

Weeks 8 and 9: Determining total energy of your own flare

Welcome to Weeks 8 and 9! During these next two labs, you will be choosing your own flare to analyze from the Space Weather Data Portal and calculating the total energy of the flare. Your lab group has been assigned a flare class—you can find this assignment on [Canvas](#).

Remember, the [X-ray flare classification](#) represents the peak irradiance of the flare (as seen for wavelengths within the 1 to 8 Å range) and includes two parts. First, a letter (A, B, C, M, and X) representing an order of magnitude of the flare, and secondly, a number (1-9) within the specified order. For example, a X3 flare has a peak irradiance of $3 \times 10^{-4} \text{ W/m}^2$; whereas, a M3 flare has ten times lower irradiance. If the flare is larger than an X9 ($9 \times 10^{-4} \text{ W/m}^2$), then the number is extended beyond 9. For example, a X14 flare has a peak irradiance of $14 \times 10^{-4} \text{ W/m}^2$.

Today, you will start working on five tasks. You do not have to finish today, as you will finish working on these in week 9, but **you should at least be starting the baseline correction in task 4** by the end of lab today.

- (1) Go to the Space Weather Data Portal and choose a flare from the [assigned class](#).
- (2) Provide a description of the flare you chose and give the flare a unique name—be creative!
- (3) Download the data from the flare into a CSV file.
- (4) Use a similar procedure from last weeks lab and modify the code you wrote to determine the total energy of the flare of this flare.
- (5) Document your code using markdown cells and comments to explain your procedure. **Save your colab notebook file as a PDF to turn in by 11pm MT the day after your Week 9 lab.**
- (6*) Bonus: If you finish, find more flares to analyze!

Task 1: Find your solar flare

Go onto the [Space Weather Data Portal website](#) and follow the [tutorial posted on Canvas](#) to find a flare of the assigned class type that occurred during the GOES-15 satellite lifetime (April 2010 - November 2018).

Choose the flares to help the research project (i.e, not flares that have been previously analyzed in the class such as the Sept. 10th, 2017 flare or the flares from the baseline correction examples

colab notebook which occurred on May 5th, 2015, November 11th, 2013 and March 31st, 2011).

Task 2: Describe your flare

Since we have analyzed multiple flares, we decided to name them simply off of their date. This particular C class flare occurred on June 23 2012. The flare peaked at 07:50:00 hours exactly with a peak height of $2.75 * 10^{-6}$. The subclass of the flare is a C2 flare. The flare itself was very smooth and had no other flare interrupt it. The background was also smooth and we were able to take a lot of data points to analyze. Our flare stood alone and didn't have much noise to interrupt it other than some background noise towards the end of the flare. After removing the baseline however, we were able to lessen the effects of the background noise.

Task 3: Download the data

Download the longwave data from the Space Weather Data Portal website as a .csv file. Make sure to change the time be in ISO units.

▼ Task 4: Determine total energy of flare

Import the XRS longwave data of the solar flare you chose onto your personal computer. You can follow the same procedure that you used in the Week 5 lab.

Execute the code cell below to save the data into the variable "solar_data". Note, you may need to change the file name to match the file you imported into this session. Additionally, if you leave this session you may need to repeat the above steps to import the file again.

```
import pandas as pd

# You will need to change the filename
solar_data = pd.read_csv('noaa_goes15_xrs_1m-9.csv')    # Import data using pandas library

display(solar_data)    # Display the dataframe of the solar data
```

	time (yyyy-MM-dd'T'HH:mm:ss)	longwave (W/m ²)
0	2012-06-22T21:35:00	1.315230e-07
1	2012-06-22T21:36:00	1.335310e-07
2	2012-06-22T21:37:00	1.324460e-07
3	2012-06-22T21:38:00	1.285500e-07
4	2012-06-22T21:39:00	1.318120e-07
...	...	...
1173	2012-06-23T17:08:00	1.380540e-07

The solar flare data is in the units of solar irradiance ($\frac{W}{m^2}$). As you know by now, the next step is to convert to energy per second (*ergs/s*) and convert time from ISO units to seconds.

