CME 211: Homework 4

Due: Friday, November 4, 2022 at 5:00pm PDT

This assignment was designed by Patrick LeGresley and modified for the purposes of this course.

Background

A simple structural engineering problem is the analysis of the stability and/or forces in a *2D truss:*

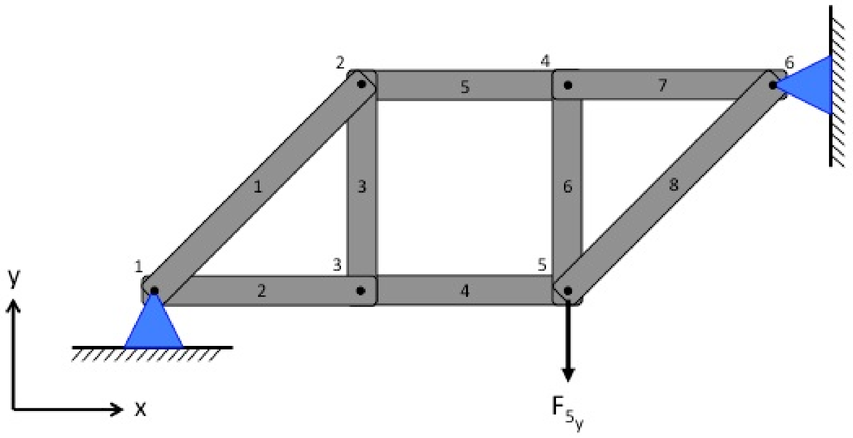

Figure 1: 2D truss

The truss is made up of *beams* that are joined by frictionless pins, and the supports are rigidly fashioned so that they cannot move. Using the *method of joints*, external forces only act at the pinned *joints* (as in joint 5 above), and the beams are perfectly rigid. This means that the only forces in the beams are in the direction of the beams. The truss is in equilibrium if the forces at each joint sum to zero. For an example joint, and assuming the beams are in compression, the two static equilibrium equations for joint 2 are shown in Figure 2 where B is the compression force that is **directed along the direction of the beam** and F is the (possibly non-zero) external force at each joint and could have arbitrary direction. These external forces are an input to the analysis and will often be zero for many of the joints. Note that if the beam is actually in tension, the sign of the force will turn out to be negative instead of positive.

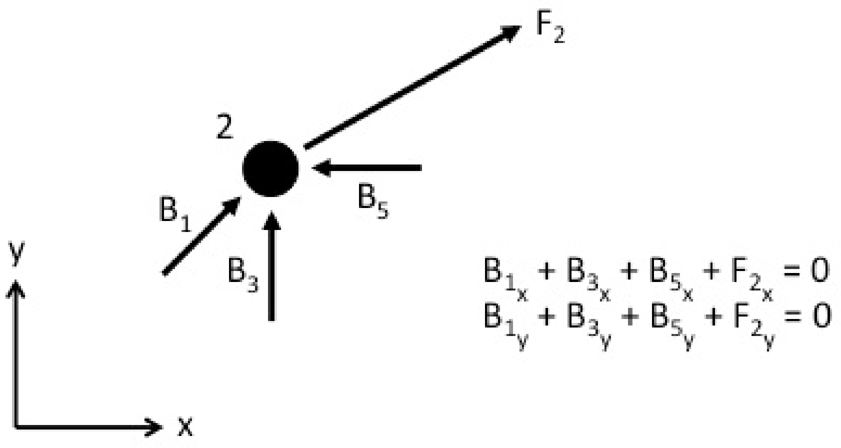
F2
2
B1
B5
B3
y
x
$B_{1_x} + B_{3_x} + B_{5_x} + F_{2_x} = 0$
$B_{1_y} + B_{3_y} + B_{5_y} + F_{2_y} = 0$

Figure 2: Example static equilibrium equations for joint 2

With two equations per joint, the total number of equations is twice the number of joints. The unknowns in the system of equations are:

- the compression or tension force for each beam
- the x and y components of the reaction force due to the fixed supports, denoted R in Figure 3

In general the reaction forces can have an arbitrary direction and depend on the external forces applied to the truss.

Note that the beam forces ($B_1, B_2 \ldots$) have a fixed direction (up to a sign) being that of the beam. Since we don't need to solve for the direction of the beam forces, we can compute a normal vector in the direction that the beam force is being applied and use that to determine our x and y components (B_{i_x}, B_{i_y}). For example, in Figure 2 we would compute B_{1_x} as:

$$\frac{J_{2_x} - J_{1_x}}{\sqrt{(J_{2_x} - J_{1_x})^2 + (J_{2_y} - J_{1_y})^2}} B_1 = B_{1_x}$$

Where J_{1_x} refers to the x location of joint 1. We would similarly compute B_{1_y} by changing the numerator of the coefficient to use the y locations. Convince yourself that B_{1_x} and B_{1_y} in the context of Figure 3 are the same values as B_{1_x} and B_{1_y} in the context of Figure 2, just with opposite signs.

