# ME 498 K; Spring 2023 — Homework 4

Aidan Hunt (ahunt94@uw.edu)

Due: April 28, 2023

In this homework, you will practice creating, manpipulating, and plotting NumPy arrays. Post questions about this assignment to the class discussion board for the fastest response. Submit your responses to problems 1) and 2) as separate files.

## 1. Airfoil Artist

The general equation for the coordinates of a symmetric NACA airfoil with zero pitch and leading edge located at $(x, y) = (0, 0)$ is given by:

$$y = \pm 5ct[0.2969\sqrt{x/c} - 0.1260(x/c) - 0.3516(x/c)^2 + 0.2843(x/c)^3 - 0.1036(x/c)^4] \tag{1}$$

where $c$ is the chord length (i.e., the distance from the leading edge to the trailing edge), $x$ is the position along the foil chord line (0 at leading edge, $c$ at trailing edge), and $y$ is the half-thickness of the foil at a given $x$ position. $t$ is the maximum thickness normalized by the chord length, and is specified by the digits in the symmetric airfoil name. For example, for a NACA 0018 airfoil, $t = 0.18$ because the maximum thickness of the airfoil is 18% of the chord length.

In this problem, you will write a Python script that computes and plots the coordinates of an arbitary number of NACA airfoils, each with a specified chord and pitch angle. You will use NumPy arrays, and should consider NumPy functions that will help you build and manipulate these arrays.

### Output Requirements

You should define a function, `computeAirfoils()`, which takes name(s) of NACA airfoil profiles, the corresponding chord length(s), and the corresponding pitch angle(s) (in degrees), and produces the following output:

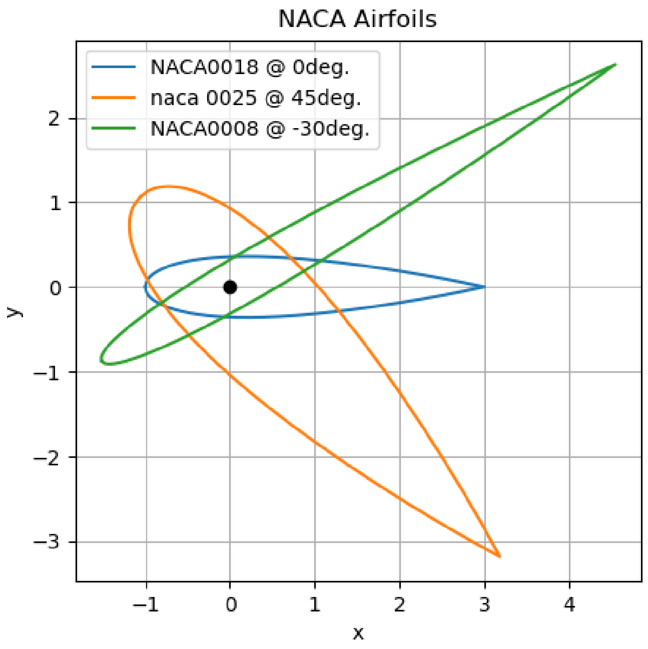
1. **Returns** a *single* 3-D NumPy array of floats that contains the *full* profile coordinates of all airfoils. Each "page" of the array should represent a single airfoil. For each airfoil's "page", the first row should represent the $x$ coordinates of that airfoil, and the second row should represent the $y$ coordinates of the airfoil. The quarter-chord position of each airfoil should be located at $(x, y) = (0, 0)$.

2. Produce a single plot that overlays all airfoils, with a legend that indicates the profile and pitch angle of each foil. (Note: the figure itself does not need to be returned, only generated. But you may return it if you wish).

