CME 211 HW4 WRITE UP

LAURA LYMAN

Contents

1. Code Description

The class `Truss` returns a data table of the net force acting on each beam. That is, this class creates a system of linear equations by balancing the x and y forces acting on each joint, where the forces provided by the beams are variables. Then the class solves this system of equations to determine each beam force and outputs the results in a data table. If the system cannot be solved (i.e. there is a singular matrix or non-square matrix) the class returns an error with an explanation.

Also, if the user inputs an (optional) file name, the code creates a figure of the truss system and saves it as that name. If no directory is included in part of the file name, the default is to save the plot as the given name in the current directory. If no plot name is provided, the plot is saved with the default name "figure" with the default extension '.png' in the working directory. Each figure is a picture of the corresponding joints/beam system, regardless of whether the associated matrix equation has a solution.

Specifically, here is a description of some of the methods in the Truss class.

Method: `_init_(self, joints_file,beams_file, plot_file)` Returned Variables: None **Description:** Initializes the class and its methods. Gives the Truss class attributes `joints_file` and `beams_file` from the user-inputted data files. Also gives the Truss class a `plot_file` attribute, which is where the plot should be saved/what to name the plot if the user specifies a file path; otherwise `plot_file` is given the default name `figure.pdf` and saved in the working directory. Then the class methods are initialized.
Class Method: `get_beam_data(self,beams_file)` **Description:** Creates the `beam_joints` attribute, which is a dictionary in which the keys are beams and the values are the joint indices of the two joints per each beam. The method reads through the `beams_file` line by line, storing all the information from this file in the `beam_joints` dictionary.
Class Method: `get_joints_data(self,joints_file)` **Description:** Reads and retrieves information from the joints data file. This method adds three class attributes that are all dictionaries and populates them with data: `joints_with_attached_beams`, `joint_position`, and `joint_forces`. Each of these dictionaries have the joints (i.e. joint indices) as keys, which are unique identifies. Then: • `joints_with_attached_beams` has values that are the beam indices of the beams attached to each joint • `joint_position` has values that are the (x, y) coordinates for each joint, and •`joint_forces` has values that are lists including: the x-forces on each joint, the y-forces on each joint, and a boolean for whether each joint is attached to a wall.
Class Method: `get_beam_length(self)` **Description:** Creates two dictionaries `beam_lengths` and `beam_trig`, whose keys are joint indices. Uses trigonometry to compute the length of each beam (populating `beam_lengths`). Computes the sine and cosine of the angles of a beam (relative to the first joint) and puts these two values into a list; then `beam_trig` is populated with these lists of angle information.
Class Method: `check_if_square(self)` **Description:** Checks if the matrix of data is square. Throws an error if the matrix is rectangular i.e. the number of equations (m) does not equal the number of variables (n). There are 2 equations per joint. There is one variable per beam. Also, there are two additional variables per attached joint representing the force of attachment in the x and y directions. So the total number of variables is `number_of_beams + 2*number_attached_joints`.

The beam equations are then balanced by `get_beam_eqn(self)`, stored into CSR arrays (`update_CSR_arrays`), and put into a CSC matrix using `scipy.sparese.csc_matrix`. Then `check_for_singular_matrix` checks if this CSC matrix is singular (to know whether a solution can be found) and outputs an error if needed. The `solve_linear_system` method solves for the forces on each beam using `scipy.sparse.linalg.spsolve`, and `PlotGeometry` handles the figures of each truss system. The code in `truss.py` contains detailed comments describing all of the other helper functions used.

2. Table Results (Beam/Force Tables)

```
$ python3 main.py 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
$ python3 main.py truss2/joints.dat truss2/beams.dat
 Beam       Force
-----------------
    1   -2000.044
    2    1732.102
    3     866.032
    4   -2500.055
    5   -2020.741
    6    3175.520
$ python3 main.py truss3/joints.dat truss3/beams.dat
ERROR: Linear system is not square. Method of joints isn't suitable.
$ python3 main.py truss4/joints.dat truss4/beams.dat
ERROR: Matrix is singular, so linear system cannot be solved.
```

3. Plotting Results

The plots of the truss systems are below.

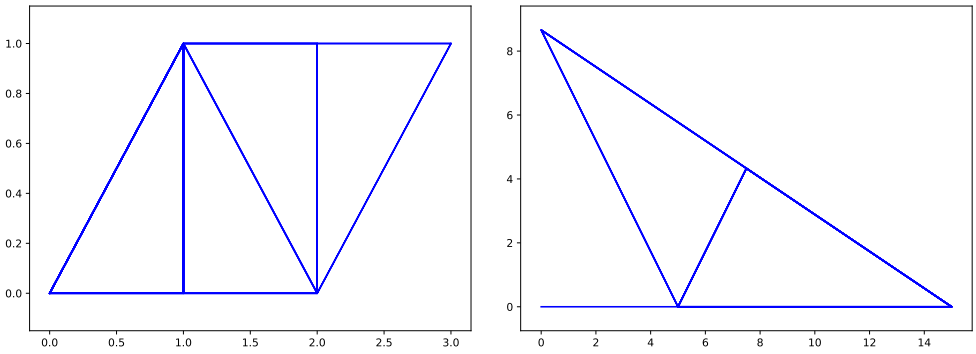

Figure 1. Plots of truss system 1 (left) and truss system 2 (right).

