PHYS 391 – Lab 2: Measurement of the Speed of Light

Key Concepts

- Statistical Inference: Mean, Standard Deviation
- Standard Deviation of the Mean
- Weighted Averages
- Systematic Errors

2.1 General Lab Introduction

All relevant data, calculations, observations, and responses to questions in the lab handouts should be recorded in your (electronic) ipython notebook. Prelab writeups are not required, although reading through the lab instructions before you begin will be quite useful. All material to be turned in should be included in the ipython notebook. Label your markdown boxes with the corresponding section number from the handout.

While it is encouraged for students to work together collecting data and understanding the concepts in these lab assignments, it is expected that each student does their own work analyzing the data and answering the questions. In particular, it is impossible to gain proficiency with writing python code by watching somebody else do it. It is a good idea to always have a prediction of what you expect for any of the tasks assigned in the labs. If you don't observe what you expect, you should think about whether this makes sense, or whether you made a mistake. You should not assume that you will always get the 'accepted' answer, however.

For this lab, a summary has been provided at the end of this write-up to help you identify all of the specific items you need to provide. In addition to these items, you will also partly be graded on providing some useful discussion in your ipython notebook related to what you are doing, observations, thoughts, analysis, and conclusions as you work through these steps.

2.2 Goals of this Lab

This lab is a python-based exploration of the key concepts of statistical inference and uncertainty in random variables. We will take multiple measurements of a physical quantity and then perform a weighted average to extract a best value for the speed of light. Some evaluation of possible systematic uncertainties will also be performed. This lab also will apply the concepts of python programming and the command line we first explored in Lab 1.

In this lab, when you are asked to calculate the mean and standard deviation of a sample of numbers, you may either use the built-in python functions or your own functions from the first lab. Make sure you understand what each built-in function is doing, however, if you use it. For the weighted average, I am going to ask you to write your own function for this.

2.3 Speed of Light Overview

The speed of light is a fundamental property of the universe, and historically the discovery that light travelled at a very fast, but not infinite speed, was a very important result. Measurements of the speed of light are always difficult, and the first credited measurement was performed in 1676 by Ole Rømer who observed the eclipse times of Jupiter's moons and used these times to correctly measure the speed of light to an accuracy of 25%. More details on this rather amazing result can be found on the wikipedia page: `https://en.wikipedia.org/wiki/R%C3%B8mer%27s_determination_of_the_speed_of_light`.

Experimentally, we often can not make pioneering measurements with high precision, and being able to extract useful data even from inaccurate tools is a sign of a good experimentalist. In this lab, we will attempt to make a measurement of the speed of light using nothing more than a computer (and the internet). Since long-distance communications now rely almost entirely on fiber optics, the time for signals to reach distant locations can be used to determine the speed of transmission in fibers, and from what we know about the index of refraction of an optical medium we can use that measurement to infer the speed of light in a vacuum (c).

Note, there are many sources of systematic uncertainty in this method. We will discuss some of them below, but even after evaluating some of these uncertainties, you shouldn't be too surprised if your result isn't in complete agreement with the accepted value. This discrepancy just serves to tell us that some of our assumptions aren't correct, and in the process we can learn something about the universe around us. The goal here is really not to make an accurate measurement of the speed of light, but rather to *try* to make an accurate measurement with the limited tools at our disposal.

2.4 Ping Times

2.4.1

The general method, which is identical to most methods to measure c, is to send a signal over some distance and time how long it takes for the signal to get there. The speed of light in the medium v_n is then given by the distance-to-time ratio $v_n = d/t$ where d is the round-trip distance to the remote site, and t is the time elapsed. This can then be corrected for the known effects of the medium to infer the speed of light in a vacuum c.

To get an idea for the kinds of distances and times we need to measure, work out the expected speed of light in a communications fiber (which are almost always made of some kind of glass) and work out an expected round-trip time to a location 1,000 km away. This calculation should go in your ipython notebook. Get in the habit of trying to identify any sources of uncertainty in a calculation such as this, and (briefly) write down a few comments about things you may not know which could impact your estimate.

If you were trying to make this measurement with a short (1 meter) and long (20 meter) cable connected between two computers in the lab (this was the original idea for this lab...) roughly estimate the timing resolution you would need to be able to measure the speed of light over these distances.

2.4.2

To measure the transmission times, we will use a network utility called `ping`. This utility sends a small message to a remote computer, which then simply bounces the message back. The `ping` command measures the round-trip transit time and prints out the result once per second. Read Section 2.11 below for some additional details about this tool.

