6.8420: Computational Design and Fabrication
Charlotte Folinus
Massachusetts Institute of Technology
Spring 2024

Homework 2: Finite Element Methods

Section 3.1 FEM with linear material model

Problem 1.1: Compute $\frac{\partial \mathbf{P}(\mathbf{F})}{\partial \mathbf{F}}$ for linear material
This section did not ask any specific questions.

Problem 1.2: Compute stiffness matrix $\mathbf{K}$
This section did not ask any specific questions.

Section 3.2: Nonlinear material

Problem 2.1: Implement Newton's method in the `solve_newton` function
This section did not ask any specific questions.

Problem 2.2: Compare between linear and nonlinear models
How "good" the linear model is depends on the geometry of the beam (aspect ratio), the magnitude of the applied load(s), and the meshing used:

- For a given load and beam geometry, increasing the mesh resolution can somewhat improve the accuracy of the linear model without the large computation time required for the nonlinear model. Although a coarsely-meshed nonlinear model can achieve a similar result to a finely-meshed linear model, it takes much longer to compute the nonlinear model; depending on the aspect ratio of the beam and the forces being applied, it may make more sense to use a rather finely-meshed linear model instead of the nonlinear model.

- For a given beam geometry, increasing the magnitude of the load increases the error associated with the linear analysis. For the linear analysis, higher loads (400) show significant distortion in overall size of the deformed beam.

- Beams which are more slender (or made of softer materials) will also be more significantly impacted by the linear vs. nonlinear modeling (they will show relatively greater error when modeled using the linear analysis). Even with a finer mesh (16x4x4 vs 8x2x2) and low load (50), the linear analysis produces significant shape distortion (the deformed beam is much larger than is physically possible).

- I did not quantitatively compare the run time for each analysis case, but a good practice (before a more complex analysis or before running an optimization)

would be to run a nonlinear analysis with a few mesh resolutions, run a linear analysis with a few mesh resolutions, time each analysis, and compare the error between the two results. Sometimes (at small enough loads and thick enough beams/geometries), you can achieve satisfactory accuracy by running a linear analysis with a fine mesh (instead of running a nonlinear analysis with a coarser mesh). Relative to running the nonlinear analysis, this could save significant computation time. In other cases, it will be necessary to run a nonlinear analysis (perhaps even a nonlinear analysis with a high-resolution mesh!) — this will all be context-dependent (based on your desired geometry and loads), so it is good practice to compare the accuracy benefits of mesh refinement and nonlinear modeling with the associated time costs.

