Milestone 3: AGN Galaxy Simulation and Spectra Modeling

Annabella Yang

The code for the project is attached to this pdf (week 3 starts from page 58 and ends at page 82) and updated to GitHub https://github.com/AnnabellaYang/PHYS129_Final_Project_3.

Last week, I created the class “Galaxy” with the black hole accretion model, cooling rate, heating rate, temperature evolution model, and spectra modeling function in this class. Since spectra modeling was so significant and reveals many information, I chose to continue my exploration on this topic. I continued using the method of optimization, increased the boundary condition and the size of iteration to increase the range and accuracy of successfully measuring the redshift. To show the implication of this computation method of AGN galaxy simulation, I also used a code called ALF developed by Conroy which is a complex model using similar idea but developed over decades. I ran a monte carlo simulation to fit the model to an actual observed spectra, which was observed using Megellan telescope in Chile couple years ago by Meng Gu from Carnegie Observatory, using Cal Tech HPC. Then, I modified a code I previously wrote to plot out the spectra and presented a corner plot that shows the correlation between metallicity, chemical elemental abundance, and age.

This week’s work focused on the spectra modeling part of my galaxy class. In addition to what I had before, I also generated error from instrumentation and atmosphere that are usually important factors in observation, and plotted the signal to noise ratio for each spectrum. In general, the higher the signal to noise ratio, the better the quality of the data, and the better result we will be able to get from simulation. The signal to noise ratio is also higher at the emission lines because they are usually strong emissions that can easily be detected. This along with the increased range and iteration mentioned above completed my galaxy class. I then did an example using the galaxy class to generate a galaxy. I then calculated the star formation rate (SFR) and H_alpha/H_beta ratio using the modeled spectrum. I plot the correlation with the data distribution contours in the same plot and observed a direct correlation between the SFR and the H_alpha/H_beta ratio. Also, the points seem to scatter at the low SFR and the low H_alpha/H_beta ratio region even if the redshift was uniformly distributed. Finally, as mentioned above, I used the ALF code to present some implications of such computation method and presented some interesting informations it could reveal using a correlation corner plot.

I faced some difficulties when creating the class, because there are so many variables and I failed to manage all of them last week. I went over the class from top to bottom one more time and compared it with the example we discussed in class. I started to use initialization condition wisely and that reduces a lot of redundant work. I also faced trouble when failing fitting the spectrum and finding the correct redshift at high redshift galaxy and did not find out the solution last time. I re-analyzed the model by plotting out the error at different redshift, then I found it interesting that this relation breaks exactly at redshift = 1, which is abnormal. Therefore, I checked the initial condition and the iteration for different attempted redshifts and found out it was a problem with my original setting. After fixing it, although it takes longer to find the best fitted model, it correctly predicts (with a very small error less than 0.01) the redshift up to redshift = 10 (because I set the upper bound to 10), which exceeds most of the observable objects.

After understanding the idea of this computation, I think I will be able to work with spectra modeling for observed data in the future. My next step is to look into other people’s simulation code and see what they did to develop such a huge physics model. Some important and popular spectra modeling code includes BEAGLE, GALFIT, and ALF. I hope I will be able to put more practice into spectra modeling even beyond this class.

Reference:

Choi, J., Dotter, A., Conroy, C., Cantiello, M., Paxton, B., & Johnson, B. D. (2016). Mesa isochrones and Stellar Tracks (mist). I. Solar-scaled models. *The Astrophysical Journal*, *823*(2), 102. https://doi.org/10.3847/0004-637x/823/2/102

Conroy, C., Graves, G. J., & van Dokkum, P. G. (2013). Early-type Galaxy archeology: Ages, abundance ratios, and effective temperatures from full-spectrum fitting. *The Astrophysical Journal*, *780*(1), 33. https://doi.org/10.1088/0004-637x/780/1/33

Conroy, C., Villaume, A., Dokkum, P. G. van, Lind, K., https://orcid.org/0000-0002-1590-8551, C. C., https://orcid.org/0000-0003-1887-0621, A. V., & https://orcid.org/0000-0002-8282-9888, P. G. van D. (2018, February 21). *IOPscience*. The Astrophysical Journal. https://iopscience.iop.org/article/10.3847/1538-4357/aaab49

Sanchez-Blazquez, P., Peletier, R. F., Jimenez-Vicente, J., Cardiel, N., Cenarro, A. J., Falcon-Barroso, J., Gorgas, J., Selam, S., & Vazdekis, A. (2006). Medium-resolution Isaac Newton Telescope Library of empirical spectra. *Monthly Notices of the Royal Astronomical Society*, *371*(2), 703–718. https://doi.org/10.1111/j.1365-2966.2006.10699.x

Physics 129L Final Project: Investigating the Outflow Dynamics of Active Galactic Nuclei through Computational Simulations

AGN Galaxy Simulation

My goal of this project is to study the outflow dynamics of Active Galactic Nucleis (AGNs) since it is crucial for comprehending their roles in galaxy formation and evolution and aims to simulate the outflow properties of AGNs using computational techniques to understand their process and implications for galaxy evolution.