You've done this multiple times before at this point, so we'll just re-use the kind of code you've previously written to do this:

1178 rows × 2 columns

```
import numpy as np
import datetime as dt

# Convert df([time (yyyy-MM-dd'T'HH:mm:ss)]) into elapsed time in seconds
dates = [dt.datetime.strptime(date, '%Y-%m-%dT%H:%M:%S') for date in solar_data.iloc[:,0]]
solar_data.iloc[:,0] = [(date - dates[0]).total_seconds() for date in dates]
# Rename dataframe index
solar_data = solar_data.rename(columns={'time (yyyy-MM-dd\'T\'HH:mm:ss)': 'time (seconds)'})

w2erg = 10**7
int_m2 = 4*np.pi*(1.496*10**11)**2
# Convert units to ergs
solar_data['longwave (W/m^2)'] = solar_data['longwave (W/m^2)']*w2erg*int_m2
# Rename dataframe
solar_data = solar_data.rename(columns={'longwave (W/m^2)': 'longwave (ergs/s)'})

display(solar_data)
```

	time (seconds)	longwave (ergs/s)
0	0.0	3.698918e+23
1	60.0	3.755391e+23
2	120.0	3.724877e+23
3	180.0	3.615307e+23
4	240.0	3.707046e+23

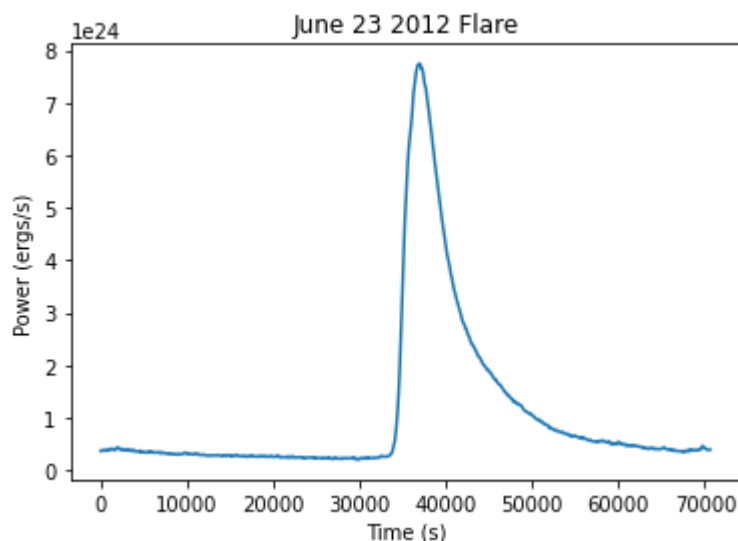
Next, let's plot the solar flare data as energy per second (*ergs/s*) versus time (*seconds*), so that we can clearly visualize the flare. Again, we've done this before, so we can reuse code:

```
1174          70440.0          3.869883e+23
```

```
import matplotlib.pyplot as plt
```

```
plt.plot(solar_data['time (seconds)'], solar_data['longwave (ergs/s)'])
plt.ylabel('Power (ergs/s)')
plt.xlabel('Time (s)')
plt.title('June 23 2012 Flare')
```

```
plt.show()
```



Baseline correction

As you learned in [Week 7 lecture](#), one of the primary challenges in determining the total energy of the flare is dealing with the "background." The sun is giving off x-rays even before a flare happens, and we do not want to accidentally include the energy of those x-rays as part of the flare energy. As a result, we need to determine the amount of x-ray power being emitted before the flare (the "baseline") and subtract it from the observed power during the flare to make sure we are only determining the *new* energy created by the flare itself.

Make sure your team has a clear plan for this; refer back to the Week 7 pre-lab lecture, the examples of baseline corrections on the Week 7 Canvas page, or ask your TA if you need help!

Hint: You will want to plot the data from the time period leading up to the flare in order to "see" it clearly and to decide on a strategy. For this part, we don't have to care yet about having time in seconds. In fact, it might be easier to leave time in arbitrary "index" units (i.e., data point 1, data point 2, data point 3, and so on). That way, it will be easy to discuss which data points you want to focus on. For example, maybe you think the pre-flare background is points 4 through 50. Then you can easily access these data points like this:

```
solar_data.iloc[4:50,1]
```

