Draft version February 18, 2024
Typeset using LaTeX **manuscript** style in AASTeX631

A Simple Python Code for Generating Galactic Spectra

Ryan Butler[1]

[1]*Rochester Institute of Technology*

1. INTRODUCTION

The goal of this work is to create a simple python code that can easily generate synthetic galaxy spectra given a few physical parameters. This is not the first code to accomplish this task, indeed much more advanced undertakings have been ongoing for decades (**?**). That said, many of the preexisting galaxy spectra synthesizers rely on complex (but realistic) models of initial mass functions, stellar populations, and contributions from non-stellar sources like dust, or may focus on generating bulk simulation data (**?**. My aim with this code is to build something incredibly simple/accessible that allows the user to generate a galactic spectra relying on very few assumptions, namely just the distance and number of each spectral type involved. This model, while simple, should still convey some of the key aspects of galactic spectra.

2. CODE OVERVIEW

The code as written takes several inputs to generate a singular "galaxy" spectrum. These inputs are basic physical properties of the galaxy which is to be generated. Notably, this includes the number of stars of a given type in the synthetic galaxy (currently it supports the major sub-types O, B, A, F, G, K, M), as well as distance to the source.

2.1. *Allowed Stellar Types*

This code relies on pre-canned synthetic stellar atmosphere models from the "BOSZ" stellar atmosphere grid (**?**). These synthetic spectra can be generated for a wide range of stellar properties, such as temperature, surface gravity, and metallicity, and cover the range from 0.1μm to 32μm. They also

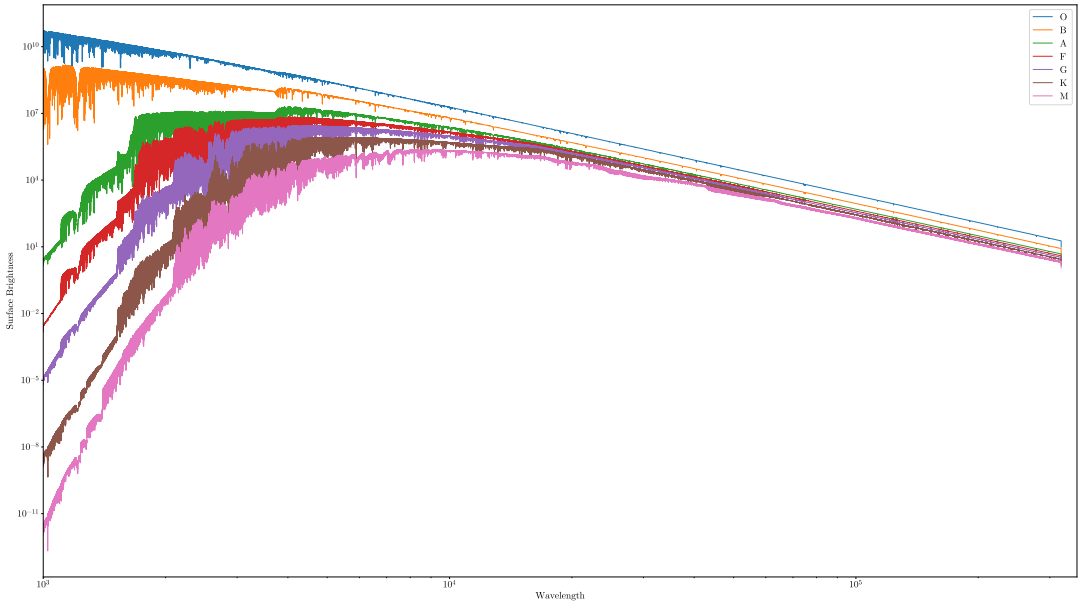

Figure 1. Loglog plot of wavelength and surface brightness with 20,000 instrumental broadening for all of the spectral types in this work.

may be generated for a range of instrumental broadenings to mirror conditions that may be achieved from observations. For this work, I've chosen to keep things to a simplified system of the 7 major stellar sub-types at two different instrumental broadenings (20,000 and 300,000), which I believe are supposed to proxy for resolving power. Metallicity is kept at solar abundances in all cases.

