gamma^e} (\delta u^e)^T (N^e)^T f \, d\Gamma \\
(\delta u^e)^T \int_{\Omega^e} B^T DBu^e \, d\Omega &= (\delta u^e)^T \int_{\Gamma^e} (N^e)^T f \, d\Gamma \\
B^T DBu^e \int_{\Omega^e} d\Omega &= f \int_{\Gamma^e} (N^e)^T \, d\Gamma \\
B^T DBA^e u^e &= f \int_{\Gamma^e} (N^e)^T \, d\Gamma
\end{aligned}$$

Looking at the left hand-side of the equation we have the formula for $K^e = B^T DBA^e$. Looking at the right-hand side we see that we need the area for the triangle in the boundary. We compute the length of the triangle edge that is part of the boundary as L^e, which completes the formula for f^e. Let's keep in mind that f is a 2×1 vector that describes force.

$$f^e = f \int_{\Gamma^e} (N^e)^T \, d\Gamma = \frac{1}{2} L^e [I_{2\times 2} \, I_{2\times 2}]^T f$$

With that, we have the same system of equations $K^e u^e = f^e$, which can be implemented and solved computationally.

Implementation

A template was given in MATLAB such as the past two weeks. It was necessary to complete the assembly process for K^e, K, f to complete the 2D elasticity example. Figure 1 shows results based on our FEM derivation of the formulas for a rectangular solid bar of 6×2 meters made of steel like material. The left side of the bar is attached to a wall and a downward nodal load is applied to the right side.

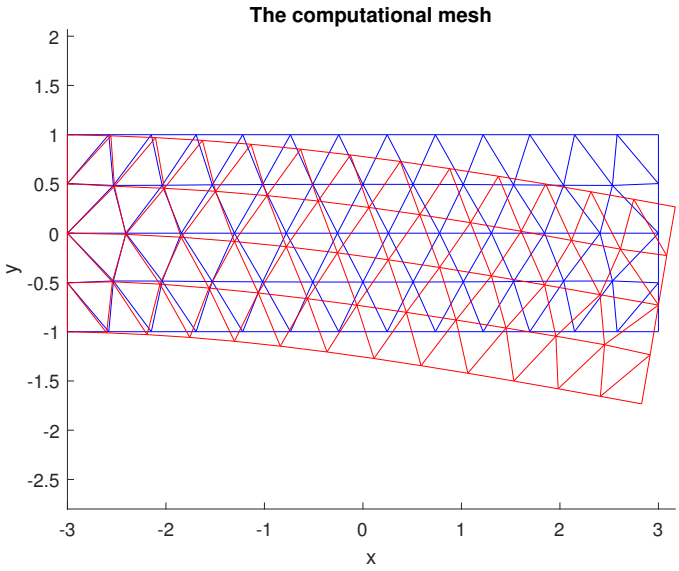


Figure 1: Steel bar with downward nodal load of 10^7

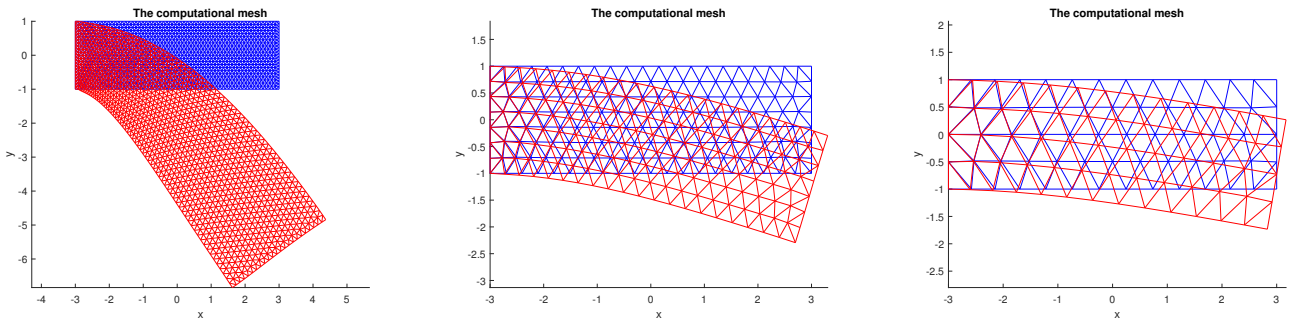
As we can see the blue mesh shows the original bar and the red shows the bar after the application of the load and consequent displacement.

The first version of the implementation uses a "cheat" as a formula for f^e, as it sets a "constant" load per node value instead of dealing with the formula derived from the integral to assemble f^e and correctly update f. For some of the experiments done in the next section we will deal with testing in both implementations, and thus will refer to the "cheat" implementation as **version 1** and the corrected one as **version 2**.