▼ How We're Doing Baseline Correction

After analyzing our preflare background, we noticed that there is a consistently decreasing level of noise. In order to take the best approximation here, since the decrease of the noise is relatively smooth, we are going to take the average of the local minimas in order to get an accurate representation of the baseline. This should account for the larger background noise we see at the beginning, and also account for the lower noise we see towards the end of the preflare data. We will need to import the argrelextrema library in order to complete the code of finding the minimas. Then we will take the average of those points to find an accurate baseline.

```
# You may need the statistics library for determining means and medians
import statistics as stats

# You will need to plot your flare
#(as shown in the examples of baseline corrections notebook)
# in order to look at the pre-flare data and show the baseline corrected flare
import matplotlib.pyplot as plt

plt.plot(solar_data.iloc[0:550,1]) #Graphing the Preflare background
plt.xlabel('Time (Seconds)')
plt.ylabel('Power (ergs/s)')
plt.title('June 23 Flare')
plt.show()

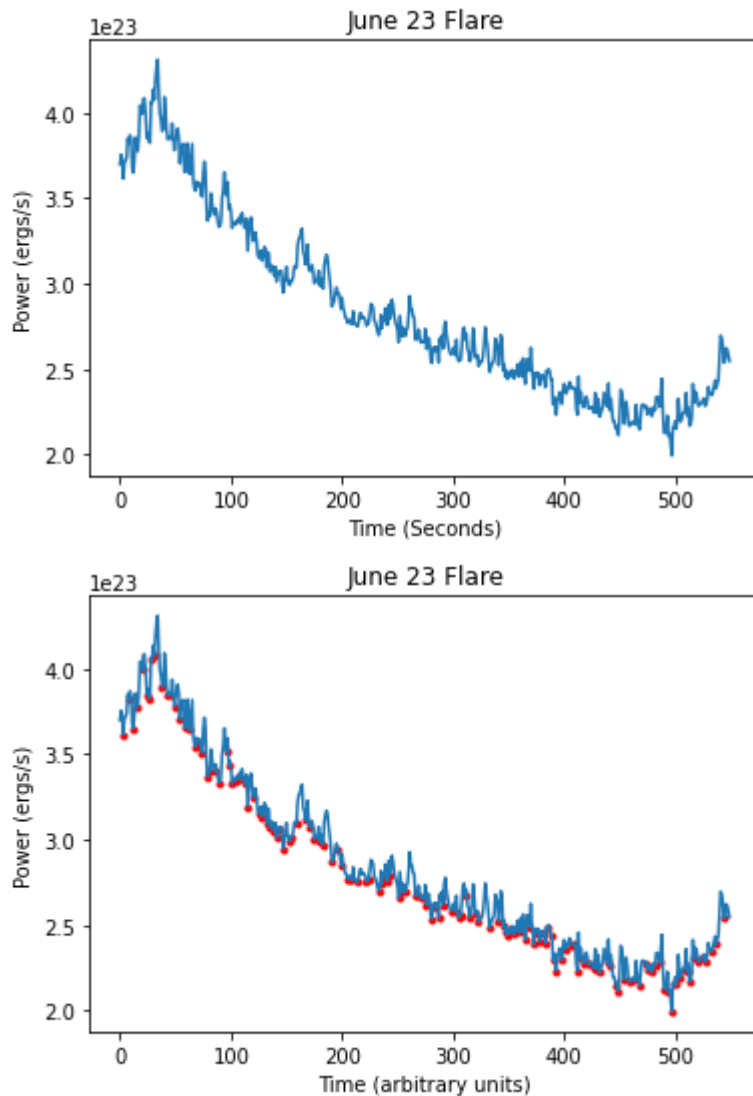
from scipy.signal import argrelextrema #import argrelextrema to find minimas

#converting it to an array
a = pd.DataFrame.to_numpy(solar_data.iloc[0:550,1])

#use argrelextrema to find the local minimas
min_locs = np.asarray(argrelextrema(a, np.less))
```

```
plt.plot(min_locs, a[min_locs], 'b') # Plot minima as red dots
```

```
plt.plot(min_locs, a[min_locs], 'r.') # Plot minima as red dots
plt.title("June 23 Flare")
plt.ylabel("Power (ergs/s)")
plt.xlabel("Time (arbitrary units)")
plt.plot(solar_data.iloc[0:550,1]) # Plot pre-flare data
plt.show()
```