In general, I aim to select a temperature that best corresponds to the X5V spectral type, where X represents the overarching class (O, B, A, etc.), as noted in **?**. In three instances this is not possible due to the limitations of the grid. These are for the O, G, and M type stars, where the O5V has a surface temperature in excess of the BOSZ grid allowances (3500-35,000 K), and similarly the M5V has a temperature below the minimum possible value. The grid spacing is too coarse to hit the appropriate temperature for G5V. In lieu of these I implement the closest possible sub-class within the same overarching spectral type. The minimum temperature constraint also precludes implementing any spectral contribution from the sub-stellar population (sometimes called L and T types), although this is a presumably negligible quantity anyways. A full exposition of the BOSZ parameters for each star type is given in table 1, with the initial plots of surface brightness shown in figure 1.

SpT	Teff (K)	Log(g)	[M/H]	[C/M]	[α/M]	Inst. Broadening (km/s)
O8V	35,000	5.0	0.00	0.00	0.00	20,000
						300,000
B5V	15,500	5.0	0.00	0.00	0.00	20,000
						300,000
A5V	8,000	5.0	0.00	0.00	0.00	20,000
						300,000
F5V	6,500	5.0	0.00	0.00	0.00	20,000
						300,000
G7V	5,500	5.0	0.00	0.00	0.00	20,000
						300,000
K5V	4,500	5.0	0.00	0.00	0.00	20,000
						300,000
M2V	3,500	5.0	0.00	0.00	0.00	20,000
						300,000

Table 1. Complete list of parameters for the synthetic stellar atmospheres used in this work from the BOSZ library.

2.2. *File Labelling*

Atmospheric models from the BOSZ grid are available in FITS and ASCII filetypes. For the main part of this code, I opt to work with CSV instead. This means that a conversion needs to be performed on any BOSZ files for the code to work properly. In the event any further BOSZ models are implemented in the future, they will need to go through this process as well. All of this is handled by a secondary snippet of code I wrote to handle the conversion and add appropriate headers to the 3 output columns.

2.3. *User-Input Parameters*

The code allows the user to enter several physical parameters of the system for which a spectra is to be generated. The first of these are `spectral_types` and `resolution`, in which `spectral_types` lets one select the spectral types to be considered (out of the allowed O, B, A, F, G, K, M) as a list,

and `resolution` lets the user select the instrumental broadening to be used. Note here the latter depends on what has been pulled from the BOSZ grid, and in this simple version is limited to either 20 or 300 (short for 20,000 and 300,000 respectively). Next appearing is `N_tot` which will set the (approximate) total number of stars in the generated galaxy. After the total number of stars has been set, there are 7 different fractions to set, one for each type of star (called `F_O`, `F_B`, etc.). Ideally these values will add up to 1, but for better or worse there is no penalty for failing to do so, the only ramification is that the actual number of stars used to generate the spectra will deviate from the set total.

If the user changes the input stars at all (either by adding more spectral sub-types or simply changing the existing ones) they will need to change a few additional parameters. Notably, the absolute magnitudes (`M_O`, etc.) as well as the masses (`Mass_O`, etc.). If these are not edited then the results will deviate from reality more-so than they already do due to the plethora of simplifications involved.

The final variables of interest to the user are `redshift` and `H_0`. The `redshift` parameter can be set to any positive value. If something other than that occurs it will trigger a warning and set to a default value near 0. `H_0` is the Hubble constant and should be kept around 70, but may be edited. This will impact the distance calculations to the generated galaxy.

2.4. *Function*

The code works, generally, by calculating the bulk flux from the user selected stars at the appropriate redshift, and then producing a combined spectra from all allowed spectral types. In the code, this process occurs in a few steps. Initially, the files for each spectral type are brought in under a dictionary, which allows the user to set the broadening of interest. Once again, in this preliminary application I have limited this to the 7 big spectral types, but one could conceivably expand this over everything available in the model grid. Immediately after setting up the dictionary, the user selects the allowed spectral types and broadening.