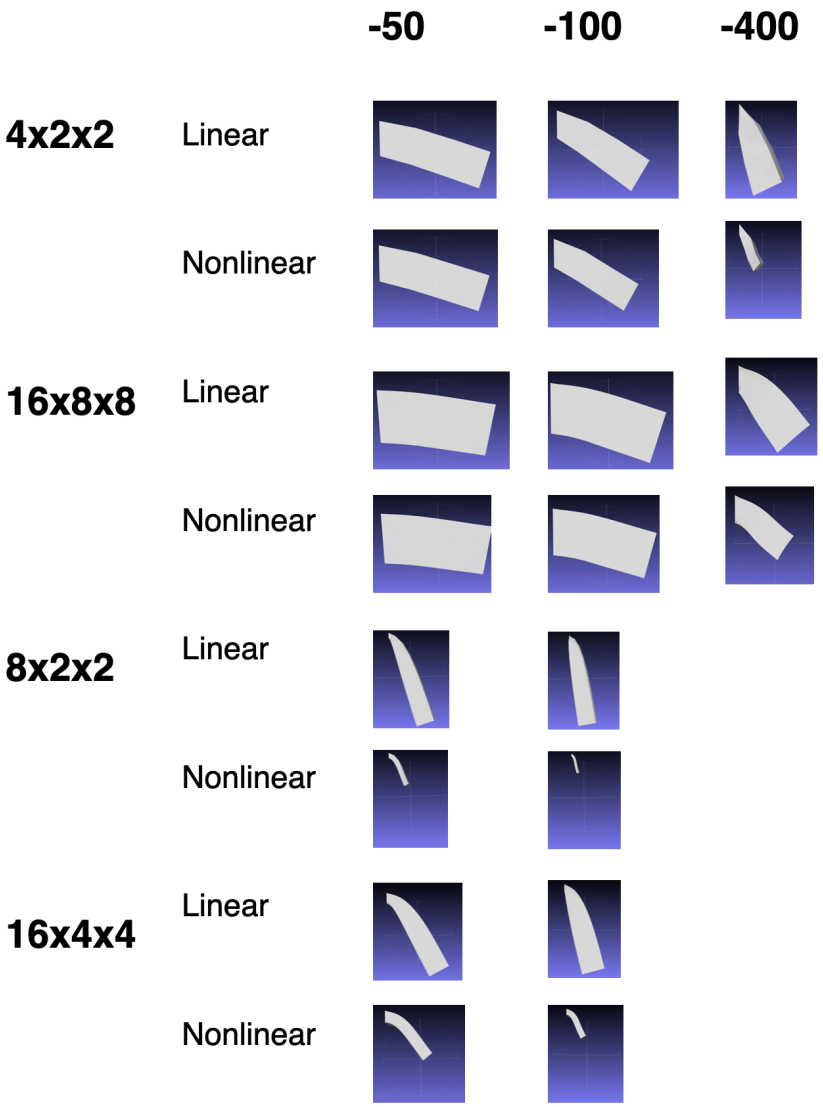

Figure 1: Summary of FEA cases run.

Section 3.3: Custom boundary condition

Problem 3.1: Implement your customized boundary conditions for `spot`

Fixed boundary conditions: I fully constrained the (vertically) lower vertices on the object (`spot`), which roughly corresponds to the legs of the cow shape. I did this in `boundary_conditions_custom()` in `main.py` using:

```
bc = V[:,1] < 0.045
```

Applied loads: I applied loads in the $< +x, +y, +z >$ direction. I applied loads to the (vertically) upper vertices, which roughly corresponded to the ears. I did this in `boundary_conditions_custom()` in `main.py` using:

```
f_ext_mask = V[:,1] > 0.225
f_ext = np.zeros_like(V)
f_ext[f_ext_mask] = [20, 20, -20]
```