Experiments

Different meshes

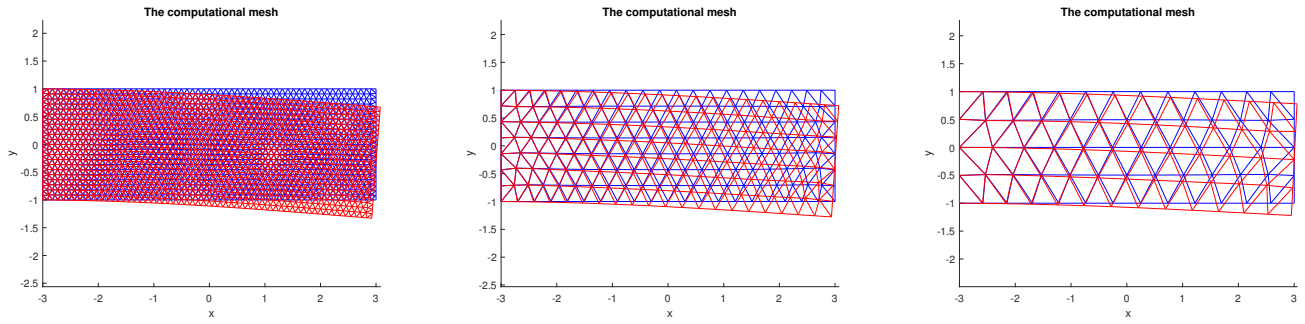
For this experiment our objective is to see how both versions of the implementation behave for different mesh resolutions and the same force load applied. The expected behavior is that for the same force applied, the same displacement should be seen, even for different resolutions. To test this hypothesis we run both versions of the algorithm on the original given mesh (`'data.mat'`) and 5 other triangular meshes created with `distmesh2d` that have different resolutions, and compare the final y coordinate of the lower right corner of the displaced mesh.



(a) Initial edge length of 0.1 (b) Initial edge length of 0.3 (c) Initial edge length of 0.5

Figure 2: Result of version 1 on DistMesh meshes with different resolutions and a downward nodal load of 10^7

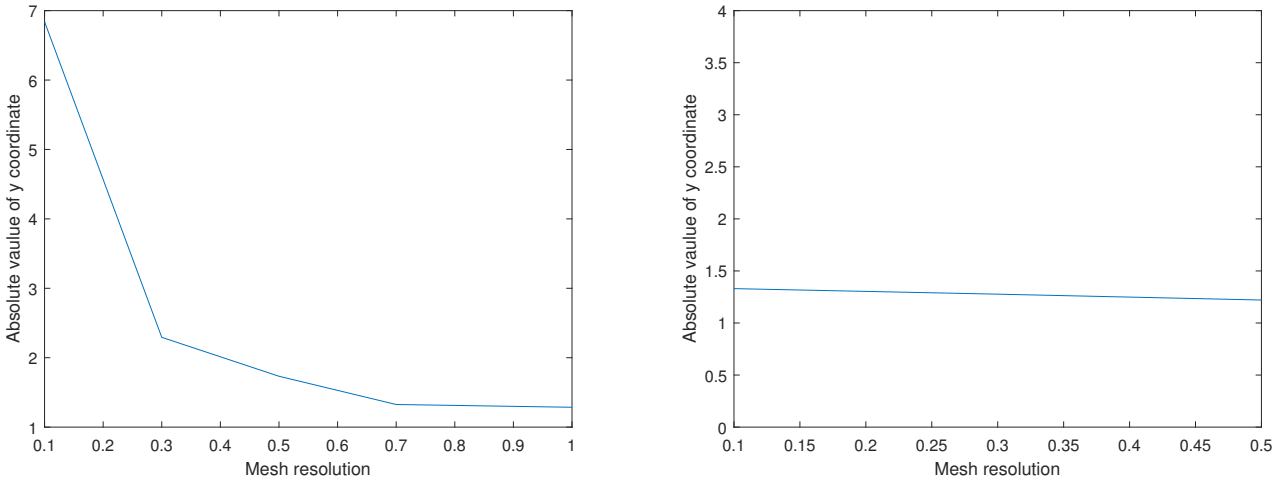
For the version 1, we can see that the hypothesis is not confirmed because the displacement for different meshes and the same amount of applied force is very different. Doing the same tests with version 2 we get the plots in Figure 3.



(a) Initial edge length of 0.1 (b) Initial edge length of 0.3 (c) Initial edge length of 0.5

Figure 3: Result of version 2 on DistMesh meshes with different resolutions and a downward nodal load of 10^7

For version 2, we can see that the hypothesis is confirmed because using the same force and different mesh resolutions we have the same displacement. It is possible to observe these changes in a graph that relates initial length of the mesh to the y coordinate of the lower right corner of the mesh, to see how the displacement behaves in both cases.