The next portion of the code accepts the input total number of stars and fraction of each spectral type to determine the number of stars of each spectral type (this is done simply by multiplying each

fraction by the total). It will warn the user if the total fraction is above or below 1, but will not stop the code from proceeding. The user then sets the redshift, which will calculate the distance to the galaxy using the simple linear relation with the Hubble constant:

$$d = \frac{c \times z}{H_0} \tag{1}$$

Where d is the distance in Mega-parsecs, c is the speed of light in km/s, z is the user set redshift ($z > 0$), and H_0 is the user specified Hubble constant, in units of km/s/Mpc. This relationship is not particularly realistic at large redshifts due to cosmological effects, nor is it particularly well served at the nearest redshifts where galaxies are not subject to the Hubble flow. That said, it is a simple approximation that works well enough in this simple code.

The code then converts the redshift derived distance to parsecs and uses this information, along with a given absolute magnitude, to calculate the relative magnitude of a star of a given type. The formula used is the well known:

$$m = 5\log_{10}(D) - 5 + M \tag{2}$$

In which D is the distance in parsecs, m is the calculated relative magnitude, and M is the absolute magnitude for the spectral type of interest. This is performed for all 7 spectral types. Down the line, this will be used to calibrate the flux for each star with respect to the observed distance. Another equation is invoked to accomplish this:

$$m_1 - m_2 = -2.5\log_{10}\left(\frac{f_1}{f_2}\right) \tag{3}$$

In particular, this is rearranged such that:

$$f_2 = \frac{f_1}{10^{\left(\frac{m_1 - m_2}{-2.5}\right)}} \tag{4}$$

Where f_2 is the distance-calibrated flux (essentially the flux observed after all is said and done). The code handles this in a two-step procedure, opting initially only to calculate the $\frac{m_1 - m_2}{-2.5}$ term in a series of values denoted `expO`, `expB`, etc. This is purely a move in the interest of readability and making sure there are no missteps in the mathematics. m_1 is set to 0 for the calibrations.

N_tot	z	F_O	F_B	F_A	F_F	F_G	F_K	F_M
3×10^{11}	0.0000001	0.0000003	0.0013	0.006	0.006	0.076	0.121	0.765

Table 2. Parameters input to generate the synthetic galactic spectra shown in Figure 2.

The rest of the code handles the generation of the amalgamated spectrum. There is a for loop in which the CSVs for each type of star are parsed. The wavelengths for redshifted per the standard formula:

$$\lambda_{obs} = (1 + z)\lambda_{emit} \tag{5}$$

In which the emitted wavelengths are the ones from the BOSZ model. Additionally, the BOSZ output only gives a surface brightness, which must be converted to flux using the formula:

$$Flux = \pi \times SB \tag{6}$$

Where SB is the given surface brightness. There is a factor of π instead of the usual 4π because the models already contain the factor of 4. In the code this calculated flux is named the *Observed Flux*, a bit of a misnomer because it is not yet distance corrected. The so-called observed flux is calibrated per equation 4, yielding the calibrated flux. That flux is then multiplied by the number of stars of the given type, providing a total flux for each type of star. The rest of the code simply combines these and plots them.

3. RESULTS

The code successfully generates synthetic galaxy spectra. Figure 2 shows a synthetic galaxy at low redshift using milky-way based parameters (table 3), and has a slew of expected spectral features visible in the first 10,000 $\mathring{A}$. These include the well known balmer-lines, as well as the UV contributions expected from the high-mass blue stars such as the O and B types. Figure 3 likewise shows a crude attempt at an elliptical galaxy at the same redshift, with very few blue stars. The peak appropriately appears in a much redder part of the spectrum, as expected.

