Homework 2: Finite Element Methods

6.4420

Computational Design and Fabrication
Assigned:03/06/2024
Due: **03/20/2024 at 11:59 PM**

Please update your codebase by executing following command in your codebase folder:

```
git add *
git commit -m "My assignment 1 code."
git pull
```

No setup is needed for this assignment if you continue with the virtual environment in previous assignments. Once you have extracted the files from the package, simply do a test run of the program by

```
cd PATH_TO_YOUR_FOLDER/compfab-hw2-24s-skeleton
python assignment2/main.py
```

The program will simply iterate over all test cases for now. Nonetheless, you should be all set if there is no error.

1 Introduction

In this assignment, we'll implement the finite element method (FEM) for linear material and nonlinear material, and test the FEM solver on a few tetrahedral meshes, as well as learning how to use neural networks to learn a surrogate simulation model.

2 Background

2.1 The High Level

Before jumping into the specifics of the assignment, let's give a quick, high-level overview of FEM.

Associated with any soft object are two aspects: material and geometry. Materials include aspects like material density, Young's Modulus (a measurement of stiffness), and Poisson Ratio (a measurement of compressability). When uniform, density, and in fact mass in general, only affects the dynamics of the system, so we will ignore it in this assignment.

Geometry, meanwhile, encodes a structure to the problem. Surface meshes are translated into volumetric meshes (*e.g* tetrahedral and hexahedral). Each element is affected by the positions of its nodes - and thus by any touching elements. As we'll see, stiffnesses and elastic forces are calculated by summing over all of the independent elements and adding their contributions to the appropriate nodal degrees of freedom. For gradients of vector values (Jacobians), we will see that the sparsity pattern, and where the individual contributions come from, is intimately tied to the mesh structure and choice of finite element type.

Although materials and geometry both go into the final formulation of the static simulation, they have discrete meaning. Materials describe the *intrinsic* local continuum mechanics of bulk objects at any given point on the object, while geometry and meshing describe how this bulk material is translated into different local stiffnesses throughout the object.

Beyond these, there are two other elements specific to the *environment* that need to be considered. The first is boundary conditions. Which aspects of an object (degrees of freedom) are fixed to the world, unable to move? Any parts of an object that are fixed will not deform. The second is external (loading) forces. Forces, such as those caused by gravity or, say, placing an external weight on the object, are necessary for the object to experience *any* deformation, and intuitively have the most "say" in how the object deforms. For instance, it should be intuitive that if you drop a marble on a vat of jello, the jello will locally sink downward below the marble - not to the right or left, or forward or back - and it certainly won't advect upward.

All of these elements come together to form a system with n vertices (nodes):

$$\mathbf{KU} = \mathbf{f} \tag{1}$$

where $\mathbf{K} \in \mathbb{R}^{3n \times 3n}$, $\mathbf{U} \in \mathbb{R}^{3n}$, $\mathbf{f} \in \mathbb{R}^{3n}$ (here the 3 comes from the fact that, in 3D, the mesh will have 3 degrees of freedom per node; in 2D, the mesh would have 2 degrees of freedom per node; in other dimensions it translates accordingly). Here, the $\mathbf{K}$ depends on materials and the mesh structure, $\mathbf{f}$ depends on the external loads, and $\mathbf{U}$, for linear systems, denotes how much each degree of freedom has moved from its resting (unloaded) position when in static equilibrium. Rows and columns corresponding to fixed degrees of freedom are removed from the system prior to solve. At the end of the day, if we can formulate $\mathbf{K}$ and $\mathbf{f}$, we can solve for $\mathbf{U}$ and calculate the final mesh vertex positions $\mathbf{q} = \mathbf{q}_{\text{rest}} + \mathbf{U}$.

2.2 The Slightly Lower Level

Let's get a bit more technical. Recall Newtonian mechanics - given a system with degrees of freedom $\mathbf{q} \in \mathbb{R}^{3n}$, define an energy function $\mathbf{E}(\mathbf{q})$. The forces in the system are the negative gradients of the energy with respect to the degrees of freedom of the system; namely $\mathbf{f}_e = -\nabla_{\mathbf{q}}\mathbf{E}(\mathbf{q})$. Here, we use the subscript e to denote that this is the elastic force of the system.

At equilibrium, it should be true that $\mathbf{f}_e(\mathbf{q}) + \mathbf{f}_{\text{ext}} = 0$, where the subscript "ext" denotes the external forces of the system. In other words, given the external force vector f_{ext}, we need to find a position vector q so that $\mathbf{f}_e(\mathbf{q}) = -\mathbf{f}_{\text{ext}}$.