For example, the following code, which specifies the properties of 3 airfoils, should produce a $2\times N\times 3$ NumPy array of floats, where $N$ is the number of coordinate points for each airfoil (you may choose any value of $N$, as long as it yields a smooth plot):

```
### Main body of the script ###
# Define properties of several airfoils
profileNames = ['NACA0018', 'naca 0025', 'NACA0008']
chords = [4, 6, 7]
angles = [0, 45, -30]

# Call function to compute coordinates and plot airfoils
foilCoords = computeAirfoils(profileNames, chords, angles)
```



In addition, by running `computeAirfoils()`, the plot above is produced. Note that the quarter-chord positions of all airfoils are located at $(x, y) = (0, 0)$, that a "positive" value of pitch corresponds to the leading edge pointing upward, and that the legend indicates the profile and pitch angle as provided to the function. Your plot should match the characteristics of that shown above: **it should have axis labels, a title, a grid, a legend, and have an "equal" aspect ratio** (e.g., one unit of $x$ has the same length as one unit of $y$, as in the real world). However, otherwise it does not need to be fancy (for example, you do not need to specify legend positioning, nor use a non-default color scheme), as the intent is mainly for you to practice basic plotting syntax.

## Input Requirements

Your `computeAirfoils()` function should be flexible in that it can accept the specifications of an arbitrary number of airfoils. These airfoil specifications should be provided as lists of values of the appropriate type. **However**, your code should also be able to handle the case of a single airfoil, where single values (rather than lists) may be provided. For example:

```
foilCoords = computeAirfoils('NACA0018', 4, 6)
```

For any number of airfoils, the profile names provided may or may not be uppercase/lowercase (e.g., 'NACA0018' vs 'naca0018' vs 'nAcA0018'), and may or may not have a space in between 'NACA' and the specification digits (e.g., 'naca0018' vs 'naca 0018'). Consider how `str` methods can help you simply address these cases.

Additionally, your `computeAirfoils()` function should also be able to handle certain incorrect or incompatible inputs, and throw errors accordingly. To summarize, your function should be able to enforce the following conditions on its inputs:

1. The number of profile names, number of chord lengths, and number of pitch angles provided to `computeAirfoils()` must all be equal. Otherwise, an `IndexError` should be thrown.

2. The chord length(s) and pitch angle(s) must be specified as numeric values or as `lists` of numeric values. The airfoil profile name(s) (e.g., NACA 0018) must be specified as a `str` or a `list` of `str`. If values of other types are provided for these inputs, a `TypeError` should be thrown. You may assume that if an input has the correct type that its value is otherwise valid.

You can raise a particular type of error via the following syntax:

```
if (condition): # If we want to throw an error
    raise TypeError('Message that will be printed when error is thrown')
```

I recommend defining helper functions to handle these error cases, and calling them at the beginning of `computeAirfoils()` (see below). To check if a variable is of a particular type, use the `isinstance()` function. For example, to check if `x` is a list: `isinstance(x, list)`. Notice that we are providing the type name directly as `list`. To check if a value is numeric (rather than whether it is specifically an integer or a decimal), you can import the `numbers` module and use `numbers.Number` as the type name. Also, just as for the `isinstance()` function, you can pass type names directly to functions that you define. You should use this to develop a single, generalized type-checking function that you can use for each input.

```
def computeAirfoils(profileNames, chords, angles)
    '''
    [docstring here]
    '''

    # Check inputs for correct typing
    anyOutputs = checkType(profileNames, additionalParameters...)
    anyOutputs = checkType(chords, additionalParameters...)
    anyOutputs = checkType(angles, additionalParameters...)

    # Check inputs for size compatibility
    checkSizes(profileNames, chords, angles)

    # The rest of your function
    ...
```

## Style

As always, your script will also be graded on whether it is well-structured and follows the Style Guide. All of the usual guidelines apply: you should factor out distinct and/or repetitive tasks into functions to improve organization and reduce redundancy, and **every** function must have a docstring that describes its parameters, returns, and other behavior. As we discussed in class, when working with multi-dimensional numpy arrays, it is important to describe what each dimension (e.g., the rows, columns, pages) represents in the docstrings of any functions that use them as parameters or returns.

As we have practiced in lecture, use pseudocode to outline the problem and break it down into smaller pieces. This will help you identify the "steps" of the problem, including sub-tasks that are well represented by helper functions. If you are unsure of where to start, you can also solve simple versions of the problem first, and then add complexity. For example, you could first write a script that computes the coordinates of a single airfoil, assuming inputs of the correct type and size. Then, you could generalize this to an arbitrary number of airfoils, still assuming correct inputs. Then, you could add error checking...and so on.

If you have any questions about general style requirements or the specific style requirements for this assignment, post them to the discussion board.

## 2. Creative Programming II

[Question removed]