4. DISCUSSION

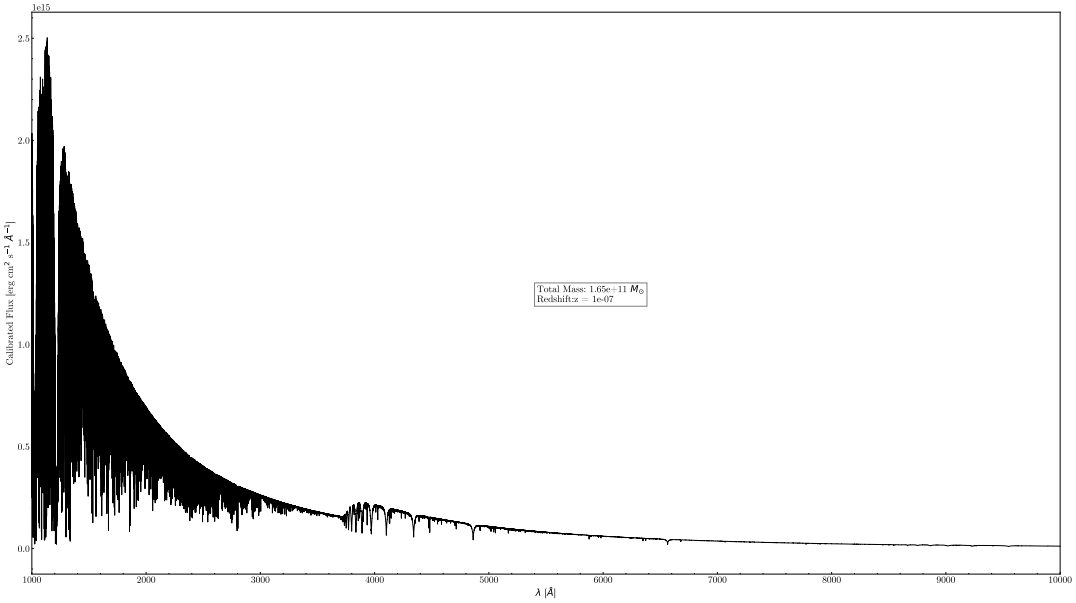

Figure 2. 0-10,000 $\mathring{A}$ spectrum for the galaxy generated by the inputs given in table 3

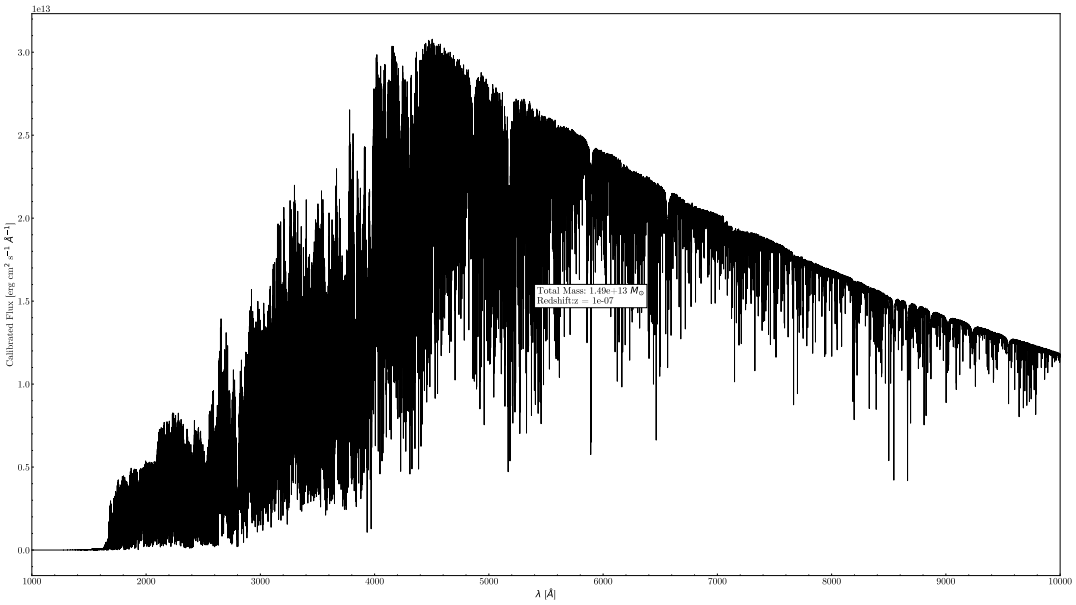

Figure 3. 0-10,000 $\mathring{A}$ spectrum for the "elliptical" galaxy generated by the inputs given in table 3

N_tot	z	F_O	F_B	F_A	F_F	F_G	F_K	F_M
3×10^{13}	0.0000001	0.0	0.0	0.00005	0.0005	0.005	0.21	0.78

Table 3. Parameters input to generate the "elliptical" synthetic galactic spectra shown in Figure 3.

This work serves as a simple proof of concept for a code that can quickly generate non-complicated galactic spectra. It includes a number of gross simplifications, but these aid in building an accessible, easy to use toy model for this sort of thing. However, these also lead to some clear drawbacks. The code makes no quarter for any complicating factors, be that sky-lines, any sort of interstellar or intergalactic medium, active galactic nuclei, etc., all of which limit its practical applications.