#Now we will take the average of the minima points and use that as the baseline

```
plt.plot(solar_data.iloc[0:550,1]) # Zoom in on data before the flare begins
plt.title("June 23 pre-flare")
plt.ylabel("Power (ergs/s)")
plt.xlabel("Time (arbitrary units)")
```

Take the average

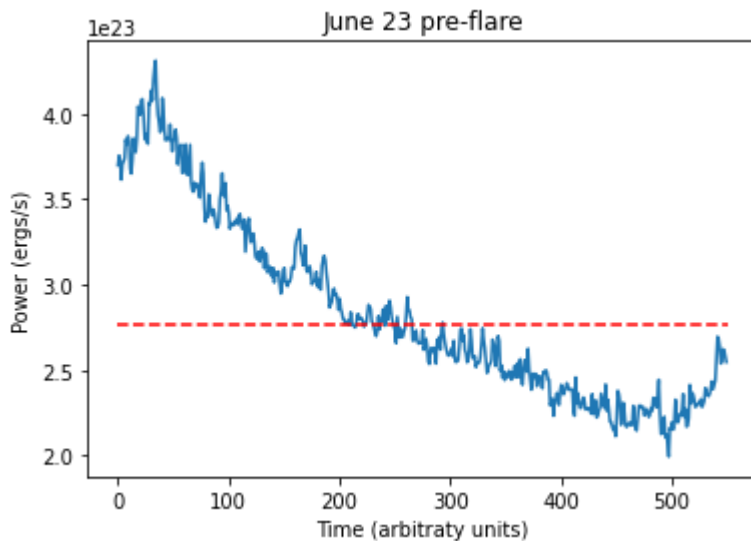
```
avg_baseline_M2 = a[min_locs].mean()
```

```
x_coordinates = [0, 550]
```

```
y_coordinates = [avg_baseline_M2, avg_baseline_M2]
```

```
plt.plot(x_coordinates, y_coordinates, 'r--')
```

```
plt.show()
```



```
#We will now plot the flare with the baseline in red
```

```
plt.plot(solar_data.iloc[0:1160,1])
plt.title("Jan. 26 Flare")
plt.ylabel("Power (ergs/s)")
plt.xlabel("Time (arbitrary units)")
```

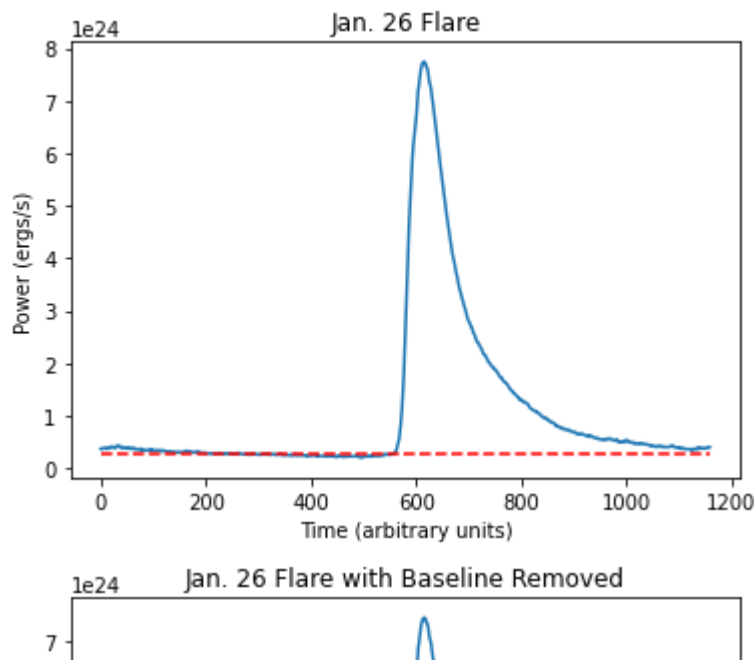
```
x_coordinates = [0, 1160]
y_coordinates = [avg_baseline_M2, avg_baseline_M2]
plt.plot(x_coordinates, y_coordinates, 'r--')
```

```
plt.show()
```

```
#The second plot is the flare with baseline removed
```

```
C_class_bltremov = solar_data['longwave (ergs/s)'].subtract(avg_baseline_M2)
```

```
plt.plot(C_class_bltremov[0:1160])
plt.title("Jan. 26 Flare with Baseline Removed")
plt.ylabel("Power (ergs/s)")
plt.xlabel("Time (arbitrary units)")
plt.show()
```



▼ Reporting total energy

err: 4 | | |

Once you have determined the baseline and subtracted this "background," you are ready to determine the total flare energy. Last week, we did this for a practice flare using a function we called `integrate.trapz`. Use this same function to compute the total energy of your new flare, referring back to your work last week if you need to.

