CME 211: Homework 3

Due: Friday Oct 21, 2022 at 5:00 p.m. Pacific Daylight Time

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

Background

Wing design might begin with the analysis and design of one or more airfoils, the 2D cross sections of a wing. Airfoils could be tested experimentally in a wind tunnel or via computer simulation using *Computational Fluid Dynamics* (CFD) to gather information about the pressure distribution, often expressed as the non-dimensional *pressure coefficient* (C_p) as shown in Figure 1.

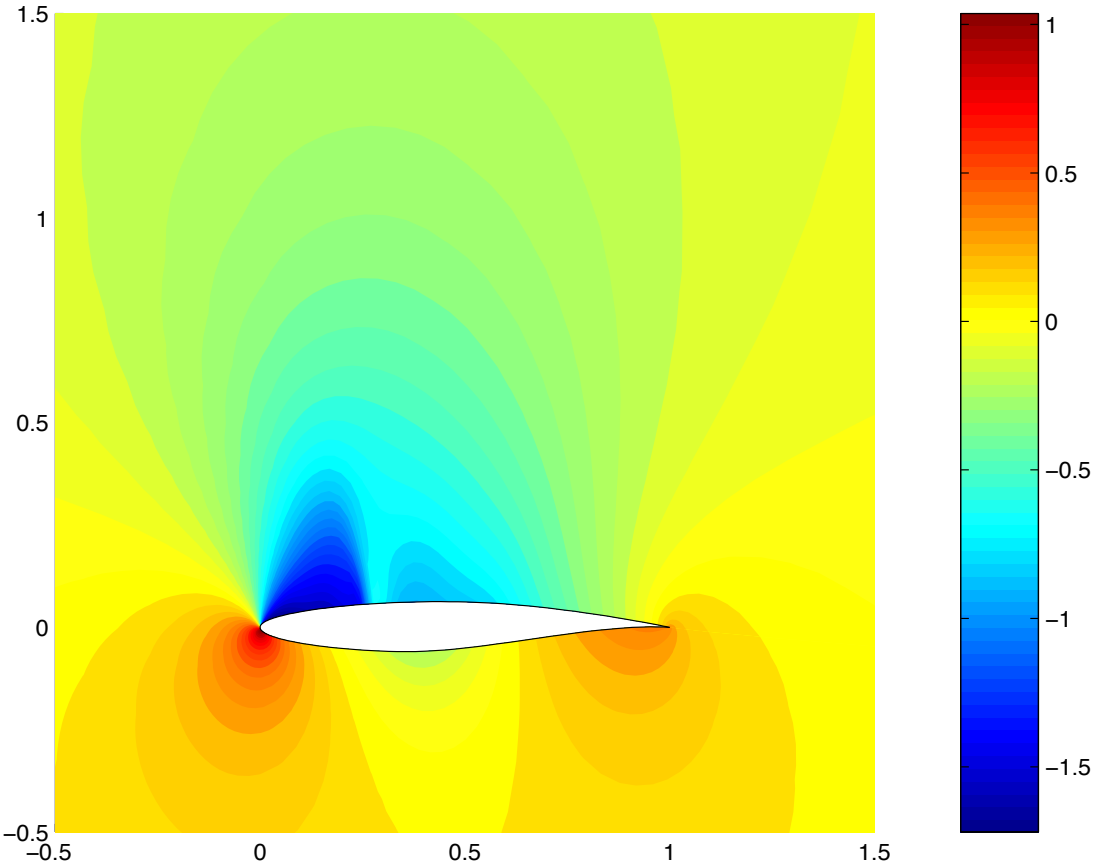

Figure 1: Pressure coefficient distribution around the RAE 2822 airfoil

The pressure coefficient distribution can be integrated over the airfoil surface to determine the non-dimensional *lift coefficient*. This can be accomplished by:

1. Computing the chord length (see Figure 2) as the distance from the leading edge to the trailing edge of the airfoil. The chord length will later be used to non-dimensionalize the airfoil geometry.
2. The airfoil geometry is defined by a series of points. Each pair of consecutive points can be considered to be connected by a straight line that forms a panel as shown in Figure 3. The pressure coefficient data is available for each of these panels and the non-dimensional force acting perpendicular to a panel can be computed by multiplying the pressure coefficient by the panel length non-dimensionalized by the chord length. This non-dimensional force acting perpendicular to a plane can then be decomposed into components into the Cartesian directions, Δc_x and Δc_y, as shown in Figure 3.