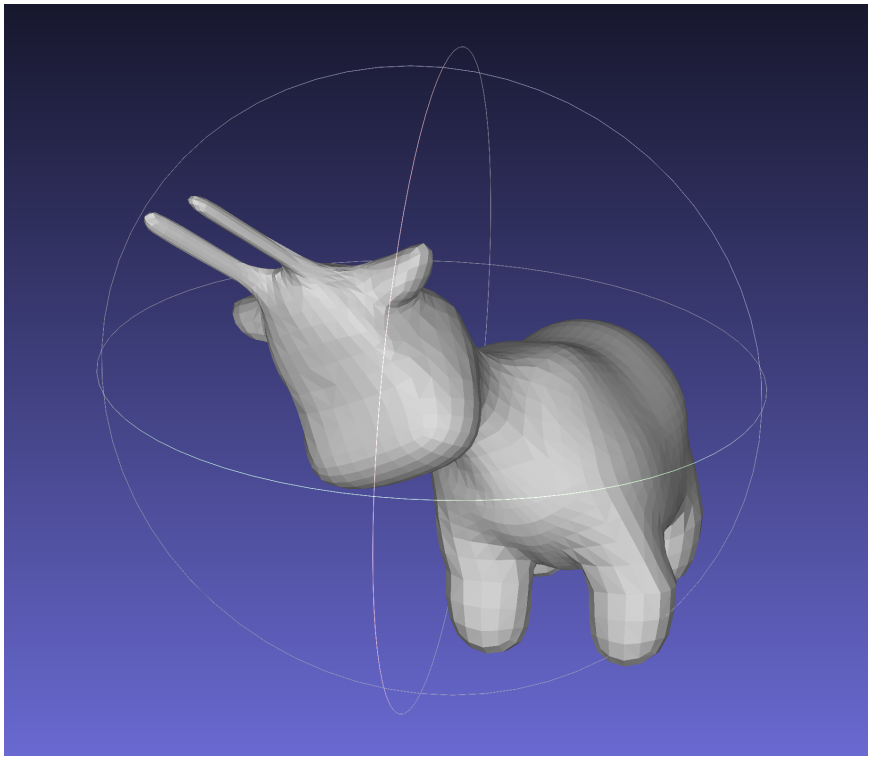

Figure 2: FEA-deformed `spot` model.

Section 3.4: Surrogate simulation using neural network

For each of the following application domains, specify good choices for the number of units in the output layer, the activation function(s) on the output layer, and the loss function.

Problem 3.1: Map the words on the front page of the New York Times to the predicted (numerical) change in the stock market average.

Output unit: This scenario is a regression problem where we are trying to predict a continuous value (the change in the stock market average). This value could correspond to one output neuron (unit) which would predict the numerical change in the stock market average.

Activation function: Linear. We don't need/want to compress our outputs to a specific range, and they can be both positive- and negative-valued.

Loss function: This model will be doing regression for a continuous, numerical value. Mean squared error (MSE) would be a good fit. Its quadratic nature will penalize large errors more heavily than small errors, which will help it push the model toward progressively smaller errors. With MSE, our training goal becomes minimizing the squared differences between predicted and actual values, which corresponds well to our regression goals.

Problem 3.2: Map a satellite image centered on a particular location to a value that can be interpreted as the probability it will rain at that location sometime in the next 4 hours.

Output unit: We are trying to predict the probability of rain at a single location, so our output could be a single unit (a single value), which corresponds to this probability.

Activation function: This problem focuses on predicting probabilities, and the probability describes the likelihood of a binary task (rain or no rain). A sigmoid function could be an appropriate choice, as it will squash the output into the $(0, 1)$ range, which is the desired range for our probability prediction task. In this case, the output of the sigmoid function would be the probability of rain.

Loss function: Since this task is focused on probability, a cross-entropy loss (negative log likelihood) could be a good fit. This loss function will measure the difference between the two probability distributions (the true distribution and the predicted distribution).

Problem 3.3: Map the words in an email message to which one of a user's fixed set of email folders it should be filed in.

Output unit: This is a multi-class classification problem, as we are trying to map words in an email to one of many potential email folders. It would make sense to have as many output units as we do email folders in our fixed set of email folders. In this case, each email folder corresponds to a class. Each unit in our output layer corresponds to the probability that the input message belongs to that folder.

Activation function: Softmax activation functions are commonly used for multi-class classification problems. Softmax squashes the output into the $(0, 1)$ range, which makes sense for probabilities, and it can ensure that the sum of all output probabilities across the multiple classes is equal to 1. Linear and ReLU activation functions are not bounded between 0 and 1, which would make them less useful here. With a sigmoid activation function, we would be treating each class prediction problem as a binary classification problem, which would not make the most sense for a multi-class classification task. We would have a softmax layer with the same number of nodes as the output layer.

Loss function: A (categorical) cross-entropy loss (negative log likelihood) would make sense for this multi-class classification problem as it would compare the probability distributions between the true distribution (represented by one-hot encoded labels for the folders) and the predicted distribution (the output probabilities from the Softmax function). This function would encourage a high probability for the correct label and heavily penalize the wrong predictions.

Problem 3.4: Map the words of a document into a vector of outputs, where each index represents a topic, and has value 1 if the document addresses that topic and 0 otherwise. Each document may contain multiple topics, so in the training data, the output vectors may have multiple 1 values.