For linear model, we have $\mathbf{K} = -\nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})$ is constant over all q. We can solve this system

by simply solving a linear system:

$$\begin{aligned} &\nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})(\mathbf{q}-\mathbf{q}_{\text{rest}}) = -\mathbf{f}_{\text{ext}} \\ \Rightarrow &\mathbf{K}(\mathbf{q}-\mathbf{q}_{\text{rest}}) = \mathbf{f}_{\text{ext}} \\ \Rightarrow &\mathbf{K}\mathbf{U} = \mathbf{f}_{\text{ext}} \end{aligned} \tag{2}$$

Thus, the game goes something like this:

1. Derive a form of the energy density of the system.
2. Integrate that energy over the hexahedral mesh.
3. Differentiate energy with respect to $\mathbf{q}$ to calculate the elastic force.
4. Differentiate the elastic force with respect to $\mathbf{q}$ to generate the $\mathbf{K}$ matrix.
5. Solve a linear system.

For nonlinear elasticity materials, $\nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})$ is not a constant value any more, so we need to use Newton's method to find the root of this nonlinear equation.

Please read the attached document `FEM_cheatsheet.pdf` in the root folder of the codebase to carefully to learn how to derive the $\mathbf{K}$ matrix. (Optional) You can also refer to [1] as a detailed tutorial.

3 Assignment

This homework has four main parts. In the first part, you'll implement linear FEM to compute the deformation of the tetrahedral mesh under some external force. This amounts to computing a stiffness matrix $\mathbf{K}$, the loads on the vertices $\mathbf{f}$, formulating a linear expression, $\mathbf{K}\mathbf{U} = \mathbf{f}$, and solving for $\mathbf{U}$, a vector of vertex displacements. In this part, you need to fill some functions including computing the stress differential and formulating the stiffness matrix $\mathbf{K}$.

In the second part, you'll use the nonlinear material model provided by us and implement a simple Newton solver to solve for a nonlinear material model.

In the third part, you'll take one of the meshes and apply your customized boundary conditions and compute the deformation of it.

In the fourth part, you'll learn using neural network to train a surrogate simulation model.

3.1 FEM with Linear Material Model (50 points)

We direct the reader to the attached document `FEM_cheatsheet.pdf`, which provides a more extensive derivation of several constitutive models. Here, we provide a brief overview of the material for the assignment.

The first component we need to have is the linear material model. As a function of the deformation gradient, the energy density Ψ of a linear material can be written as:

$$\Psi(\mathbf{F}) = \mu\epsilon : \epsilon + \frac{\lambda}{2}\,\mathrm{Tr}^2(\epsilon) \tag{3}$$

where $\mathbf{F}$ is the deformation gradient (some places, like in the code, we use $\mathbf{f}$ for deformation gradient, but we'll use $\mathbf{F}$ in the writeup so we don't confuse it with force). ϵ is the small strain tensor ":" is the tensor contraction, and μ and λ are Lamé parameters, which can be calculated from the Young's Modulus and Poisson's ratio (please see the table here for conversions: https://en.wikipedia.org/wiki/Lam%C3%A9_parameters).

Differentiating, we achieve:

$$\mathbf{P}(\mathbf{F}) = \frac{\partial \boldsymbol{\Psi}(\mathbf{F})}{\partial \mathbf{F}} = \mu(\mathbf{F} + \mathbf{F}^T - 2\mathbf{I}) + \lambda\,\mathrm{Tr}(\mathbf{F} - \mathbf{I})\mathbf{I} \tag{4}$$

As can be seen, the Piola-Stress Tensor ($\mathbf{P}$) (a measure of pressure, or force per cross-sectional area) is linearly dependent on the deformation gradient. This is good, because it implies that the elastic force is linear in the mesh deformation - implying that our eventual $\mathbf{K}$ matrix will be constant. In order to construct the stiffness matrix $\mathbf{K}$, you need to implement a function to compute the stress differential $\frac{\partial \mathbf{P}(\mathbf{F})}{\partial \mathbf{F}}$.

The remaining part necessary for implementing FEM is to implement the stress differential function in `assignment2/material.py`. The stress differential should be the derivative of $\mathbf{P}$ with respect to $\mathbf{F}$. Your code only has to work for dim=3, though if you implement it for general dim your system can be extended to other dimensional systems.