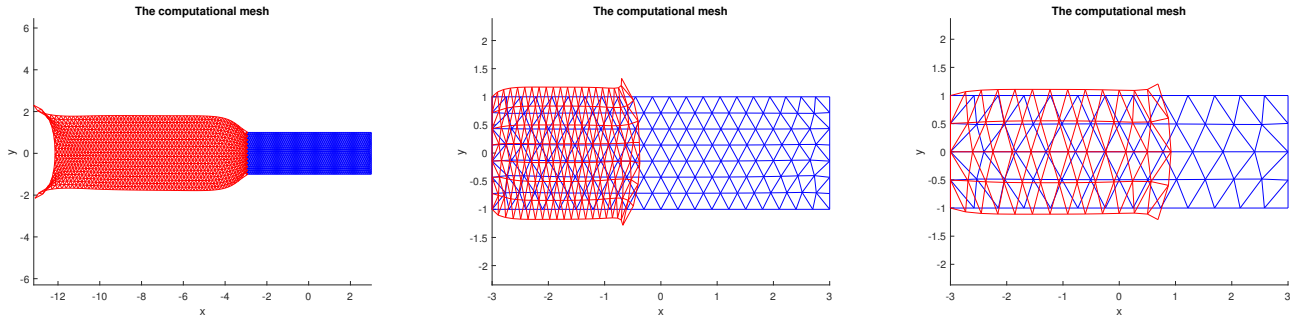
(a) Version 1 (b) Version 2

Figure 4: Relation of mesh displacement to mesh resolution

By visual inspection of Figure 4 we can see that for version 1, the smaller the mesh resolution the bigger the displacement. This happens due to the "cheating" implementation. It considers that since there are more elements, there is more force being applied and thus the displacement should be bigger. On the contrary, with version 2 and the correct implementation of f^e and f, the mesh resolution is taken into account when applying the force, which results in a line that is practically constant.

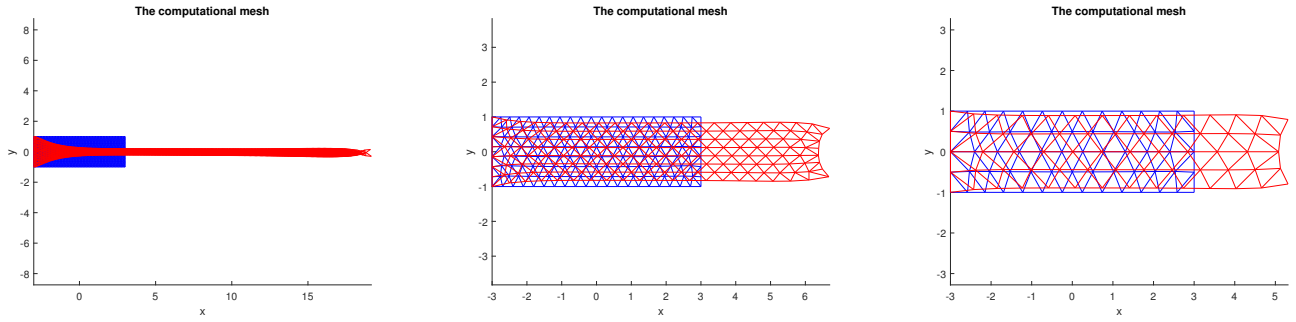
Different directions

We can also try and observe how the displacement behaves for loads applied on different directions instead of only downward. By repeating the experiment in the section above, but applying the same force of 10^7 in the horizontal, both to the left and right, for different meshes and for the first version of the algorithm we get the Figures 5 and 6.



(a) Initial edge length of 0.1 (b) Initial edge length of 0.3 (c) Initial edge length of 0.5

Figure 5: Result of version 1 on DistMesh meshes with different resolutions and a horizontal nodal load to the left of 10^9



(a) Initial edge length of 0.1 (b) Initial edge length of 0.3 (c) Initial edge length of 0.5

Figure 6: Result of version 1 on DistMesh meshes with different resolutions and a horizontal nodal load to the right of 10^9

As expected, the same applied force for different mesh resolutions has a very different displacement result, either for the left or the right. When stretching it to the left, the bar seems to dilate vertically, and when stretching it to the right, the bar seems to shrink inwards.

From visual inspection of the Figures 2, 5 and 6 we can also see that there is no area conservation from the original bar to the displaced one using the version 1 of the implementation. This can be explained due to the infinitesimal strain theory, as small changes will remain unnoticed, bigger loads will cause very large deformations in the bar.