Total Solar Eclipse Light Curves
A Bash Data Munging Project

tl;dr

You write a Bash shell script to attach a timestamp to every line in the supplied data file, and then make a graph of some of the resulting data.

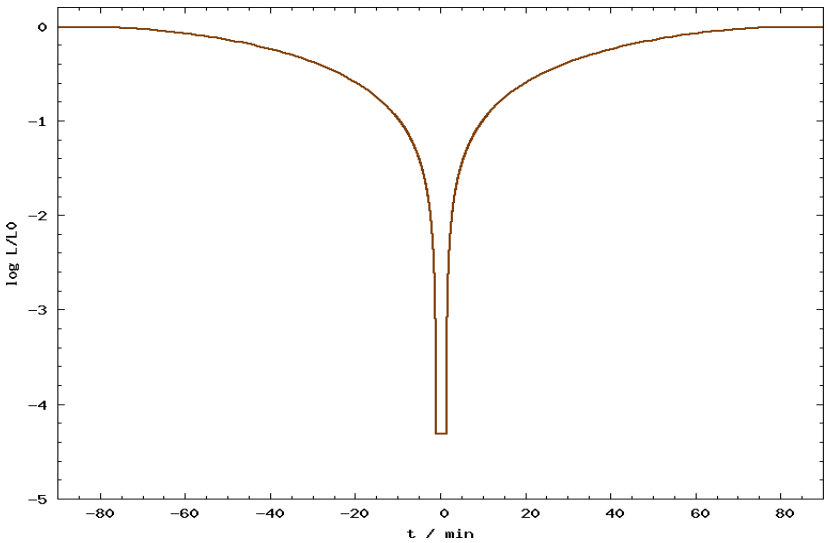
Background

My wife and I watched 2017's total solar eclipse from the parking lot of Piedmont Technical College in Newberry, South Carolina. It was fun, the weather was clear by the time of the eclipse, we saw the corona, met some nice people, and the whole experience was great.

Illustration 1: Professor Reed, aka "Eclipse Man" dons protective gear for eclipse viewing

Since I love to tinker around with microcontrollers such as Arduino, and as something of an amateur astronomer, I set up an experiment to measure and record the change in the sun's brightness as the eclipse progressed. Astronomers call these graphs *light curves*. Any eclipse produces them and in fact, observing the light curves of distant stars is one way of finding out if they have planets.

Here's an example light curve for a total solar eclipse:



Notice how quickly the light level changes. You can see this in the last panoramic picture I was able to get during the eclipse.

Even though this image was taken only four minutes before totality, the sky and the parking lot at Piedmont are still pretty light. Four minutes later, we were standing in the dark in what looked like moonlight; it was that quick. (the pink tone is because this is a Near Infra-Red image, to better show the clouds).

You can see the effect in the graph and I bet you can see it in the data I collected, too.

The challenge

I collected the data, and I want to make a graph of it, like the theoretical light curve above, but there's a problem. In my haste and excitement, I forgot to note when the data collection started. The Arduino program measures temperatures and light levels (more on that in a minute), once per second, and it stamps a relative timestamp on that observation. The timestamp is the number of milliseconds since the program was launched.

Fortunately I also know what time the program ended, thanks to the Linux timestamp on the file. So working backwards from the end time, I was able to get the beginning time.

Here's the beginning data: (used the head command)

```
10510707,PV1,1,753.00,PV2,2,129.00,TS1,5,114.13,TS2,7,97.70,WWVB,0,213.00
10512621,PV1,1,753.00,PV2,2,130.00,TS1,5,114.57,TS2,7,97.70,WWVB,0,212.00
10514536,PV1,1,752.00,PV2,2,128.00,TS1,5,114.69,TS2,7,97.70,WWVB,0,212.00
10516450,PV1,1,752.00,PV2,2,129.00,TS1,5,114.80,TS2,7,97.70,WWVB,0,211.00
```

and here's the ending data (used the tail command)

```
20067422,PV1,1,700.00,PV2,2,89.00,TS1,5,117.39,TS2,7,96.80,WWVB,0,198.00
20069336,PV1,1,700.00,PV2,2,90.00,TS1,5,116.94,TS2,7,96.80,WWVB,0,198.00
20071248,PV1,1,700.00,PV2,2,90.00,TS1,5,116.94,TS2,7,96.80,WWVB,0,198.00
20073161,PV1,1,700.00,PV2,2,90.00,TS1,5,116.94,TS2,7,96.80,WWVB,0,198.00
```

By my calculation, the timestamp on the first line should be:

```
Mon Aug 21 13:04:42 EDT 2017,10510707,PV1,1,753.00,PV2,2,129.00,TS1,5,114.13,TS2,7,97.70,WWVB,0,213.00
```

and the timestamp on the last line should be:

```
Mon Aug 21 15:44:04 EDT 2017,20073161,PV1,1,700.00,PV2,2,90.00,TS1,5,116.94,TS2,7,96.80,WWVB,0,198.00
```

That covers the first and last lines, *but to make a graph with time as the X axis, and brightness or temperature as the Y, we need to timestamp every observation.* That's where the script you write comes into play.