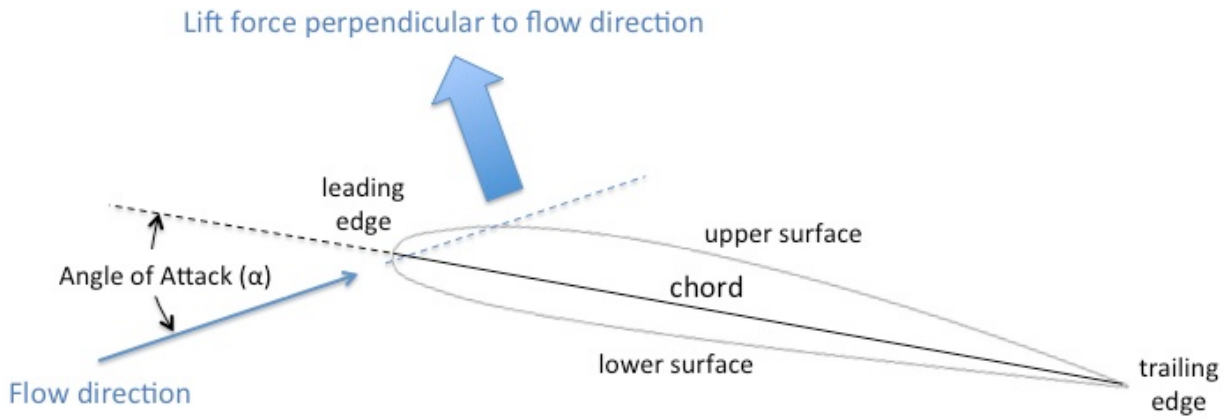

Figure 2: Relationship between flow direction and lift direction

3. The non-dimensional force contributions summed over all the of the panels yield the total Cartesian force coefficients c_x and c_y. Note that when you sum the contributions you will need to be careful about the sign of the contribution depending of whether it is in the positive or negative x or y direction.
4. And finally the lift coefficient, which is the non-dimensional force component perpendicular to the flow direction, can be computed as:

$$c_l = c_y \cos\alpha - c_x \sin\alpha$$

Other interesting characteristics of the flow could include identify the *stagnation point(s)* where the flow velocity goes to zero. The stagnation point corresponds to the pressure coefficient of 1.0 and an airfoil would typically have a stagnation point somewhere near the leading edge.

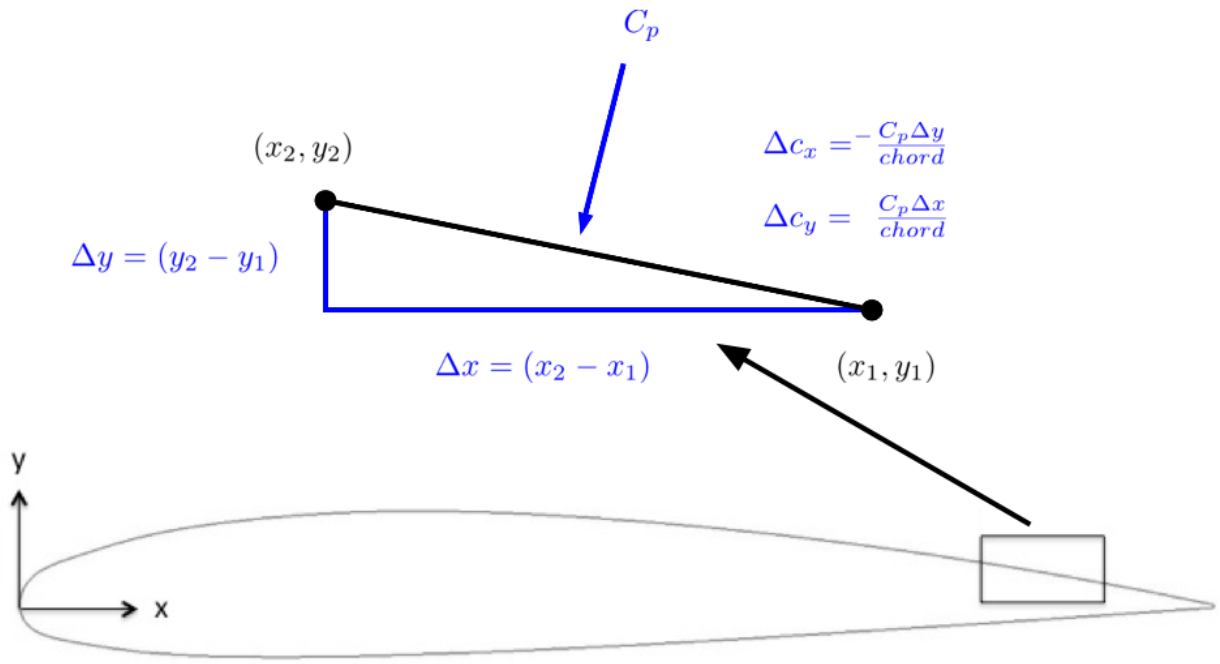

Figure 3: Relationship between flow direction and lift direction

Preparation

This assignment requires that you have completed CME211 HW0. If you have not completed HW0, please do so (even if it is past the HW0 deadline). Part of HW0 is the creation of your homework repository, which will be used throughout the quarter. You will not be able to complete HW3 (or any future CME211 homework) without completing HW0.