In its present form it also relies entirely on the BOSZ synthetic stellar spectra library. I debated this decision a bit since there are other equally valid libraries of both synthetic and standard stellar spectra that may be used. The BOSZ is very user friendly, so if a user wished to implement more detail into the code I think this approach makes that fairly straightforward. That said, adding further capability in this realm may increase the overall robustness of the tool, as comparing multiple model outcomes is important in this work.

Despite the simplified, straightforward approach, this code sets out to do what I wanted it to quite well. Given a few simple physical parameters, one can output a detailed model spectra. It is perhaps not the most useful/practical tool out there but it provides an interesting basis to build something that is more applicable.

5. FUTURE WORK

There are an almost alarming number of ways to improve upon this code/increase it's functionality in a meaningful way. The "obvious" next step would be to build off of this and create a sort of fitting tool, i.e. by providing it with an input spectrum the code would try and output some of the basic properties of the galaxy, such as total mass and composition. I also think that task seems deceptively straightforward and would take significantly longer to put-together/implement than this toy code does. The silver lining there is than most surveys that exist (i.e. SDSS (**?**)) provide some calibrated redshift measurements already, so that parameter could be taken care of as is. There exist code packages in the wild that already do this for certain types of galaxies/spectra, such as `PyQSOFit` which is designed specifically to fit quasar spectra (**?**).

The path I am more interested in pursuing is turning this into a web-tool. Something like a simple form where the user could enter the basic parameters (total stars, fraction of each type, and redshift)

and then be returned the spectra to look at. Essentially the same thing the code does now but with a simple user interface.

An additional area of improvement considered, but not gotten to, is implementing some well known initial mass functions/probability mass functions in lieu of having the fraction of each star type be user implemented. This would reduce what is presently one of the less straightforward parts of the code into something with a handful of options that should keep things grounded in observation better and also simplify the experience. It might be particularly interesting to implement these in addition to an "age" calculation to show how the spectra evolves over time, since in it's current state the code is very much a static look at galactic spectra, without consideration for stellar lifespans.

6. ACKNOWLEDGEMENTS

This work makes use of models drom the ATLAS9 Model Atmosphere Database ("BOSZ"), accessed through the Mikulski Archive for Space Telescopes.

REFERENCES

Bayo, A., Rodrigo, C., Barrado Y Navascués, D., et al. 2008, A&A, 492, 277, doi: 10.1051/0004-6361:200810395

Gagné, J., Mamajek, E. E., Malo, L., et al. 2018, ApJ, 856, 23, doi: 10.3847/1538-4357/aaae09

Gaia Collaboration, Prusti, T., de Bruijne, J. H. J., et al. 2016, A&A, 595, A1, doi: 10.1051/0004-6361/201629272

Gaia Collaboration, Vallenari, A., Brown, A. G. A., et al. 2023, A&A, 674, A1, doi: 10.1051/0004-6361/202243940

Jarrett, T. H., Comrie, A., Marchetti, L., et al. 2021, Astronomy and Computing, 37, 100502, doi: 10.1016/j.ascom.2021.100502

Kastner, J. H. 2016, in Young Stars & Planets Near the Sun, ed. J. H. Kastner, B. Stelzer, & S. A. Metchev, Vol. 314, 16–20, doi: 10.1017/S1743921315006547

Lee, J., & Song, I. 2017, Monthly Notices of the Royal Astronomical Society, 475, 2955, doi: 10.1093/mnras/stx3195

—. 2019, Monthly Notices of the Royal Astronomical Society, 489, 2189, doi: 10.1093/mnras/stz2290

Lee, J., Song, I., & Murphy, S. 2020, MNRAS, 494, 62, doi: 10.1093/mnras/staa689

Zuckerman, B., & Song, I. 2004, ARA&A, 42, 685, doi: 10.1146/annurev.astro.42.053102.134111