Note: As several of our teams learned last week, this function is a little tricky! Make sure to use the appropriate limits for integration by indexing the `x` and `y` variables. You also have to tell it both the `y`-data AND the `x`-data to get the integration right (by default, it integrates assuming the `x`-coordinates are 1, 2, 3, 4, etc.) Be careful with that this week!

[*Use this markdown cell to describe how you are computing

We're computing the total energy by first plotting the first half of the graph to find a good starting point for our data. After this, we repeat the same process to find the end point for our data. Finally, we'll integrate our data using `scipy.integrate` folder to find the total energy.

```
#Import the integrate function; we will use trapezoidal integradion
#('integreat.trapz')
import scipy.integrate as integrate
```

```
## Write your code here
## Write your code here
plt.plot(C_class_b1remov[550:600], '-.')
plt.title("solar data start")
plt.xlabel("Time (arbitrary units)")
```

```

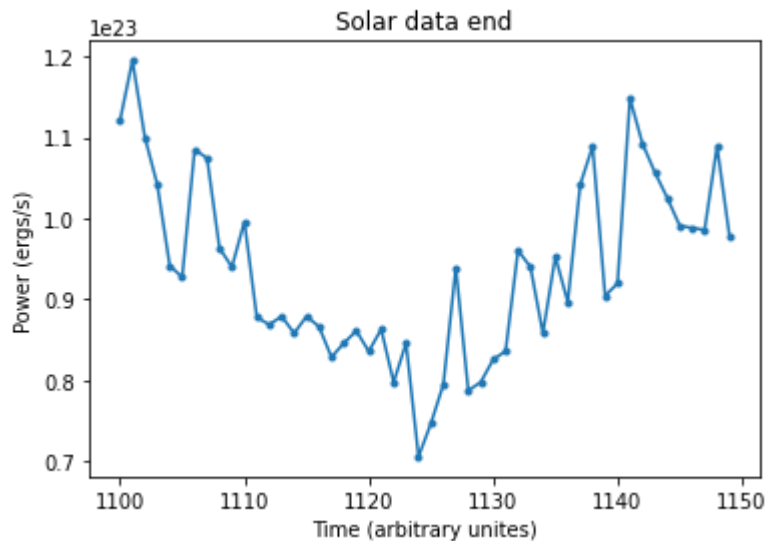
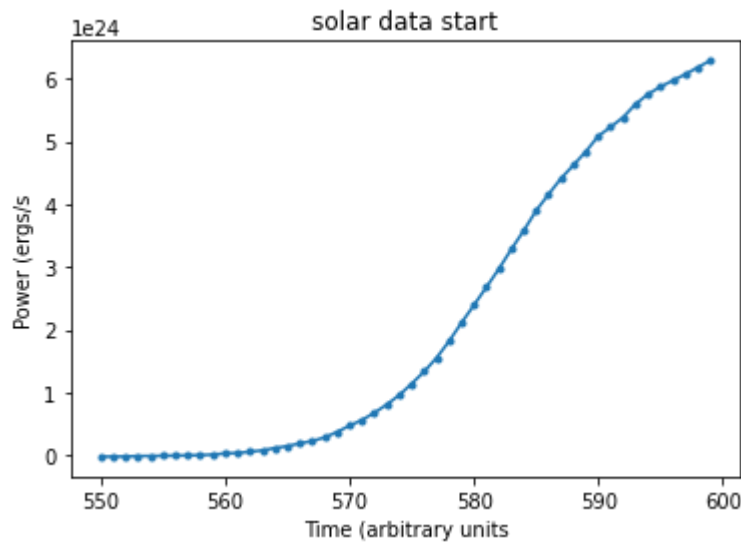
plt.ylabel("Power (ergs/s)")
plt.show()
#After analyzing the graph, we determined that the start point is at 550.

##This is to see the end points
plt.plot(C_class_bltremov[1100:1150],'.-')
plt.title("Solar data end")
plt.ylabel("Power (ergs/s)")
plt.xlabel("Time (arbitrary unites)")
plt.show()
#After analyzing the graph, we determined that the end point is at 1125.

##Now we integrate by using the integrate function f
A = integrate.trapz(C_class_bltremov[550:1125], solar_data.iloc[550:1125,0])

print('\n Total area under the curve: ', A, ' ergs')