These instructions assume that you are logged into `rice.stanford.edu` and have cloned your CME211 homework repository to your Farmshare user directory. First, navigate to your homework repository:

```
$ cd /farmshare/user_data/[sunet_id]/cme211-[github_user]
$ ls
README.md  STUDENT  hw0  hw1  hw2
```

In the above `cd` command, `[sunet_id]` must be replaced with your SUNetID and `[github_user]` must be replaced with your GitHub username.

At this point, it is good to check if your local repository is clean and up-to-date with the remote repository on GitHub. This is achieved by running:

```
$ git status
On branch main
Your branch is up-to-date with 'origin/main'.

nothing to commit, working directory clean
```

If you get a different message, make sure to commit (or revert) all modified files or perform a `$ git pull` to retrieve remote changes. Now, create a `hw3` directory (it must be lower case):

```
$ mkdir hw3
```

```
$ ls
README.md  STUDENT  hw0  hw1  hw2  hw3
```

Now, create a basic `README` inside of `hw3`, commit it to the repository, and push to GitHub:

```
$ emacs hw3/README
# add some basic info to the README
# check status of the repo with $ git status
$ git add hw3
$ git commit -m "add hw3 readme"
# ... output omitted ...
$ git push origin main
# ... output omitted ...
```

All HW3 files must go inside of the `hw3` directory you just created. So, `$ cd hw3` and you are ready to go.

Assignment

Scoring: (60 points functionality, 20 design, 10 style, 10 writeup)

In this assignment you will use Object Oriented Programming (OOP) concepts to create an `Airfoil` class that will handle the loading and basic processing of data associated with an airfoil at multiple angles of attack. Each of the provided directories `naca0012/` and `naca2412/` contain data for one airfoil.

The main program that will invoke your `Airfoil` class has already been written in the provided `main.py` file that you should not change:

```
import sys

import airfoil

if len(sys.argv) < 2:
    print('Usage:')
    print('  python3 {} <airfoil data directory>'.format(sys.argv[0]))
    sys.exit(0)

inputdir = sys.argv[1]

try:
    a = airfoil.Airfoil(inputdir)
except RuntimeError as e:
    print('ERROR: {}'.format(e))
    sys.exit(2)

print(a)
```

A data directory, such as `naca0012`, is provided to the program via the command line arguments and used to instantiate an instance of your `Airfoil` class. Your program must be able to handle absolute and relative paths, and the data directory argument can be specified with or without a trailing slash. Using `print` on the resulting instance of the class should summarize the test data as shown here:

```
$ python main.py naca0012/
Test case: NACA 0012

alpha     cl           stagnation pt
-----  -------  -------------------------
-3.00  -0.3622  ( 0.0030,  0.0094)  0.9906
 0.00   0.0000  ( 0.0000,  0.0000)  0.9944
```

```
 3.00   0.3622  ( 0.0030, -0.0094)  0.9906
 6.00   0.7235  ( 0.0099, -0.0170)  0.9967
 9.00   1.0827  ( 0.0219, -0.0246)  0.9977
```

The summary shows the lift coefficient for each angle of attack and the (x, y) location and pressure coefficient for the stagnation points. Note that there may not be a point with a pressure coefficient of exactly 1.0 so you should return the panel value closest to 1.0 and an (x, y) coordinate that is the average of the points defining that particular panel.

Design Requirements & Considerations

How you implement your `Airfoil` class is largely up to you. However there are a few constraints and requirements:

1. You are not allowed to modify the `main.py` file. So your class will need to be instantiated via the name of a directory containing airfoil data, and it needs to implement the string representation method such that the output of the program is consistent with the example shown above.
2. Think about the OOP concepts discussed in lecture and implement your class in terms of multiple methods. Splitting it up into about 6 methods would be appropriate. Be sure to provide a document string for each method, plus one for the class as a whole.
3. It is important to provide clear, easily understandable feedback when errors exist in input data. In this assignment, you are required to generate `RuntimeError` exceptions with understandable messages in the following cases:
   - the argument to `airfoil.Airfoil()` is not a valid directory
   - any of the required data files cannot be found in the data directory
   - an error is detected in reading an input file

Raising a `RuntimeError`

It is simple to *raise* a `RuntimeError` in Python:

```
raise RuntimeError("error message goes here")
```

A `RuntimeError` is a built-in Python exception type for runtime errors.