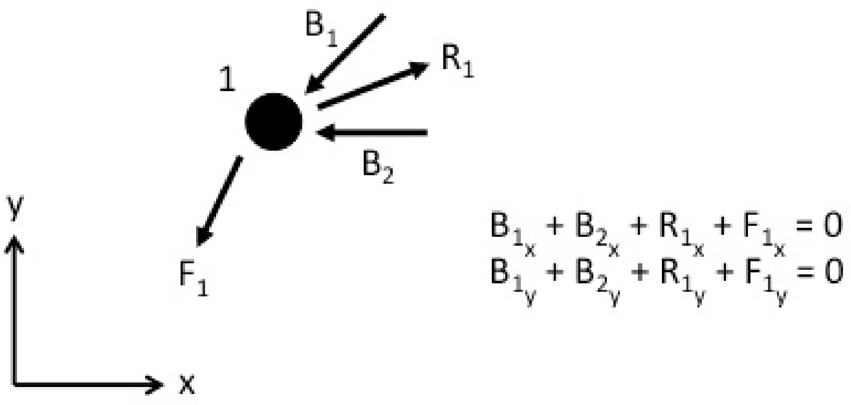
B1
R1
1
B2
y
F1
x
$B_{1_x} + B_{2_x} + R_{1_x} + F_{1_x} = 0$
$B_{1_y} + B_{2_y} + R_{1_y} + F_{1_y} = 0$

Figure 3: Reaction force at the supports

If there are a different number of equations and unknowns the system is either over or underdetermined and is not a suitable geometry for static equilibrium analysis using the method of joints. However, just because a given truss geometry has an equal number of equations and unknowns does not mean the resulting system of equations can be solved. An unstable geometry, such as in Figure 4, will be apparent if the linear system of equations turns out to be singular.

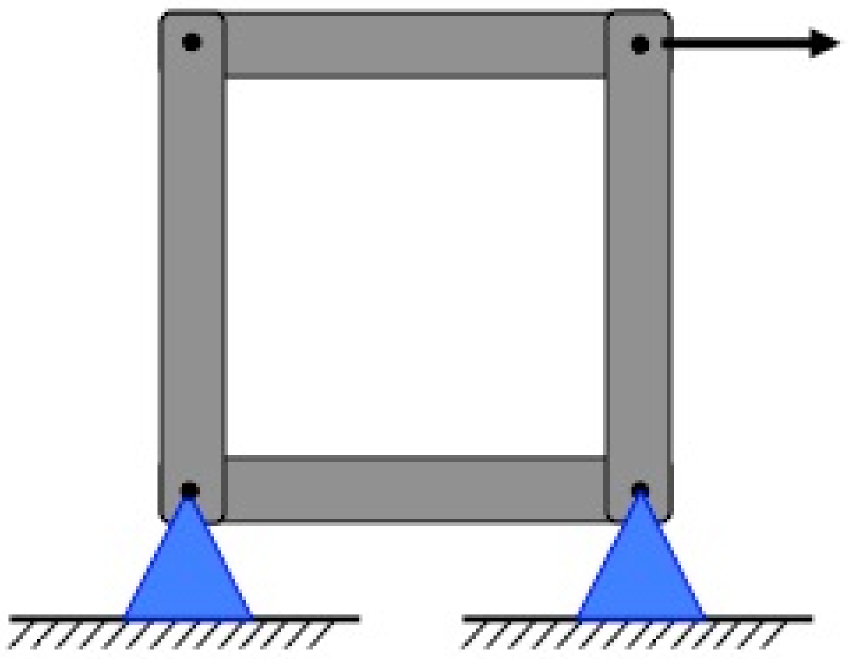

Figure 4: Unstable truss geometry

Preparation

- **This assignment requires that you have completed CME211 HW0.**
- Make sure you have an up-to-date local clone of your CME211 homework repository.
- Create directory named `hw4` at the top level of your CME211 homework repository.
- All work for this homework goes inside of the `hw4` directory.

Assignment (60 points functionality, 20 points design, 10 points style, 10 points writeup)

Your submission will include a `truss.py` file defining a `Truss` class for loading and analyzing a 2D truss using the method of joints and a `main.py` file to execute your program from the command line. The initialization method of your class should take as arguments the names for joints and beams files respectively.

Figures 5 and 6 show the files `truss1/joints.dat` and `truss1/beams.dat`.