APPENDIX

A. CODE

```
    # -*- coding: utf-8 -*-
"""
@author: Ryan Butler

For Computational Astrophysics final project Fall 2023

This is a code that takes a few basic inputs and then outputs an aggregate
galactic spectra from 0-35 microns.

Email: rwb5439@rit.edu
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings



plt.rcParams["font.family"] = "CMU Serif"
plt.rcParams.update({'font.size': 16})

# This is a dictionary for the different instrumental broadening values
# for each SpT. USER MUST UPDATE THE FILEPATHS TO THEIR OWN
```

```
file_paths = {
    'O': {'300': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\O8V_300000.csv',
          '20': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\O8V_20000.csv'},
    'B': {'300': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\B5V_300000.csv',
          '20': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\B5V_20000.csv'},
    'A': {'300': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\A5V_300000.csv',
          '20': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\A5V_20000.csv'},
    'F': {'300': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\F5V_300000.csv',
          '20': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\F5V_20000.csv'},
    'G': {'300': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\G7V_300000.csv',
          '20': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\G7V_20000.csv'},
    'K': {'300': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\K5V_300000.csv',
          '20': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\K5V_20000.csv'},
    'M': {'300': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\M2V_300000.csv',
          '20': r'C:\Users\rbutl\Desktop\Comp_Project_F23\Output\M2V_20000.csv'}
}

# Specify the spectral types and resolution
spectral_types = ['O', 'B', 'A', 'F', 'G', 'K', 'M']
resolution = '20'  # You can change this to '300' if needed, it takes a lot longer
                   # and doesn't change much at all.

# Specify the TOTAL NUMBER of each type of stars in this galaxy
N_Tot = 3e11

# Specify the FRACTION each star comprises. SHOULD ADD TO ~1.0
# The given values come from wikipedia...
```

```
F_O = 0.0000003
F_B = 0.0013
F_A = 0.006
F_F = 0.03
F_G = 0.076
F_K = 0.121
F_M = 0.765



# Elliptical galaxy params
#F_O = 0.0
#F_B = 0.0
#F_A = 0.00005
#F_F = 0.0005
#F_G = 0.005
#F_K = 0.21
#F_M = 0.78

SUM = F_B + F_A + F_F + F_G + F_K + F_M + F_O
print(SUM)
# Check if the sum is over or under 1
if SUM > 1.0:
    print("Warning: The sum of all star type fractions is over 1. Check your values.")
elif SUM < 1.0:
    print("Warning: The sum of all star type fractions is under 1. Check your values.")
else:
    print("Star fractions add to 1. Good.")
```

```
# Calculate total number of each type of star
N_O = N_Tot * F_O
N_B = N_Tot * F_B
N_A = N_Tot * F_A
N_F = N_Tot * F_F
N_G = N_Tot * F_G
N_K = N_Tot * F_K
N_M = N_Tot * F_M

Sum_Num = N_B + N_A + N_F + N_G + N_K + N_M + N_O
#print(Sum_Num)

# REDSHIFT - USER MUST SET THIS
redshift = 0.0000001 # This is redshift (z). Please do not set to 0.

# Check if redshift is 0, if so, set to arbitrarily close to 0
if redshift <= 0:
    redshift = 0.00000000000001  # This prevents disaster
    print("Redshift SHOULD NOT be 0 or negative. Setting to default value (z = 0.0000000000
z = redshift
c = 299792 #km s^-1
H_0 = 70 # km s^-1 Mpc


#Need to convert redshift to distance (d = cz/Ho)
# This is a gross oversimplification but works okay here
```

```
d = ( c * z )/ H_0 #Mpc
ly = 3.262e6 * d # just prints the above distance in ly. Sanity check.

D = d * 1000000 #pc

