Before you begin

- Create a file, `analysis.py`, to put your code in. All code will go in here. Get the four data files (`data1.csv` through `data4.csv`) and put them in your new project
- These are spreadsheets in csv format, and have data from a (simulated) experiment in them. We performed an experiment with something that we want to model as a mass-spring-damper system. Open up these files in a spreadsheet program, and have a look at them (plot the data). The first three are all different, and the fourth is a more realistic version of the first, with some measurement noise added in.
- Download and read Controls.pdf (available in the Canvas Homework files) Figure 14.4 should remind you of the data you plotted in the previous step. We're going to get you estimate the things on the y-axis of this graph (see Problem 1), and then use them to estimate a number of characteristics of the system (see Problem 2), and then use those to estimate the values of the mass, spring constant, and damping factor that we should use in our model (see Problem 3).
- Read Problem 4 to clarify how to structure your code; you can do problems 1-3 without the required code structure, but it might be easier to structure your code with functions from the beginning.

Learning Objectives:

The goal of this homework is to give you some experience analyzing experimental data with Python, and some more practice using the data structures and ideas we've talked about in class.

Thoughts (come back and read these after you've read the problems):

- For those of you with some controls background, these are all generated from a 2nd order linear system, and the goal is to estimate the parameters of the transfer function.
- The parameters have been defined such that they should be completely unambiguous, i.e. all answers should arrive at the exact same answer. The autograder will tell you if your answer has roughly the same magnitude as the real answer, which should help diagnose major coding errors.
- To evaluate your code, we will both import your two functions into our own testing code.
- Try to write as little code as you can for this homework. The goal is for you to use as much of the existing stuff in Python as you can. Try not to write the same code twice. For example, both of the functions we ask you to write involve reading data from a file. You should not have two blocks of code for reading from a file; try to figure out a way of only having it once (although it's clearly going to be used from at least two different places). If you're copy-pasting significant amounts of code, your code can likely be hugely improved.
- You may write as many functions as you wish, but we will only be testing the functions which we explicitly ask for. However, you should write more helper functions in addition to the ones we ask for if it helps to increase the readability of your code.
- If you do the extra credit, then you need to be able to tell the system whether or not to filter the data. You should make the default be no filtering.
- We will test your code on our own data sets, not just the ones we gave you. Don't hard-code anything specific about the test data sets into your code (such as the number of lines in the file). You can rely on our files having the same format as the ones we gave you. Your code should also not crash on any non-stationary input, i.e. even if we give you a file which is full of white noise, your code should at least still be able to process it (even if the numbers produced don't have any meaning).

Grading Rubric: [25 points]

In regards each item, please see associated footnotes for each item * † ‡

- Problem 1, helper functions
  - [1.5 points] `load_data_from_file` works * .
  - [1.5 points] `greater_than_index` works * .
- Problem 2, estimate values
  - [3 points] Estimate `c_initial, c_max, and c_final` (1 point each) † .
- Problem 3, estimate characteristics
  - [2 points] Estimate Rise Time correctly † .
  - [2 point] Estimate Peak Time correctly † .
  - [2 points] Estimate Percentage Overshoot correctly † .
  - [3 points] Estimate Settling Time correctly † .
- Problem 4, model
  - [2 points] Estimate `m`, `k`, and `c` correctly † .
- Overall
  - [2 points] Script, when run, prints out system information for data1.csv, ordered alphabetically by the key.
  - [2 point] Functions import and work as expected * .
  - [2 point] Able to work with grader data files ‡ .
- Standard Grading Checkpoints
  - [2 points] Code passes PEP8 checks with 10 errors max * .
- Extra Credit: deals with noisy data.
  - [1 point] Backwards filter function works ‡ .
  - [1 point] Analysis of parameters from data1.csv and data4.csv.

*Indicates that the autograder will tell you if this problem is correct on our set of test cases and assign you credit.
†Indicates that the autograder will verify that your data is of the right data type and that it is within 1 log2 order of magnitude; however, it will not tell you if your answer is actually correct. Full points will be assigned for getting within 1% of the correct answer. Half points will be assigned for getting within 10% of the correct answer.
‡Indicates the autograder will grade this in the background, but only tell the result after the homework has been published.