The second step is to construct the stiffness matrix, $\mathbf{K} = -\nabla_{\mathbf{q}}\mathbf{f}_e$. The function to compute it is `stiffness_matrix` in `assignment2/fem.py`. Computing $\mathbf{K}$ needs to numerically integrating the stiffness over the tetrahedral mesh. In order to do that, the discretization rule should be applied. We compute the contribution from each tetrahedron element individually. The computation can be performed *via* the chain rule, as shown on the attached note. A good property of such discretization is that the $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ is constant over the whole tetrahedron, which helps us simplify the integration inside each element. After you've computed the contribution from each single tetrahedron, the global stiffness matrix is the sum of contributions of all stiffness matrices of each tetrahedron (indexed to the correct dimension of the system).

We have implemented the simulation function for linear material `solve_linear` in `fem.py`. The function computes the static equilibrium force for a beam as shown in Figure 1. The left side of the beam is fixed and external forces are applied to the bottom right region. The function calls your implemented stiffness matrix function to get the $\mathbf{K}$ matrix and solve $\mathbf{KU} = \mathbf{f}_{\text{ext}}$. You can visualize the deformed mesh in the result folder using **Meshlab**.

HINT: To aid you in debugging all of your steps so far, we have provided results from our solution code. The first beam is with size $4 \times 2 \times 2$, and the second beam is with size $20 \times 8 \times 8$. We provided the final deformed meshes for them and additionally the K matrices for the first case. You can find them in the folder `data/assignment2/std/` under the name `K_*_linear.txt` and `deformed_*_linear.stl`.

Your task in this section is summarized as follows:

1.1 (20 points) Compute $\frac{\partial \mathbf{P}(\mathbf{F})}{\partial \mathbf{F}}$ for linear material: finish the `stress_differential` function in `assignment2/material.py` by filling in the blank lines.

1.2 (30 points) Compute stiffness matrix $\mathbf{K}$: finish the `stiffness_matrix` function in `assignment2/fem.py` by filling in the blank lines.

The rules for filling in the blank lines remain unchanged: you are allowed to substitute each placeholder marked by "`<--`" using one line of code only. Compressing an if-else statement or a loop into one line is not permitted.

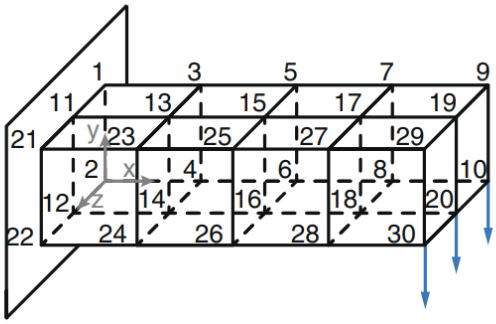

Figure 1: An example of how your voxel grid loading and contraints should look (dimensions may vary)

3.2 Nonlinear Material (40 points)

Linear material model gives a descent approximation when deformation is small. When large deformation is desired, linear material model will lead to unrealistic results. In this part, we are going to try nonlinear material model.

Unlike linear material model, the static equilibrium deformation for nonlinear material cannot be solved by a single linear equation. For nonlinear elasticity materials, $\mathbf{K}(\mathbf{q}) = \nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})$ is not a constant value any more, so we need to use Newton's method to find the root of this nonlinear equation. Roughly speaking, in order to solve $\mathbf{f}_e(\mathbf{q}) = -\mathbf{f}_{ext}$, Newton's method linearize the function around current solution $\mathbf{q}_i$ and update the solution $\mathbf{q}_{i+1}$ by solving the linear system:

$$\nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})|_{\mathbf{q}_i}(\mathbf{q} - \mathbf{q}_i) + \mathbf{f}_e(\mathbf{q}_i) = -\mathbf{f}_{\text{ext}} \tag{5}$$

By iteratively solving such local linear system, Newton's method will converge to the solution to the original nonlinear system.

We have implemented a nonlinear material model (Neo-Hookean material) for you, and you need to complete the tasks below:

2.1 (30 points) Implement the Newton's method in the `solve_newton` function in `assignment2/fem.py` for the nonlinear material model.

2.2 (10 points) Compare between linear and nonlinear models by choosing appropriate parameters for testing (including beam size and external force, see the hints below for instructions). Describe the pros and cons of both methods in your report based on your designed example. **Attach snapshots of your results to support your argument**.

HINT:

- To aid you in debugging all of your steps so far, we have provided results from our solution code. The first beam is with size $4 \times 2 \times 2$, and the second beam is with size $20 \times 8 \times 8$. We provided the final deformed mesh for both cases in the folder `data/assignment2/std/` under the name `deformed_*_nonlinear.stl`. Your results don't have to be numerically identical to the reference due to various possible implementations (e.g., termination criteria and line search) of the Newton's method, but they should be visually the same.

