

Name: Evan Leglar	Lab Time T 12:00 PM
--------------------------	----------------------------

Names of people you worked with:
<ul style="list-style-type: none">•

Websites you used:
<ul style="list-style-type: none">• Stackoverflow

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	8
--	---

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

Evan Leglar

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

Turn your Python code and PDF into Gradescope

BEFORE YOU BEGIN

1. Make a new project for this Homework
 - a. Create a file, **analysis.py**, to put your code in. All code will go in here.
2. Get the four data files (data1.csv through data4.csv) and put them in your new project
 - a. 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.
3. Download and read Controls.pdf (available in the Canvas Homework files)
 - a. 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).
4. **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):

1. 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.
2. 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.
3. To evaluate your code, we will both import your two functions into our own testing code.
4. 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.**

5. 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.
6. 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.
7. 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 Checkpoints

1. Problem 0, helper functions
 1. `load_data_from_file` works. [1 point]*
 2. `greater_than_index` works. [1 point]*
2. Problem 1, estimate values
 1. Estimate `c_initial`, `c_max`, and `c_final` (1 point each). [3 points]**
3. Problem 2, estimate characteristics
 1. Estimate Rise Time correctly. [2 points]**
 2. Estimate Peak Time correctly. [1 point]**
 3. Estimate Percentage Overshoot correctly. [2 points]**
 4. Estimate Settling Time correctly. [3 points]**
4. Problem 3, model
 1. Estimate `m`, `k`, and `c` correctly. [2 point]**
5. Overall
 1. Script, when run, prints out system information for `data1.csv`, ordered alphabetically by the key. [1 points]
 2. Functions import and work as expected. [1 point]*
 3. Able to work with grader data files. [1 point]^
 4. Appropriate code comments. [1 point]
 5. Appropriate code structure and use of functions. [1 point]
6. Extra credit: Deals with noisy data. [2 points]
 1. Backwards filter function works. [1 point]^
 2. Analysis of parameters from `data1.csv` and `data4.csv`. [1 point]

Remember to turn in your files and this PDF to Gradescope.

*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.

No marker indicates it will be manually graded.

Problem 0: 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`.

1. Write a function called `load_data_from_file` which takes in a file path as a string, 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.

2. 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

Comments for grader/additional information (if any)

Problem 1 Estimate values

Start by writing a function which computes the following:

1. `c_initial` – the initial position of the system
2. `c_max` – the largest position of the system
3. `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`.
(You should try running your code on other files as well!)

Comments for grader/additional information (if any)
Function returns the c-values which can be used when called inside of other functions

Problem 2 Estimate characteristics

Next, write a function which computes the following metrics:

1. The rise time, T_r . This is the time that it takes to go from 10% of the way from the c_{initial} to c_{final} to 90% of the way.

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.

2. Estimate the Peak Time, T_p . This is the time at which the position has the maximal value.
3. Estimate the Percentage Overshoot, %OS. 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).**
4. Estimate the 2% Settling Time, T_s . This is the earliest time when the current and all 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 $c_{\text{initial}} = -1$ and $c_{\text{final}} = 1$, 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).

Comments for grader/additional information (if any)
Function returns the system characteristics estimates and can be called in other functions. Settling time utilizes list comprehension iteration to iterate backwards through c-values until the last time it crosses the 2% final value threshold.

Problem 3 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 zeta 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.

Comments for grader/additional information (if any)
Function returns the values as expected

Problem 4 Code structure

1. 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': %OS`
 - `'settling_time': T_s`
 - `'system_mass': m`
 - `'system_spring': k`
 - `'system_damping': c`
2. Replace your previous testing code so that the only output now is to print out all of the above parameters in alphabetical order, in the format “`{key}: {value}`”. For instance, part of the output might look like:

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

Comments for grader/additional information (if any)

```
Console Output
/usr/local/bin/python3.7
/Users/evanleglar/Desktop/ME499_Python/Homework1/analysis.py

Final System Parameters for data1.csv:
```

```
c_final : -8.96706075522e-06
c_initial : -1.0
c_max : 0.854459357957
peak_time : 0.315
perc_overshoot : 85.4
rise_time : 0.186
settling_time : 19.959
system_damping : 0.400821684453129
system_mass : 1
system_spring : 15.954760214234753

Process finished with exit code 0
```

Problem 5 Extra credit

1. Write a function called `backwards_filter` which takes in a list and a positive integer 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]`.

2. 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
```

3. Demonstrate your output on `data4.csv` without filtering and with `window_size=5`. How do they compare to the results from `data1.csv`?

Comments for grader/additional information (if any)

analysis.py
<pre>% Copy and paste backwards_filter here</pre>

Console Output
Copy and paste the result of running your code on data4.csv with no filtering here

Console Output
Copy and paste the result of running your code on data4.csv with filter $n=5$ here

Discussion
Comment here on how using the filter affects the similarity to <code>data1.csv</code> .