Output unit: In this problem, each index in the output vector is a topic, and we want to predict whether the document contains each topic. It would make sense to have one output unit for each topic, giving us as many output units as we have topics. Each output unit will be a label corresponding to whether the document contains that topic.

Activation function: We could use a sigmoid function, which will squash our outputs into the $(0, 1)$ range. This makes sense for classification problems like this one (binary classification). If we had instead used a softmax function (which also squashes between 0 and 1), we could end up getting mutually-exclusive topics. In this problem, we want to be able to assign multiple binary labels to a given input document, so this would not make sense. The sigmoid layer would have the same number of nodes as the output layer, and the sigmoid function would be applied to each unit/node indepdently, since the likelihoods for each topic are independent of each other.

Loss function: It would make sense to use a binary cross-entropy loss, which would be appropriate for measuring the difference between probability distributions. In this case, each output unit corresponds to an individual (and independent!) binary classification task. We would compute the binary cross-entropy of each of the sigmoid outputs and sum them together for a combined loss score.

We will now be focusing on using neural network to learn a surrogate model for physical simulation. Run the cells on 2 Learning Simulation from the Google Colab notebook and answer the following questions:

Problem 3.5: We could also consider some other metrics. Which ones might you consider, and why?

We used a mean squared error (MSE, L2 norm) loss function, but we alternatively could have used a mean absolute error (MAE, L1 loss). Using L1 loss would treat all errors equally without consideration of their magnitude (the errors don't get squared, so large errors are not penalized relatively more than small ones). MAE might make more sense than MSE if we have outliers that we either don't want to penalize or don't want to heavily penalize. We could also use a Huber Loss function (smooth mean absolute error), which would have a quadratic penalization for small errors and a linear component for large errors. This gives takes advantage of the benefits of both the MSE and MAE approaches.

Problem 3.6: How large is your error? Is this a good error, or a bad error, in your opinion? Play around with different neural network architectures based on what you have learned in class. What is the best performance you can achieve and what are the modifications you performed?

Neural network training and evaluation is stochastic, which means it's helpful to train the model multiple times for a better estimate of how "good" the error is. I trained the default architecture) three times with a maximum of 2e4 iterations, yielding training losses of 0.0531, 0.0420, and 0.0601. These corresponded to test losses of 0.0855, 0.0698, and 0.0955. This error is pretty small (good!), but the test losses are significantly larger (50-75%) higher than the training losses. This indicates that we might be overfitting the training data. It would be advantageous to have a different model that both achieves a lower training and testing score than this default model.

I also made a few modifications to explore how changing the architecture slightly impacted both the training and testing loss scores:

1. Default: `inp_features` → 64 units → 64 units → 2 (output features)
   - Training loss: 0.0531, 0.0420, 0.0601
   - Testing loss: 0.0855, 0.0698, 0.0955.
2. `inp_features` → 64 units → 64 units → 64 units → 32 units → 2 (output features)
   - Training loss: 0.0806, 0.1097, 0.0838
   - Testing loss: 0.1152, 0.1478, 0.1500
   - Not better than default! The testing/training losses are relatively closer to each other, so our model is likely overfitting the data less, but it's still performing less well overall.
3. `inp_features` → 64 units → 64 units → 64 units → 2 (output features)
   - Training loss: 0.0567, 0.0395, 0.0457
   - Testing loss: 0.0862, 0.0585, 0.1151
4. `inp_features` → 64 units → 64 units → 64 units → 64 units → 2 (output features)
   - Training loss: 0.0442, 0.0528, 0.0384
   - Testing loss: 0.0980, 0.0527, 0.0704

5. `inp_features` → 48 units → 48 units → 48 units → 48 units → 2 (output features)
    - Training loss: 0.0482, 0.0389, 0.0387
    - Testing loss: 0.0523, 0.0729, 0.0823
    - This model performs pretty similarly to model 4 (which has the same overall architecture but larger layers). Because it has fewer parameters, it's less likely to be overfitting the training data, which means it could better generalize to data outside of the testing/training data sets.