The data

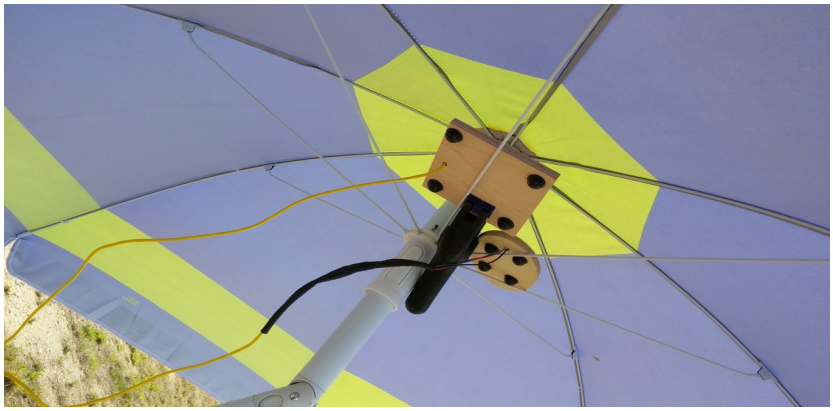
The file consists of 5,000 lines, each ending with a newline character. The fields in the data are separated by commas. In general, this format is called CSV for Comma Separated Values. It is very common and easy to import into Excel (but you must do this in Bash).

Each line in the file contains two brightness levels and two temperatures for that second. Here's the format:

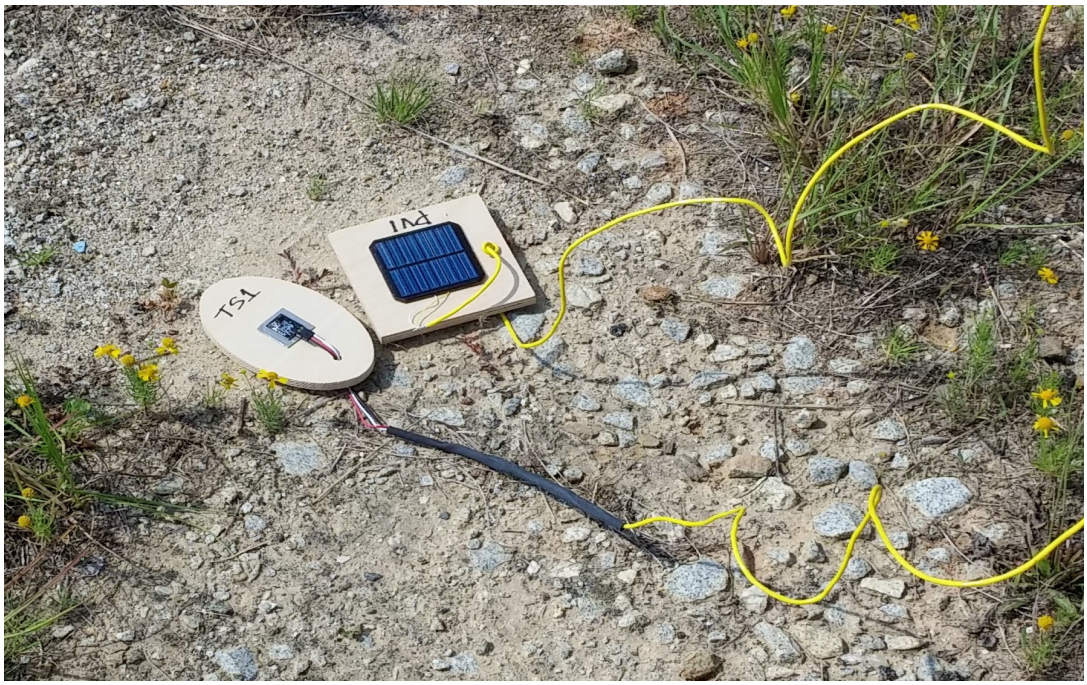
Field Number	Meaning	Example
1	Timestamp in milliseconds	10510707 – this means this line was collected 10510707 / 1000 = 10510.7 seconds = 175.2 minutes after data collection began. (I chopped off some data from the morning when nothing was happening)
2	Label for brightness 1	PV1, for Photovoltaic cell (solar cell) #1. This cell was located in direct sunlight.
3	Pin number on Arduino for first solar cell	1 = Analog input #1
4	Measured brightness for full sun solar cell	753.00 – arbitrary units, larger numbers are brighter light. This will go all the way to zero during totality.
5	Label for brightness 2	PV2, for Photovoltaic cell (solar cell) #2. This cell was located in direct the shade of my beach umbrella
6	Pin number on Arduino for second solar cell	2 = Analog input #2
7	Measured brightness for full sun solar cell	129.00 – arbitrary units, larger numbers are brighter light. Readings in the shade are lower than those in the full sunlight, of course. This too will go all the way to zero before and during the total portion of the eclipse.
8	Label for temperature 1	TS1 for temperature sensor #1. This sensor was located in direct sunlight.
9	Arduino pin for TS1	5 = digital pin 5
10	Measured temperature in degrees Fahrenheit for temperature 1	114.13 degrees F.
11	Label for temperature 2	TS2 for temperature sensor #2. This sensor was located in the shade of my beach umbrella
12	Arduino pin for TS2	7 = digital pin 7
13	Measured temperature in degrees Fahrenheit for temperature 2	97.70 degrees F.