Joints files start with a comment line with column names indicated with `#`. The first column is the one based joint number and the file will always be in joint number order. The second and third columns refer to the x and y positions of the joints respectively. Columns labeled `Fx` and `Fy` are components of the external forces being applied at each joint. The last column is a Boolean value indicating whether this is a rigidly supported point with zero displacements.

```
# joint      x      y   Fx  Fy  zerodisp
      1     0.     0.   0.  0.         1
      2     1.     1.   0.  0.         0
      3     1.     0.   0.  0.         0
      4     2.     1.   0.  0.         0
      5     2.     0.   0. -1.         0
      6     3.     1.   0.  0.         1
```

Figure 5: file: `truss1/joints.dat`

Beam files also starts with a comment line. The first column is the one based beam number and the file will always be in beam number order. For each beam the remaining two columns contain the one based index of the two joints connected by the beam.

```
# beam Ja Jb
     1  1  2
     2  1  3
     3  2  3
     4  3  5
     5  2  4
     6  4  5
     7  4  6
     8  5  6
```

Figure 6: file: `truss1/beams.dat`

The `PlotGeometry()` method of your class should use matplotlib to create a simple plot showing the geometry of the truss as shown in Figure 7. The method should take in a file name as a Python string and save the plot at that location. In `main.py`, this method should be optionally invoked if the user of your program specifies a file name as the third command line argument.

A note on using matplotlib in Rice: When using matplotlib versions older than 3.1 (which is the case for the matplotlib on Rice), it is necessary to explicitly instantiate the 'Agg' backend to write to a file. Please import matplotlib in the following way:

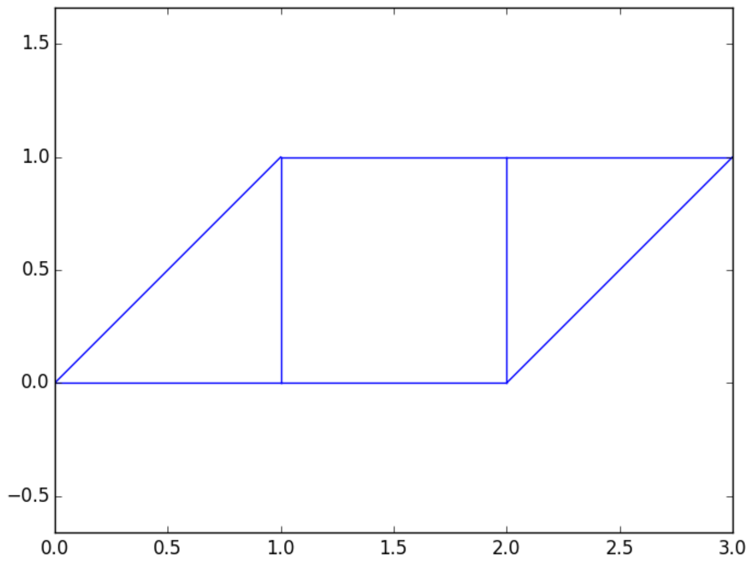

Figure 7: Simple matplotlib plot of 2D truss geometry from Figure 1

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

However, 'Agg' does not support `plt.show()`. We recommend secure copying your file from Rice onto your local computer to confirm that the plot was what you expected. You could also follow the instructions in the document found at the following link to view the files remotely from your local machine: https://docs.google.com/document/d/1XjQymDdo6LCYfd5NmxOOjSUqbxrCDGq6FB0Lq713pok/edit?usp=sharing.

Furthermore, a string representation method should print the beam forces for the truss (see below for expected output formatting).

In addition to the `__init__()`, `PlotGeometry()`, and `__repr__()` methods, you will need to implement one or more additional methods to implement the actual analysis of the truss to compute the beam forces. Use good programming practices like decomposition and where possible avoid loops in Python by using slicing notation and calling functions from NumPy or SciPy.

For the matrix form of the equations you need to use a sparse matrix. Either a CSR and/or COO format would be a reasonable choice. First forming the dense matrix and then converting it into a sparse matrix is not acceptable. However, you might find it easier for your initial implementation / testing / debugging to start by forming the dense matrix and later switch to a sparse matrix. For this, you might find the NumPy `zeros()` function useful for pre-allocating storage for matrices. The other option is to form the sparse matrix from the beginning, but use the `todense()` method for purposes of printing it out. Either way, you may find the NumPy `set_printoptions()` function useful for controlling how NumPy wraps rows of the matrix when printing it out.