```

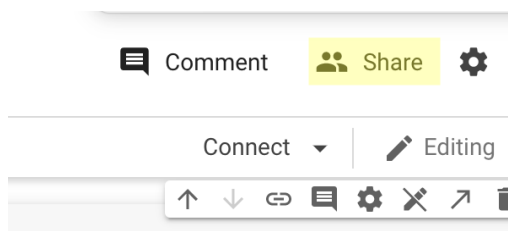


Total area under the curve: 5.446678495985554e+28 ergs

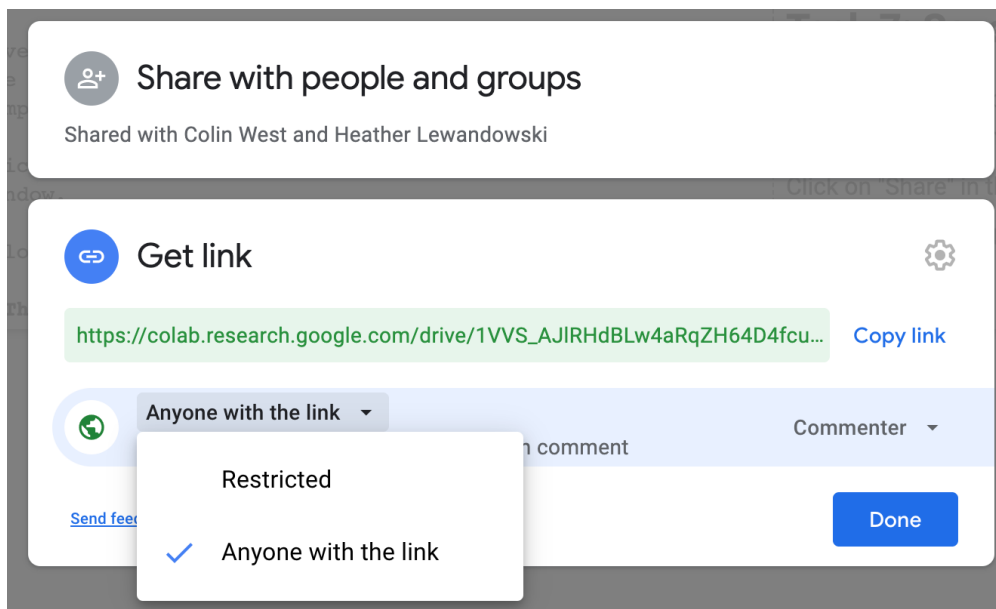
Note! As you may know, every measurement in experimental physics is subject to some **uncertainty**. The flare energy is no different-- we don't have time to figure out the uncertainty this week but we will definitely need to do that before reporting out final results

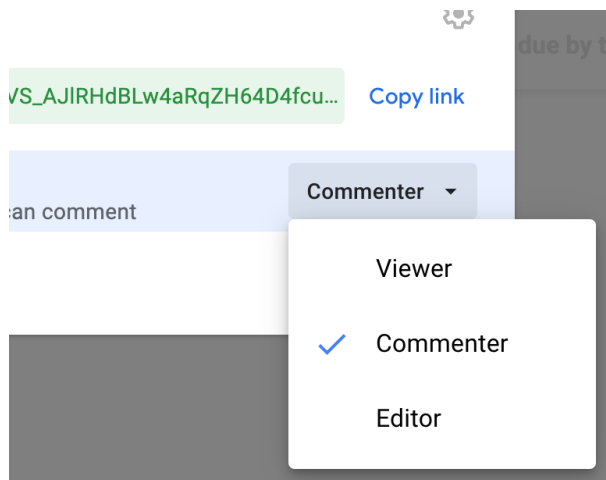
▼ Task 5: Comment, save, and submit

- Comment your code.
- Double-check all your code cells have been executed.
- Save this file as a PDF, click Print. In the printer dialog box select "Save as PDF" as your destination and click "Save".
- Double-check that you can see all of the executed code cells, tables, and graphs in the PDF. **IF YOU ARE HAVING TROUBLE PRINTING TO A PDF, PLEASE RUN THE CODE CELL AT THE END OF THE COLAB NOTEBOOK.**
- In this Google Colab notebook, click on "Share" in the upper right-hand corner of the window.