The best way to complete this lab is to log in to `shell.uoregon.edu`, following the access instructions here: `https://service.uoregon.edu/TDClient/2030/Portal/KB/ArticleDet?ID=30853`.

Once you have logged in, test that `ping` is working by pinging a nearby server (the main OSU web server `www.oregonstate.edu` works well) and type the following.

```
ping www.oregonstate.edu
```

This should give you one output line per second continuously until you type control-C to cancel the output. Note that `ping` finishes with a helpful status line which includes the mean time and the standard deviation. You can also use something like

```
ping -c 100 <hostname>
```

to run for a fixed number of cycles (in this case 100).

A machine located at OSU should have a very short transit time, and should take less than ~ 2 milliseconds to respond. For short distances, the time is actually dominated by the latency (the time it takes the remote computer to respond or the time for intermediate switches to receive and retransmit the message).

You can get an idea of the path your signal is taking with the command

```
traceroute www.oregonstate.edu
```

which is another Unix network utility. There are likely fancy graphical applications for Windows which provide similar functionality. Many of the names returned by `traceroute` will be illegible, but there should be some obvious city names associated with any large hop as the signals travel through the main internet backbone routers.

Ideally, you will use a machine where your `ping` command gives results with a resolution of at least 100μs or better. Very old machines were limited to 1 ms precision, and this just won't work well, as all of your `ping` times will end up being identical and you will be unable to assess your statistical uncertainty.

2.4.3

Collect a series of data files where each file contains ping times to a single academic institution somewhere around the country or the world. Each file should have on the order of 100 independent measurements, most easily set by using the `-c 100` modifier to `ping` shown above. You should pick 5–10 servers including some that are close by (which can be used to measure the intrinsic latency) as well as servers that are far away. Academic institutions are good, as their servers are almost always located on campus (which is not true for most companies) and almost all institutions have a web server with an address of www.something.edu which responds to `ping`. This statement is less true now than it used to be five years ago as more sites (including `uoregon.edu`) block ping access for security reasons. Institutes up and down I-5 are particularly good, as the fibers tend to generally follow the freeway route, and you can use the driving distance to get a pretty good estimate of the fiber distance.

Long distances are good if you can figure out the route. Sydney is particularly nice, as there is an undersea cable which goes from Los Angeles through Hawaii directly to Sydney, although other routes are possible as well. New Zealand, Hawaii, and Tokyo are also probably worth trying, although it may take some work to find servers that respond. If you look online, you can find main undersea cable routes which could be helpful. Some possible candidates are in Table 1 below, but be skeptical about the observed ping times. More and more, web sites are being hosted in remote data centers, or cached in other sites, so if the time seems way off, check the route carefully. Feel free to use Google to find the address of your own favorite institution, but it is worth checking first that you can figure out reasonably well the route the `ping` traffic is taking. Not all sites are equally suitable for this experiment.

Table 1: Addresses of a few academic web servers that respond to `ping`.

`www.oregonstate.edu`
`www.washington.edu`
`www.berkeley.edu`
`www.ucla.edu`
`www.cern.ch`
`sydney.edu.au`

To get the `ping` output into a text file, either copy-paste from the command window, or else use the Unix 'redirection' facility to save the output directly to a file. One nice way to do this is to use the `tee` command, which will both print the `ping` output to the screen as well as save it to a file. This way you can check that the `ping` command is working properly as the data is being recorded. The syntax for this is shown below.

```
ping -c 100 www.somewhere.edu | tee myfile.txt
```

The vertical line is the Unix 'pipe' command which sends the output of `ping` to the input of `tee`. This pipeline syntax works for any Unix command.

Note that the very last line `ping` outputs will give the mean and standard deviation of the times. It is worth noting this down in your ipython notebook so that when you recalculate these later you can check that you get something similar. You should also absolutely write down the *time* when each data file is recorded. The time of day can make a significant difference on latencies and in fact taking the data later at night or on the weekend could possibly give a significant difference. Testing a single site at various times of the day could actually be a very good way to evaluate systematics associated with varying latencies or other effects which are unrelated to the speed of light.

If you are logged into `shell.uoregon.edu` when you write your data to a file, you will need to transfer your file back to your own machine. Once you exit your ssh session (type `exit`), you can use `scp`:

```
scp DuckID@shell.uoregon.edu:~/myfile.txt .
```

(note the final dot, which copies your file to your current working directory). The portion of the above command after the colon gives the path to your file.

2.5 Ping Time Analysis

2.5.1

For each site you select, we want a best estimate of the time it takes a signal to travel that distance. You can either use the round-trip time and double the one-way distance, or use half of the round-trip time and estimate the one-way distance. Make sure it is clear in your ipython notebook which specific thing you are trying to measure.