# Masses ( all in M_sun)
MassO = 23.6
MassB = 4.7
MassA = 1.88
MassF = 1.33
MassG = 0.95
MassK = 0.70
MassM = 0.44

# Calculate the total mass of each stellar type in the galaxy
Tot_MassO = MassO * N_O
Tot_MassB = MassB * N_B
Tot_MassA = MassA * N_A
Tot_MassF = MassF * N_F
Tot_MassG = MassG * N_G
Tot_MassK = MassK * N_K
Tot_MassM = MassM * N_M

Mass_tot = Tot_MassO + Tot_MassB + Tot_MassA + Tot_MassF + Tot_MassG + Tot_MassK + Tot_MassM
print(r"The total mass of the galaxy in M_sun is:")
print(Mass_tot)
# Absolute Magnitudes
```

```
M_O = -4.50 # O8V

M_B = -0.85

M_A = 2.01

M_F = 3.37

M_G = 5.20 # G7V

M_K = 7.28

M_M = 10.21 # M2V


#Calculate apparent magnitudes using the distances.

m_O = (5 * np.log10(D)) - 5 + M_O

m_B = (5 * np.log10(D)) - 5 + M_B

m_A = (5 * np.log10(D)) - 5 + M_A

m_F = (5 * np.log10(D)) - 5 + M_F

m_G = (5 * np.log10(D)) - 5 + M_G

m_K = (5 * np.log10(D)) - 5 + M_K

m_M = (5 * np.log10(D)) - 5 + M_M


#m1 for flux calibration. Do not edit this.

m1 = 0.0


# exponents to get calibrated fluxes

expO = (m1 - m_O) / -2.5

expB = (m1 - m_B) / -2.5

expA = (m1 - m_A) / -2.5

expF = (m1 - m_F) / -2.5

expG = (m1 - m_G) / -2.5

expK = (m1 - m_K) / -2.5
```

```
expM = (m1 - m_M) / -2.5

# This loop is soley so that the total flux array inializes in a non-broken way
# It is inefficient but harmless.
for spectral_type in spectral_types:

    # Use the dictionary to get the file path
    file_path = file_paths[spectral_type][resolution]

    # Read the CSV file
    df = pd.read_csv(file_path)

    # Calculate 'Observed Wavelength' w.r.t. z
    df['Observed Wavelength'] = (1 + redshift) * df['Wavelength']

    # Calculate 'Observed Flux' using formula from BOSZ webpage
    df['Observed Flux'] = np.pi * df['Surface Brightness']  # erg cm^-2 s^-1 A^-1

# Initialize an array to store the total flux
total_flux = np.zeros(len(df['Observed Wavelength']))

# IMPORTANT LOOP THAT CREATES SPECTRA
for spectral_type in spectral_types:

    # Use the dictionary to get the file path
    file_path = file_paths[spectral_type][resolution]
```

```
# Read the CSV file
df = pd.read_csv(file_path)

# Calculate 'Observed Wavelength' w.r.t. z
df['Observed Wavelength'] = (1 + redshift) * df['Wavelength']

# Calculate 'Observed Flux' using formula from BOSZ webpage
df['Observed Flux'] = np.pi * df['Surface Brightness']  # erg cm^-2 s^-1 A^-1

# Bring in the calibration exponents
exp_SpT = globals()[f'exp{spectral_type}']

#Calibrate fluxes
Flux_Calib = df['Observed Flux'] / (10**exp_SpT)

#Bring in total number of each SpT
N_SpT = globals()[f'N_{spectral_type}']

F_tot = N_SpT * Flux_Calib

# Overplot fluxes (Sanity check)
plt.loglog(df['Wavelength'], df['Surface Brightness'], label=f'{spectral_type}')

plt.xlim(0.1 * 10**4, 35 * 10**4)
plt.xlabel('Wavelength')
plt.ylabel('Surface Brightness')
plt.legend()
```

```
    plt.show()



    total_flux += N_SpT * Flux_Calib



plt.figure() #new fig
# Plot the total calibrated flux
plt.plot(df['Observed Wavelength'], total_flux, label='Total Flux', color = 'black')


plt.minorticks_on()


plt.xlim(0.1 * 10**4, 1 * 10**4)


plt.xticks(minor=True)
plt.yticks(minor=True)
plt.tick_params(axis='x', which='both', direction='in')
plt.tick_params(axis='y', which='both', direction='in')



plt.xlabel(r'$\lambda$ [$\AA$]')
plt.ylabel(r'Calibrated Flux [erg cm$^{2}$ s$^{-1}$ $\AA^{-1}$]')
total_mass_sci = "{:.2e}".format(Mass_tot)
textstr = f'Total Mass: {total_mass_sci} $M_{{\odot}}$\nRedshift:z = {redshift}'
plt.gcf().text(0.5, 0.5, textstr, fontsize=16, bbox=dict(facecolor='white', alpha=1))
#plt.legend()
plt.show()
```