- Under "Get Link" click on "Change". Change the "Share" settings so that anyone with the link and be a commenter.





- In Canvas, upload the PDF file as an assignment and copy and paste the link to this notebook as the comment.

▼ Need help converting your colab notebook to a PDF?

The following code cell will covert your colab notebook to a PDF.

(1) You will need to first **change the filename** to the name the colab notebook you would like to convert. For example, your colab file for Week 8 and 9 is likely to be called 'Week8and9_DetermingTotalEnergyofFlare.ipynb' .

(2) You will need to mount your google drive. If your drive is not mounted already, the code cell will prompt you with the steps to do this as it runs.

(3) The PDF will be saved to the same folder as this code cell. You can then download the PDF from google drive and upload it onto Canvas.

(4) If you run into an error saying that the "list index is out of range" try two fixes:

- First, make sure your filename is correct
- Move your Colab notebook into the main area of the drive instead of having it in a folder.

```
#####
#                               CODE TO CONVERT COLAB NOTEBOOK TO PDF                               #
#This code converts an .ipynb file to a Tex based PDF and saves it in the Colab#
#Notebook folder with the same filename.pdf                                     #
#####

# Please change the file name to the name of your colab notebook
filename = 'Week8and9_DetermingTotalEnergyofFlare.ipynb' # Ex. 'Coding_Packet.ipynb'
```

```

from google.colab import drive
drive.mount('/content/gdrive', force_remount = True) # This is done first-
#- because we need a temporal delay between mounting and calling
#Install some dependences into the RUNTIME (is not local, needs to reinstall-
#- every runtime)
!echo 'Installing pips. This may take a few minutes.'
!pip install IPython >> outputsuppressed.txt
!pip install Latex >> outputsuppressed.txt
!pip install pandoc >> outputsuppressed.txt
!pip install nbconvert >> outputsuppressed.txt
!pip install jupyter >> outputsuppressed.txt

#Well known Ubuntu Tex library
!apt-get install texlive-xetex texlive-fonts-recommended texlive-generic-recommended >> outpu

# Searches the Google drive directory for the filename and gives back it's location
#(This accounts for Wildcards and Spaces in the directory names)
#Uses jupyter and nbconvert to convert the ipynb file to a Tex file, then into a
#pdf

!IFS=$'\n' #Sets the reader to only break at newlines instead of spaces,tabs,and newlines
!echo 'Finding file. This may take a minute or two depending on the size of your drive'
loc= !find '/content/gdrive' -name "{filename}"
fileloc = loc[0]
!jupyter nbconvert --to pdf "{fileloc}"

from datetime import datetime
import random
now = datetime.now()
print('\n', now.strftime("%d/%m/%Y %H:%M:%S"), random.randrange(1000000, 9999999, 1))

# The PDF will be in the same folder as the original file
!echo "";echo "Conversion Complete. The pdf will be in the same location as your .ipynb file"

##### Version:
# Created by Thomas Horning
# Last Update: 2020-09-25, 12:49 pm - Thomas Horning
# - Moved Dev Notes to Markdown Cell
# - Created Version Log
# - Added completion response

##

```

```
-----
KeyboardInterrupt                                Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in _input_request(self,
prompt, ident, parent, password)
    728             try:
--> 729                 ident, reply = self.session.recv(self.stdin_socket, 0)
    730             except Exception:
```

```
----- 6 frames -----
zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()
zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()
zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._recv_copy()
```

KeyboardInterrupt:

During handling of the above exception, another exception occurred:

```
KeyboardInterrupt                                Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in input_request(self,
```

Task 6*: Find more flares!

If time allows, choose another flare from C through X class! Follow the same procedure from above to report the total energy of the flare.

As before, make sure to provide a detailed description of the flares, which includes the date and time (in UCT) of the peak of the flare, as well as the subclass (e.g., C5.3) and other descriptive information.