The hardest part of this analysis is parsing the raw `ping` output into a list of numeric times. Ping output lines look something like the following, although this will vary slightly between different operating systems:

```
64 bytes from 169.229.216.200: icmp_seq=0 ttl=50 time=34.788 ms
64 bytes from 169.229.216.200: icmp_seq=1 ttl=50 time=34.394 ms
64 bytes from 169.229.216.200: icmp_seq=2 ttl=50 time=36.421 ms
64 bytes from 169.229.216.200: icmp_seq=3 ttl=50 time=34.334 ms
64 bytes from 169.229.216.200: icmp_seq=4 ttl=50 time=34.340 ms
```

This is always a tradeoff between spending a lot of time designing a very clever, flexible solution and just finding a quick solution that barely works. One could just copy these numbers by hand into a text file using a text editor, but this is probably completely unacceptable for hundreds of lines across many files. A global search-and-replace to just eliminate all of the unchanging text in the file is one viable option however. There are sophisticated Unix command-line tools like `sed` which can parse complex patterns out of text files, but it would probably take hours to figure out the syntax. Unless you already have experience with these tools, this 'fancy' solution is also probably unacceptable. One pragmatic possibility is to use something like python's `split` command knowing that the time is usually in the same place in the output line. Because the numeric times are likely separated by both spaces and an equal sign, you may need to first delete all the equal signs from your files, or else do the parsing in two steps.

There are simpler Unix tools that can do enough simple text parsing to get the job done with only a little effort, for instance the `cut` command. You can get information on `cut` (or any Unix command) by typing `man cut` (`man` stands for manual), or just look it up with a web search. With the following syntax, `cut` uses an equals sign as a delimiter and selects the fourth delimited column from the file `myfile.txt`:

```
cut -f 4 -d "=" myfile.txt
```

This reduces my original text down to

```
34.788 ms
34.394 ms
36.421 ms
34.334 ms
34.340 ms
```

Now I could follow this with `cut -f 1 -d " "` to pick everything before the first space, which is the number I want. I can then use `tee` to redirect this into a new file. I can run all of these at once using the Unix pipe construction, and the entire command looks like this:

```
cut -f 4 -d "=" myfile.txt | cut -f 1 -d " " | tee mydata.txt
```

Note that this doesn't work on a Windows machine, but it will work on a Mac through the Terminal, as well as on any Unix/Linux machine. There may be a few extra lines at the start and end of each file that you need to clean up by hand, but it shouldn't take you more than about 10 minutes to get all of your data ready to read into python.

2.5.2

For each location, you should read the raw data in, compute the mean and standard deviation of this data sample, and use those values to get a best estimate for the mean time (expressed as $\bar{t} \pm \delta\bar{t}$). You should also produce a histogram of t and check whether the data plausibly looks like it has come from a single parent distribution, or whether there are 'outliers' that clearly don't appear to belong. Make sure to use enough bins in your histogram so you can really see something (one bin per 100μs is probably a good goal). Can you think of reasons why certain ping times might be anomalously long or short? If necessary, describe some

criteria for 'cleaning' your data, such as Chauvenet's criterion described in Taylor Chapter 6, and apply it to your times before calculating your final best time. You should only need to do this in extreme cases, and I am not encouraging you to use Chauvenet's criterion, but if you have a few points which are clearly way off, this is arguably a justification for removing them. If you have some other funny looking distribution, you probably need to think about how to handle this. To select elements out of an array based on some condition, the easiest method is to use the `find` command to index the array elements. So to get all elements of `x` which are greater than zero, use: `x(find(x>0))`. This will return a new array with the conditional expression `x>0` applied to each element of `x` in turn.

Once you have final numbers, make a table in your logbook for each location and give the final average time including an uncertainty. In this case, the uncertainty is only a statistical uncertainty based on the random variations in times observed. We will think about possible systematic uncertainties later. Leave a column in your table for the distance, which we will determine next.

2.6 Distance Analysis

This is probably the most inaccurate part of this entire measurement. For each site, you need to determine an estimate of the distance and an uncertainty on that estimate. Use your own judgement and *describe your reasoning* in your notebook. There is no right answer here, although there are plenty of wrong answers. Use `traceroute` or some similar program to try to estimate the actual path, and make some estimate of the distance and possible uncertainty, especially in terms of how much extra cabling there might be within a city. Los Angeles, for instance, is a big place, so knowing that a router is in LA still carries a significant uncertainty. Use Google or any other mapping utilities which you can find and try to do as good of a job as you can here, although in the end you will still likely have significant uncertainties on each distance. These uncertainties don't all have to be the same however, and after justifying your estimates in your notebook, complete your table with distances to go with your ping times.