Problem 1: Helper functions

~~To start out, we're going to write some helper functions which will be used for later parts of the code. We will not tell you exactly how to use them, but you will find that for certain problems, they will make your life a lot easier. All code should be in a file called `analysis.py`.~~

- Write a function that takes a file path as a string, `load_data_from_file`, and returns two lists: the first list should be a list of all the time indexes, and the second list should be a list of all the values as Python floats.
  You may assume the following properties of the CSV file:
  - The CSV file always has the same first header row
  - It has at least three numeric rows with distinct values (i.e. the signal will not be constant), and two columns of equal length
  - The final position value will always be greater than the initial value
  - The time values are ordered in ascending order and start at 0.

  You should not assume anything else about the position values.
- Write a function called `greater_than_index` which takes in 2 arguments: A list of numbers, and another single number. The function should return the position of the first element in the list which is greater than or equal to the given number. If no such element exists, you should return None.
  Some examples:
  - `greater_than_index([1, 3, 4, 7, 10], 6) ## Output 3`
  - `greater_than_index([-2.5, 1, 4, 8, 4, 1, -2.5], 4) ## Output 2`
  - `greater_than_index([1.1, 2.2, 3.3, 4.4, 5.5], 100.5) ## Output None`

Problem 2: Estimate values

Start by writing a function which computes the following:

- `c_initial` – the initial position of the system
- `c_max` – the largest position of the system
- `c_final` – the steady-state value of the system (which you can assume is simply the final position of the system)

Also write some test code which displays the result of running your code on data1.csv. inside `if __name__ == "__main__"`. You should try running your code on other files as well.

Problem 3: Estimate characteristics

Next, write a function which computes the following metrics:

- The rise time, T_r. This is the time that it takes to go from 10% to 90% of the way from `c_initial` to `c_final`.
- The "10% time" is defined as the first time at which the system obtains a value greater than or equal to the value which is 10% between `c_initial` and `c_final`. Same with the 90% time.
- Estimate the Peak Time, T_p. This is the time at which the position has the maximal value.
- Estimate the Percentage Overshoot, `percent_overshoot`. This is the amount that the system overshoots `c_final`, expressed as a percentage of the range from `c_initial` to `c_final`. Do not use the definition from the book (4.35).
- Estimate the 2% Settling Time, T_s. This is the earliest time when the current and subsequent positions of the system are within a certain threshold of `c_final`. This threshold is defined by 2% of the range between `c_initial` and `c_final`. Do not use the estimate from the book (4.41).
  For example, if `c_initial` = -1 and `c_final` = 1, the 2% threshold would be (0.96, 1.04) (non-inclusive of endpoints).
  Note that we do not require a specific function name here. All values will be checked through the function `analyze_data` (see Problem 4).

Problem 4: Estimate model values

Write a function called `get_system_params` which takes in two parameters, a percent overshoot and a settling time. It should then output the mass, spring, and damping constants, in that order.

- First, compute ζ using 4.39.
- Next, compute ω_n using 4.42.
- Finally, use equation 4.29 to get the mass, spring constant, and damping factor.
  - You can assume the mass is always 1.
  - The spring constant is the ω_n^2 term in the numerator of 4.29.
  - The damping factor is the coefficient of the linear s term in the denominator of 4.29.

Problem 5: Code structure

Write a function called `analyze_data` that takes a filename as its argument, and returns a dictionary with the following keys and the corresponding values:

- `'c_initial'`: `c_initial`
- `'c_max'`: `c_max`
- `'c_final'`: `c_final`
- `'rise_time'`: `t_r`
- `'peak_time'`: `t_p`
- `'perc_overshoot'`: `percent_overshoot`
- `'settling_time'`: `T_s`
- `'system_mass'`: `m`
- `'system_spring'`: `k`
- `'system_damping'`: `c`

Replace your previous testing code in `if __name__ == "__main__"` so that the only output now is to print out all of the above parameters in alphabetical order from `data1.csv`, in the format `key value`, e.g. `print(key, value)`. For instance, part of the output might look like:

```
...
c_max 1.0
peak_time 2.05
perc_overshoot 25.131951323
...
```

Problem 6: Extra credit

- Write a function called `backwards_filter` which takes in a list and a positive integers and performs a backwards-looking mean filter with a customizable window size n. Unlike the mean filter from Lab 2, your filter should average points at the beginning even if the filter extends off the edge of the list.
- For instance, if we have the list `[0, 1, 2, 3, 4, 5]` and `n=3`, the filter should turn the data into `[0, 0.5, 1, 2, 3, 4]`.
- Modify `analyze_data` to accept a `window_size` argument which runs the position data through the above filter before computing the parameters. It should default to no filtering, i.e. a window size of 1.

```
analyze_data('my_file.csv', window_size=5) ## 5-window filtering
analyze_data('my_file.csv') ## No filtering
```