Field Number	Meaning	Example
The remaining fields can be ignored.		

This image shows how PV2 and TS2 were mounted under the shade of the umbrella (they are on top of the wood, facing the fabric of the umbrella):



And this image shows TS1 and PV1 on the ground in direct sunlight:



Method

Some ground rules:

You may use Bash shell commands, including awk if needed. Yes, one could write a Python or Java program to do the data munging here – but the point is that this class is about shell scripting so that is what we need to use here.

Step by step:

1. Read the data file into your script, line by line. See the link below for some details:

https://stackoverflow.com/questions/10929453/read-a-file-line-by-line-assigning-the-value-to-a-variable

2. Inside your read loop, get the first field in the CSV, the milisecond timestamp. Lots of ways to do this, I like the cut command, personally.

3. "Remember" this very first millisecond value in a variable, call it first if you want. You'll be calculating how long after this time each line was recorded. This first number corresponds to Mon Aug 21 13:04:42 EDT 2017. (You can drop the day, and the date and just use the times if you like. We know it all happened on the same day).

The Bash date command has an option to convert a human-readable date, like "Aug 21 13:04:42 EDT 2017", into a Unix epoch date – the number of seconds since January 1, 1970. This number can be used to calculate a new date for each line, by adding the seconds (milliseconds / 1000) to it, and converting back to a human-readable date, again using the date command.

4. For each line after the first, calculate the milliseconds from the current line minus $first to get the time difference. You can use the expr shell function for this, or use bc. Remember that milliseconds are 1,000th of a second, so all your times must be divided by 1,000 to get seconds. Add the time difference in seconds to the Unix epoch start time calculated in step 3 above, and convert back to a human-readable timestamp.

https://www.shell-tips.com/2010/06/14/performing-math-calculation-in-bash/

5. Use Bash's printf or echo commands to write out your calculated timestamp, followed by a comma, followed by the original line.

6. And this is the easy part – use any of the techniques from your textbook to plot a chart of any two of:

a. the brightness by time (PV1 or PV2)
b. the temperature by time (TS1 or TS2)

Make sure the X axis is time and is readable. The Y axis is the light or temp as you choose. If formatting the time on the X axis is a problem, then try different date formats. Remember, you only need the hour and minute, not the day or date or seconds on your chart.

Hints and Tips

Try out the various commands you need interactively on the Bash command line – that's a plus for Bash development, by the way. No need to launch a compiler, just type in the command you want to try.

Use the up arrow key to replay your previous command and use the left and right arrow keys, along with the backspace key to edit your commands before replaying them.

Use man to find out more command options. If you really can't figure out what to look for with man, try the aprops command to get a list of manpages containing some string.

You can also run most commands and specify -? or –help as the only argument and get a quick usage summary.

Use nano for editing script files in Bash. As soon as you get a command working, paste it into a file using nano before it gets lost. (But the Bash history command can often find older commands you issued a while back – just don't count on it always working).

Google as needed, and use your text. When you find a good reference or link, insert it into your script in the form of a comment:

```
# Like this comment
```

Don't forget that your script must start with the !# hack (“pound bang hack”) and must be made executable with the chmod command. So assuming your script is called: myscript.sh

Inside myscript.sh

```
#!/bin/bash

# lots of commands and comments
```

Then (only need to do this once):

```
chmod a+x myscript.sh
```

Then to run it, use

```
./myscript.sh
```

References

This link gives a lot of detail about what can be learned from observing solar eclipses:

https://www.nature.com/articles/s41550-017-0190