One possible method for reducing this uncertainty is to ping intermediate routers displayed in the traceroute output. You should usually be able to find a router very close to the start and end of a long fiber run, such as an trans-pacific cable. These routers are usually very close to where the cable enters the water, which can minimize uncertainties about location.

2.7 Speed of Light

We now have all the raw data we need, and we need to determine a method to combine these to give a best value of the speed of light in vacuum. I always prefer to work with values close to the actual measured values and only apply significant corrections at the end. Philosophically, I find it more honest to first quote the quantity that was actually measured, and then apply any additional assumptions or interpretations at the end of the analysis. In that spirit, you should work out a way to measure the speed of light in fiber v_n, and only once you have a final result for that value should you worry about converting this into c.

2.7.1

The most obvious method to determine v_n is to make a plot of distance vs. time and fit a line to that data. The slope of the line (assuming there is a linear relationship) will be the speed. This method has the added advantage of naturally taking out any offset of the line from the origin and giving a direct measurement of the slope. Why might there be a non-zero ping time, even at a distance very close to zero? Evaluate whether your distance measurements or time measurements are more uncertain and provide error bars for the most uncertain variable using something like the `errorbar` function in `matplotlib`. Make this plot and include it into your ipython notebook. (A fit would be best, but if you can't get this to work, you can overlay an estimate of a best-fit line by adjusting the slope and intercept by hand.) *Do not* connect the individual data points with lines.

Write down any observations you might have, especially if there are obvious trends in your data, such as anomalous points, or a different trend for the overseas sites from the US-based ones. If you really have holes in your coverage, you might think about picking one or two more sites at particular distances to help fill in your graph. Do all of the data support a linear relationship between distance and time? If not, can you think of any reasons why?

2.7.2

Each site should give you an independent measurement of v_n. A method for combining the data which doesn't involve fitting a line is to evaluate these separate measurements then combine these taking into account their uncertainty as a weighted average. As we will see later in the quarter, aside from the issue of any possible offset at zero, these two methods are equivalent.

First, decide whether there is any substantial latency in your time measurements, and either come up with a strategy to correct these (don't forget error propagation) or else just treat this as an additional source of systematic uncertainty and possibly add a systematic uncertainty in quadrature with your statistical uncertainty on t if it appears to be significant.

If you are really doing a careful job, you might try to estimate the 'latency-per-hop' and correct different sites according to the number of routers encountered along the path, although I am a bit skeptical that this would make too much of a difference for long routes.

So explain what you have chosen to do in analyzing your data, and calculate a single value of v_n and a total uncertainty for each site you pinged. Don't forget your error propagation since you will have an uncertainty on both the transit time and the distance. You can either do this calculation in python, by hand, or any other method you choose, although I would encourage you to start with your measured distance-time data (including uncertainties) in a single text file, one line per site. Read this file into your Jupyter notebook and do the entire analysis there. Please also upload this file to Canvas. The end result should be a table with individual v_n values and uncertainties for each site.

2.7.3

Make a second plot, similar to the one made in Section 2.7.1, but showing v_n vs. distance. Here the error bars should reflect the total uncertainty. It is probably worth commenting on whether these v_n values appear to be consistent with each other. This is the same information as went into the plot in Section 2.7.1 so these two plots should be consistent with each other, but trends may be more apparent here. If your uncertainties have been evaluated correctly, these all should be more or less consistent (within Gaussian probabilities). If there is more scatter in your measured values than the uncertainties would predict, that is a sure sign that there are other uncontrolled systematics lurking in the measurement (or you have made a mistake). You don't have to be quantitative here, but make a rough assessment of the agreement to demonstrate (or refute) the accuracy of your errors. If a single site is way off, consider whether you should exclude that point from the measurement, and provide some plausible explanation for why this may not be consistent to justify your decision.

2.7.4

Once you have your v_n values, combine them using the rules for performing a weighted average from Taylor Chapter 7. First, write down the actual formulas that you need to use, then write a python function that will take two input arrays (one with values and the second with uncertainties), performs the weighted average calculation, and also returns the uncertainty on the weighted average. Finally, put it all together into a code snippet which performs the weighted average calculation from an input data file (either time-distance data, or v_n data) and prints out the final result (plus uncertainty).