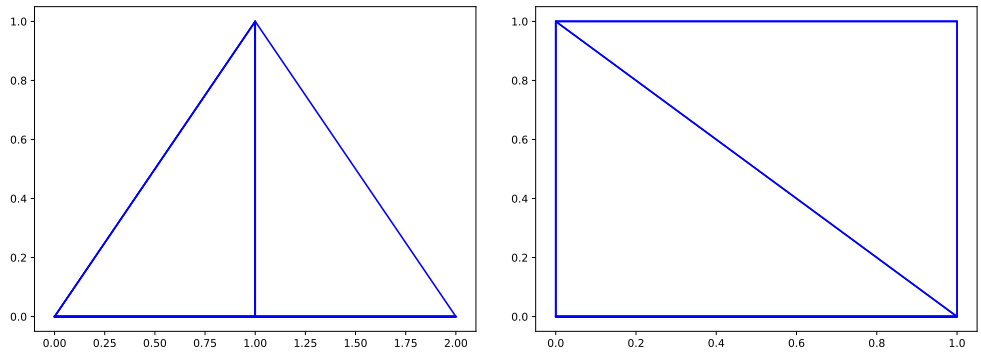

FIGURE 2. Plots of truss system 3 (left) and truss system 4 (right).

4. Error Handling

Here we verify that an error is thrown if the inputted joints or beam file has formatting issues that will cause the file to not be found by `os.path` (i.e. "correct" file names but excessive '/' characters).

```
$ python3 main.py /truss1/joints.dat truss1/beams.dat
ERROR: Make sure '/truss1/joints.dat' does not start or end with a '/' character.

$ python3 main.py truss1/joints.dat truss1/beams.dat/
ERROR: Make sure 'truss1/beams.dat/' does not start or end with a '/' character.
```

Then we verify that an error is thrown if the inputted joints or beams file is located in a nonexistent directory.

```
$ python3 main.py truss5/joints.dat truss1/beams.dat
ERROR: File 'truss5/joints.dat' not found.
```

The following checks how the code handles a reference to a nonexistent file in one of the valid directories.

```
$ python3 main.py truss1/joints.dat truss1/filenothere
ERROR: File 'truss1/filenothere' not found.
```

When the user specifies a valid directory without a file, the error is caught under the formatting issues umbrella.

```
$ python3 main.py truss1/joints.dat truss1/
ERROR: Make sure 'truss1/' does not start or end with a '/' character.
```

We can check how the code handles these similar file issues when the user specifies a plot file name.

```
$ python3 main.py truss1/joints.dat truss1/beams.dat /myplotfile
ERROR: Make sure '/myplotfile' does not start or end with a '/' character.
$ python3 main.py truss1/joints.dat truss1/beams.dat truss10/myplotfile
ERROR: Directory 'truss10' not found.
```

```
$ python3 main.py truss1/joints.dat truss1/beams.dat /truss5/fig
ERROR: Make sure '/truss5/fig' does not start or end with a '/' character.
```

Finally we need to verify that you can actually specify a plot file name and have the plot saved accordingly. If you have a certain directory in the plot file name (that is valid), we will check that the plot is actually saved there. If no directory is specified (i.e. the user just says "fig1"), we will check that the plot is saved as "fig1" in the current directory. Note that if no extension is provided by the user for the plot, then the plot is saved as a '.png' by default. If the user does not input a plot file, we should verify that the plot is saved with the default name "figure" in the working directory. Finally, we should ensure that tables are still working/outputting correctly when a plot file has been provided.

```
$ python3 main.py truss2/joints.dat truss2/beams.dat testfig2
 Beam       Force
-----------------
    1   -2000.044
    2    1732.102
    3     866.032
    4   -2500.055
    5   -2020.741
    6    3175.520
$ ls
Figure1.pdf         README.synctex.gz       testfig2.png
Figure2.pdf         README.tex              truss3
Figure3.pdf         README.toc              truss4
Figure4.pdf         cme211-hw4.pdf
README.aux          main.py
README.log          truss.py
README.out          truss1
README.pdf          truss2
$ python3 main.py truss2/joints.dat truss2/beams.dat
$ ls
Figure1.pdf         README.synctex.gz       truss2
Figure2.pdf         README.tex              testfig2.png
Figure3.pdf         README.toc              truss3
Figure4.pdf         cme211-hw4.pdf          truss4
README.aux          figure.png
README.log          main.py
README.out          truss.py
README.pdf          truss1
$ python3 main.py truss1/joints.dat truss1/beams.dat truss1/testfig.pdf
 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
$ cd truss1
$ ls
beams.dat        joints.dat        testfig.pdf
$ cd ..
$ python3 main.py truss3/joints.dat truss3/beams.dat fig
ERROR: Linear system is not square. Method of joints isn't suitable.
$ python3 main.py truss4/joints.dat truss4/beams.dat fig
ERROR: Matrix is singular, so linear system cannot be solved.
```

Therefore, the program appears to work correctly and manage several types of errors.