File formats

In each airfoil directory there should be one `xy.dat` containing the (x, y) coordinates for the panels that define the airfoil geometry and multiple data files such as `alpha0.0.dat`, `alpha+3.0.dat`, etc. that contain pressure coefficient data for one angle of attack such as 0.0 degrees, +3.0 degrees, etc. as encoded within the filename.

A quick inspection of the first 5 lines of the data files with the unix `head` command shows:

```
$ head -n 5 naca0012/xy.dat
NACA 0012
    1.000000      0.1260000E-02
   0.9916770      0.2421811E-02
   0.9803640      0.3982079E-02
   0.9672594      0.5762782E-02
$ head -n 5 naca2412/alpha+3.0.dat
#      Cp
     0.31755
     0.18812
     0.12934
     0.07974
```

The first line of the `xy.dat` file contains the name of the airfoil, followed by the lines containing the x and y coordinates for each of the points that define the panels. Points start at the trailing edge (approximately $x = 1.0$), proceed along the upper surface to the leading edge (approximately $x = 0.0$), and then along the lower surface back to the trailing edge again (counter-clockwise around the airfoil).

For the pressure coefficient files, the filenames should be of then form `alpha<angle of attack>.dat`, e.g. the `alpha+3.0.dat` has an angle of attack of +3.0 degrees. Note that your program should robustly handle input files with differing angles of attack. Within each file the first line is a header and then each line contains the pressure coefficient for one panel. The first value is for the first panel (defined by the first and second points), the second value is for the second panel (defined by the second and third point), and so on. So given the order of the points the panels start at the trailing edge, go along the upper surface to the leading edge, and then back to the trailing edge.

You may find the `glob.glob()` function to be useful for determining which pressure coefficient files are present in the data directory, and the functions from `os.path` for working with file paths, and testing for the presence of directories and files:

https://docs.python.org/3/library/os.path.html

Disallowed Imports

Please do not use any module from the following set: `csv`, `numpy`, `pandas`, `tabular`, `statistics`. If you have a question about whether you may use a particular module, please post on Canvas. In general, we'd like you to implement your solution using the fewest dependencies as possible; we recommend sticking with `glob`, `math`, `os`, and `sys` for this assignment.

Error Handling

This is the first assignment that we ask you to consider handling runtime errors gracefully. The program we ask you to write depends on a set of well-formatted input files being present. What happens if (some of) these files are non-existent or malformed, can your program handle such cases? What if the input-directory is specified with or without a trailing delimiter? We don't ask that you bullet proof your program against the universe of possible errors, but we do ask that you consider a few different cases of errors or variations in inputs that could arise, and write (error handling) code accordingly.

Performance

The computation that is required for this assignment is quite trivial, and your programs should run in a split second for any set of input files we provide. You do not need to consider how your program would scale to handle larger inputs.

Learning Goals

This assignment packs in a lot. We will not only implement our own module and class, complete with a private method, but we will also consider how to incorporate error handling into our program. After the assignment, we will have a stronger appreciation for what it means to decompose a complex program into reliable and robust individual pieces.

Writeup

In your `hw3/README` file, provide a description of your code design to someone who has never read this assignment handout. Be sure to explain how the design illustrates key OOP concepts such as abstraction, decomposition, and encapsulation. Also describe what error checking and exception generation you incorporated into your class implementation.

Checklist & submission

The following files must be present in the `hw3` directory of your CME211 GitHub homework repository by the deadline:

- `README`: text document with answers to questions from the assignment. You may also call this file `README.txt` or `README.md` (if you want to use Markdown formatting).
- `main.py`: the Python script listed above to drive the program. Note you must use `main.py` exactly as provided.
- `airfoil.py`: Python module containing your `Airfoil` class definition.

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 hw3 tag with the following command:

```
$ git tag -am "HW3 Submission" hw3
```

or

```
git tag -m "HW3-Submission" -a hw3
```

The quotes contain a tag message. In the above command hw3 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 hw3 # delete the hw3 tag pointing to old commit

$ git push --delete origin hw3 # delete the remote hw3 tag

$ git tag -am "My HW3" hw3 # create hw3 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:

- **Do not commit `naca2412/` or `naca0012/` data directories or other data files to your repository. We will deduct points if extraneous data files are present in your repository.**
- Take care to follow the file and directory names exactly. Everything is case sensitive.
- Please avoid committing other data or temporary files to the repository. We only want to see your `README` file and Python code.
- 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.