2.8 Interpretation

Finally, apply any additional known corrections (including possible uncertainties) to evaluate the speed of light in vacuum. How well does your answer agree with the 'expected' value? You should certainly be within an order of magnitude of the correct answer, although you may not be consistent within your quoted uncertainties.

Remember, what you have actually measured is the 'speed of light on the internet' and you have made assumptions about how you can relate that to the speed of light in a vacuum. Discuss whether it is possible that you have underestimated some of the uncertainties you have identified, and in particular think about whether some of these uncertainties might push the final measured value in one direction or the other preferentially. Also, try to think of other effects that have not been previously considered which might effect this measurement. In a real experiment, where the 'correct' answer is not known, this reflection on the result is important to identify additional systematics that ideally can be constrained with further measurements.

Write a paragraph or so discussing these issues, as well as any additional things that you might chose to do differently if you did this over again.

2.9 Final Thoughts

The relative precision of a random process improves as $1/\sqrt{N}$ since the standard deviation of the mean scales with this factor. In this measurement, you can easily take enough data to make the statistical uncertainties negligibly small, which then leaves you with systematic effects which need to be evaluated. Systematic effects which involve some level of variation can additionally be constrained by making multiple measurements, either of the same thing over and over again, or in different systems which should give the same answer. You have done this here by making multiple measurements at different distances, and you could have also repeated the measurements at different times of the day to further assess systematic variations (perhaps you did)!

The most difficult uncertainties to assess are systematic uncertainties which lead to inaccurate results. Understanding whether there is a fixed scale error throwing your measurement off can be maddeningly difficult to assess, and this is why the scientific method depends on others being able to reproduce your results, particularly with different techniques. If two measurements give inconsistent results, there is likely something wrong either in the measurements themselves or the interpretations of those measurements. This is why almost no discovery is really believed until it has been confirmed independently.

2.10 Required Elements

This list summarizes the minimal set of required elements you should provide for this lab. This is intended to help you identify all of the questions posed in the lab instructions. Please include any descriptions/explanations requested in *labeled* markdown boxes directly in your ipython notebook. Upload both your notebook (ipynb) and your data file (txt format) to canvas.

- Sec. 2.4.1: Expected speed of light in fiber, expected round-trip times, necessary time resolution to make this measurement using a 20 meter cable
- Sec. 2.4.3: Description of data acquisition
- Sec. 2.5.1: Explanation of what you are measuring, description of data parsing
- Sec. 2.5.2: Histograms of t for at least a few sites, discussion of outliers and data cleaning (if needed), table with at least times and (statistical) uncertainties
- Sec. 2.6: Describe your distance analysis and make some estimate of the uncertainty
- Sec. 2.7.1: Plot of distance vs. time
- Sec. 2.7.2: Table of v_n values with uncertainties
- Sec. 2.7.3: Plot of v_n vs. distance, discussion of consistency
- Sec. 2.7.4: Weighted average combination of v_n
- Sec. 2.8: Discussion of results

2.11 Additional Information on `ping`

2.11.1 Overview

Ping is the name a command-line network utility which was initially written for Unix to be able to tell if a remote server was online and reachable. Over time, ping has turned into a verb which is even used in common conversation to indicate the process of checking in on somebody. Anyone who has played online games has also probably worried about their 'ping times' and the lagging associated with slow responses.

2.11.2 Operating Systems

Because ping is really a Unix utility, and the Mac OSX is fundamentally based on a variant of Unix, ping and other related utilities like traceroute work most seamlessly on Unix or OS X. Because ping is so useful, however, Windows has incorporated this into the operating system ages ago, and there are doubtlessly many different graphical ping and traceroute applications available for free on Windows.

To run ping and other command-line tools on a Mac, you need to open the Terminal application, which can be found under Applications/Utilities. On Windows, the equivalent tool is the 'Command Prompt' which can be found under Accessories in the Start menu.

2.11.3 Oregon Network

Originally, this lab was intended to be run on campus, on a computer that is connected to the Oregon network. The reason for this is that academic institutions in the US share a 'private internet' called Internet2 which has higher intrinsic bandwidth and less traffic than the long-haul fibers used by commercial providers. Making connections between universities should in principle give considerably more reliable results for this measurement where we want to minimize any additional delays which we might misinterpret as a slower speed of light. We recognize that you may not be able to log into `shell.uoregon.edu`, so please just record what you do and do your best! If you are off-site, note that commercial internet service providers often send their traffic all over the place and make it very difficult to figure out the actual route taken by internet traffic. It also may be difficult to find a site to ping which is a short distance away, depending on your current location. Do your best to estimate/explain the potential impacts of this on your measurement