Section 4.1: Writeup

Code description: Based on the structure of this assignment, the majority of the code was already implemented. I'll focus my writeup on things that were particularly interesting/difficult for me in the implementation.
Computing $\frac{\partial \mathbf{P(F)}}{\partial \mathbf{F}}$ was a little tricky at first, because $\frac{\partial \mathbf{F^T}}{\partial \mathbf{F}}$ is the derivative of a second-order tensor (matrix) with respect to another second-order tensor (matrix). Strictly speaking, this yields a fourth-order tensor (with size pxqxmxn). It's tricky to work with fourth-order tensors (and the code provided needed this in a 2D format), so we needed to get the pxqxmxn tensor into a pqxmn format.

For computing $\frac{\partial \mathbf{F^T}}{\partial \mathbf{F}}$, our goal is to have a 2D matrix with this format:

$$\begin{bmatrix} \mathbf{D_{11}^{11}} & \mathbf{D_{11}^{12}} & \cdots & \mathbf{D_{11}^{mn}} \\ \mathbf{D_{12}^{11}} & \mathbf{D_{12}^{12}} & \cdots & \mathbf{D_{12}^{mn}} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{D_{pq}^{11}} & \mathbf{D_{pq}^{12}} & \cdots & \mathbf{D_{pq}^{1}} \end{bmatrix}_{[p\times q]\times[m\times n]}.$$

All values are zero except for: $\mathbf{D_{11}^{11}}, \mathbf{D_{12}^{21}}, \mathbf{D_{13}^{31}}, \mathbf{D_{21}^{12}}, \mathbf{D_{22}^{22}}, \mathbf{D_{23}^{32}}, \mathbf{D_{31}^{13}}, \mathbf{D_{32}^{23}}$, and$\mathbf{D_{33}^{33}}$ (or, we could write this as $\mathbf{D_{ji}^{ij}}$ for $i, j = 1, 2, 3$).

Setting up the `index_map` was also a little tricky. This vector maps the rows/columns of K_t to the rows/columns in K that correspond to the same degrees of freedom. An individual tet element has four nodes with three DOF per node (the stiffness matrix is 12x12, with the DOFs for each element placed one after the other). Finding the correct index in the global stiffness matrix (K) required going node-by-node in the element (looping over the simplex). For each node, I looked up the corresponding vertex number, which I used to find the starting index for that DOFs nodes (`dim * T[t][i]`). I added this to the current degree of freedom (`j`) to get the value for that DOF, which was stored at index `dim*i + j` within the `index_map`.

Configuring the newton solver was mostly straightforward, with much of the math given to us in the comments already (things like `U = Ui + dU * l`. It was important to correctly calculate the elastic forces — in practice, this was pretty easy, because we already had a helper function to do this! (At first, forgetting this helper function, I re-implemented most of `elastic_force()` function.) Setting the exit condition for line search was also important to ensure that we exited at the correct time. We wanted the line search to end when the residual forces at U were smaller than those computed at our next guess U_i — this required setting the loop to `break` when `f_res_l_norm < f_res_norm`.

What did you like? I liked that we were given a file we could run to see if our virtual environment was set up correctly. I also liked that some of the questions gave us freedom to poke around with settings and gain intuition on our own.

What did you not like? It was confusing to me that the test run of the program (`main.py`) failed when it was initially run (before implementing the solution). The

homework instructions said we should be fine if there was no error, but the program would throw an error from the CG solver. The instructions could be reworded to say that there should only be CG solver errors (or something along those lines). It was also confusing that the Google Colab notebook had different questions/content than what we were expected to do/ask/answer for this homework assignment.

Any other issues Nope, all good!