- You can optionally specify the beam size and the external force using command line arguments when running `main.py` like

  ```
  python assignment2/main.py -c 8x4x4 -f 0,0,-100
  ```

  The `-c` parameter refers to the beam size and has a format of `axbxc`, where `a`, `b`, and `c` are the number of vertices in X, Y, and Z directions. The `-f` parameter specifies the external force to apply in the format of `fx,fy,fz` (default to `0,0,-50`), where `fx`, `fy`, and `fz` are the force components in X, Y, and Z directions.

3.3 Custom Boundary Condition (10 points)

With your linear and nonlinear solver in tow, you are now prepared to simulate one of the 3D meshes named `spot`. Recall that we first translated a triangle mesh to a tetrahedral mesh and then deformed it by BBW. Now, you are going to use FEM to simulate the deformation of `spot` (with the Neo-Hookean material model).

We already provided you with the tetrahedralized `spot` mesh in the `data/assignment2` folder and the starter code for creating boundary conditions in `assignment2/main.py`. Your task is to implement your own boundary conditions to deform the mesh as you like. Specifically, you will make some simple rules to determine what vertices should be fixed and what vertices to exert force upon. Then, you will run the program using the following command

```
python assignment2/main.py -m spot
```

to obtain the deformed mesh.

3.1 (10 points) Implement your customized boundary conditions for `spot` by changing the lines that define the constraints in the codebase. The function to work with is `boundary_conditions_custom` in `main.py`. Please see the comments in the code for details. Describe in your report what kind of boundary constraints you designed and **attach screenshots of the deformed mesh**.

Tips: If the external force is too large, the Newton's method might take a long time to converge. In that case, consider reducing the external force while keeping the deformation visible.

3.4 Surrogate Simulation using Neural Network (40 Points)

This section aims to deepen your understanding of neural networks through practical simulation exercises. You will first engage with a Google Colab notebook to familiarize yourself with the key concepts and functionalities of neural simulators. The will need to upload a dataset to the Colab, which is provided in `data/dataset.tar.gz` Following this guided exploration, you will tackle a set of problems that apply these concepts in practical scenarios.

To start, access the provided Google Colab notebook. Ensure you're logged into your Google account to interact with the notebook. Proceed to run and engage with all cells in the notebook, following any instructions provided within. This part is designed to introduce you to neural network concepts, including their architecture, how they process information, and how they learn.

One important part of designing a neural network application is understanding the problem domain and choosing

1. A representation for the input
2. The number of output units and what range of values they can take on
3. The loss function to try to minimize, based on actual and desired outputs

In this problem we will concentrate on the number of output units, activation functions on the output units, and loss functions. These should generally be chosen jointly.

Just as a reminder, among different loss functions and activation functions, we have studied:

1. Activation functions: linear, ReLU, sigmoid, softmax
2. Loss functions: hinge, negative log likelihood (NLL a.k.a. cross-entropy), quadratic (mean squared)

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. When you choose to use multiple output units, be very clear on the details of how you are applying the activation and the loss.

3.1 (5 points) Map the words on the front page of the New York Times to the predicted (numerical) change in the stock market average.

3.2 (5 points) 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.

3.3 (5 points) Map the words in an email message to which one of a user's fixed set of email folders it should be filed in.

3.4 (5 points) 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.

We will now focusing on using neural network to learn a surrogate model for physical simulation. Upload the dataset file to Colab before you run the cell. The dataset is created from a fluid simulation process. In particular, the dataset encodes the location of deformation handles of a geometry as the features, and the predicted lift and drag of the geometry in from fluid simulation as the labels. We will train a neural network to fit to this simulation data. Run the cells on *2 Learning Simulation* from the Google Colab notebook and answer the following questions:

3.5 (5 points) We could also consider some other metrics. Which ones might you consider, and why?

3.6 (15 points) 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?

4 Submission Guidelines

All assignments must be submitted onto the Canvas submission system by the deadline. The submission should be in one single .zip file, including two parts.

1. All the code in the `assignment2` folder.

2. A short .pdf report describing how your code works and the results.

Late assignments will incur a 25% penalty for each late day (3 late days are permitted without penalty throughout the semester). No assignments will be accepted after the solutions are released (typically within a few days of the deadline; contact the TA for details).

4.1 Writeup Instructions

The writeup should be relatively short (around $1-2$ page). In your writeup, please tell us the following:

- At a high-level, what did you implement in order to complete the assignment?
- Some pictures to show your results.
- What did you like about the assignment?
- What did you not like about the assignment?
- Any other issues about which we should be aware?

Please submit the writeup as a .pdf and embed any relevant images inside.

References

[1] Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, pages 20:1–20:50, New York, NY, USA, 2012. ACM.