The `main.py` file gathers command line arguments, demonstrates use of your `Truss` class, and prints the instance to show the computed beam forces for the truss. If called with no input arguments, the program outputs a usage message. For basic operation `main.py` must be passed filenames for joint and beam data as the first two command line arguments. An optional third argument specifies the output filename for a matplotlib plot. Example usage is shown below:

```
$ python3 main.py
Usage:
    python3 main.py [joints file] [beams file] [optional plot output file]
```

```
$ python3 main truss1/joints.dat truss1/beams.dat
 Beam       Force
-----------------
    1       0.000
    2      -1.000
    3       0.000
    4      -1.000
    5       0.000
    6       0.000
    7      -0.000
    8      -1.414
```

Both `truss1` and `truss2` are suitable geometries for static equilibrium analysis via the method of joints. Other examples include `truss3`, which is an overdetermined system, and `truss4`, which is the unstable geometry from Figure 4. Your `Truss` class should detect these issues and raise exceptions:

```
$ python3 main.py truss3/joints.dat truss3/beams.dat
Traceback (most recent call last):
    File "main.py", line 16, in &lt;module&rt;
        t.ComputeStaticEquilibrium()
    File "/afs/.ir.stanford.edu/users/p/l/plegresl/CME211/Assignment4/truss.py",
        line 78, in ComputeStaticEquilibrium
    raise RuntimeError, "Truss geometry not suitable for static equilbrium analysis"
RuntimeError: Truss geometry not suitable for static equilbrium analysis
```

```
$ python3 main.py truss4/joints.dat truss4/beams.dat
Traceback (most recent call last):
    File "main.py", line 16, in &lt;module&rt;
        t.ComputeStaticEquilibrium()
    File "/afs/.ir.stanford.edu/users/p/l/plegresl/CME211/Assignment4/truss.py",
        line 116, in ComputeStaticEquilibrium
    raise RuntimeError, "Cannot solve the linear system, unstable truss?"
RuntimeError: Cannot solve the linear system, unstable truss?
```

Note that the SciPy sparse direct solver only issues a warning for a singular system. To catch warnings as exceptions, import the `warnings` module and then do this sometime prior to invoking the solver:

```
# Catch warnings as exceptions
warnings.filterwarnings('error')
```

Modules

You *are* allowed to use `numpy` and `scipy` on this assignment. However, you are *not* allowed to use `pandas`, `statistics`, or `tabular`.

Performance

Your code will not be stress-tested. We will only test your code for correctness using the provided data files.

Write up (5 pts)

In a `README.tex` file provide a description of the inner workings of your `Truss` class. Imagine that you would like to hand the code off to another engineer. The document should explain to the other engineer the set of methods in the class and how they work. It is important to include details like the type of sparse matrix used and the input and output arguments for each method.

Create a `README.pdf` from `README.tex` and commit both files to the repository.

Checklist & submission

The following files must be present in the `hw4` directory of your CME211 GitHub homework repository by the deadline:

- `truss.py`: python module containing your `Truss` class
- `main.py`: python driver to compute forces and optionally generate a plot
- `README.tex`: LaTeXdocumentation for your `Truss` class
- `README.pdf`: PDF generated from `README.tex`

We will use a Git tag to mark the commit that you want graded. A Git tag is essentially a bookmark to a particular commit. After you have committed the version of the code you would like graded, create a hw4 tag with the following command:

```
$ git tag -am "HW4 Submission" hw4
```

or

```
git tag -m "HW4 Submission" -a hw4
```

The quotes contain a tag message. In the above command hw4 is the actual tag. You can push commits along with tags to the remote repo with:

```
$ git push --tags origin main
```

It is a good idea to check if the tags show up on GitHub. These appear in the "release" tab on the web interface.

It is possible to change your tag later, too, if you discover a bug after pushing:

```
$ git tag -d hw4 # delete the hw4 tag pointing to old commit

$ git push --delete origin hw4 # delete the remote hw4 tag

$ git tag -am "My HW4" hw4 # create hw4 tag pointing to current commit

$ git push --tags origin main # push changes to code and tags
```

We will use Git time-stamps on tags and the associated commits to check deadlines. Be careful with overwriting tags – you don't want to lose the submission tag.

Notes:

- **It is not acceptable to commit auxiliary LaTeX files to your GitHub repo.**
- It is acceptable to commit the truss data file directories into your GitHub repository.
- Take care to follow the file and directory names exactly. Everything is case sensitive.
- Your homework is not complete until you have pushed the above files to GitHub.
- Do regular commits when you have finished little pieces of the puzzle. Don't do a commit for small individual changes, but also don't make only one huge commit. There's no set rule for when to commit. Whenever you feel the need to 'save' your (good) progress, think about committing.
- When you do a commit, type a sensible commit message such that your repository's commit log makes sense to an outsider, for tracking what you have done. This is very important if you work with a team on the same code.