However, before starting analyzing the properties of AGN, we need some sample of AGN host galaxies to start with, and this is one of the most complex process since there are too many things to be considered. We start by considering some most important factors of AGN host galaxies in the simulation:

1. **Modeling the Black Holes**: Simulating the behavior of supermassive black holes at the centers of these galaxies, including their mass, spin, and accretion rates.
2. **Modeling the Gas Dynamics**: Modeling the accretion of gas onto the black hole and the resulting jets and winds.
3. **Consider the Galactic Interactions**: Considering the gravitational interactions between galaxies, especially in dense environments like galaxy clusters.
4. **Consider the Feedback Processes**: Including the effects of the AGN on its host galaxy, like outflows that can heat up or expel gas, affecting star formation.
5. **Consider the Fact that it is a Part of Our Universe (Cosmological Context)**: Placing these galaxies within the larger structure of the universe, considering the expansion of the universe, dark matter distribution, and the cosmic web.
6. **Include the Radiation Processes**: Simulating the radiation emitted by the AGN, which includes a wide spectrum from radio to gamma rays.

I think the best way to simulate such galaxies is to create a galaxy class and define these above properties seperately. So the first step is to initiate a class called galaxy with basic parameters.

In [93]:
```
# First, import useful packages (subject to change, will be adding all the p
import numpy as np
import scipy as sp
import random
import matplotlib.pyplot as plt
import corner
import seaborn as sns
from scipy.stats import norm
from scipy.optimize import minimize
from scipy.optimize import curve_fit
from scipy.stats import gaussian_kde
from astropy import units as u
```

In [2]:
```
# Second, define some constant
# (unit included by using the astropy package units)
'''Gravitational constant'''
G = 6.67430e-11 * u.m**3 / u.kg / u.s**2
'''speed of light'''
c = 299792458 * u.m / u.s
'''number of galaxies to be simulated'''
num_galaxies = 1000
'''time steps'''
time_steps = 100
```

In [3]:
```
# Now we can start by defining a class of galaxy, and think about the struct
class Galaxy:
    # initialize the galaxy
    def __init__(self, mass, position, velocity, black_hole_mass):
        self.mass = mass
        self.position = position
        self.velocity = velocity
        self.black_hole_mass = black_hole_mass

    # black hole accretion model
    def accretion_model(black_hole_mass):
        '''
        This will be an impliment or accresion model for black hole accretic
        This should return accretion_rate, radiation, jets
        '''
        pass

    # Gas Dynamics and Interstellar Medium
    def gas_dynamics(galaxy):
        '''
        This models the impliment model of the dynamics of gas in and around
        The processes like cooling, heating, and interaction with AGN radiat
        '''
        pass

    # Galaxy Interaction and Dynamics
    def update_galaxy_dynamics(galaxies):
        '''
        Implement gravitational interactions between galaxies.
        Update positions and velocities based on gravitational forces.
        '''
        pass
```

A) Accretion Model

The accretion model also contains many parts:

1. accretion rate
2. radiation
3. jet production

I will do these step by step by using and analyzing the equations that describe each model

1. Accretion Rate

The accretion rate of a black hole often depends on the **mass of the black hole**, the **density of the surrounding medium**, and the **velocity at which this medium is moving relative to the black hole**.

I here use the Bondi-Hoyle-Lyttleton accretion rate model: (https://en.wikipedia.org/wiki/Bondi_accretion)

$$\dot{M} = \pi \left(\frac{2G^2 M^2 \rho_\infty}{(v_\infty^2 + c_s^2)^{3/2}} \right)$$

$$\begin{align}
\dot{M} &: \text{Accretion rate} \tag{1}\\
G &: \text{Gravitational constant} \tag{2}\\
M &: \text{Mass of the accreting object} \tag{3}\\
\rho_\infty &: \text{Density of the ambient medium (far from the accreting object)} \tag{4}\\
v_\infty &: \text{Relative velocity between the object and the ambient medium} \tag{5}\\
c_s &: \text{Sound speed of the ambient medium} \tag{6}
\end{align}$$

Since I am working with AGN host galaxies, some of the parameters including ρ_∞, v_∞, and c_s are constrained. However, they are also kind of randomized within the range.

I was thinking if I should use a constant or use a randomization function. However, since the blackhole mass are already an input, I will keep these three fixed first. Later on, I can also test how these values impact the output accretion rate solar masses per year. I made the default values $\rho_\infty = 10^{-13} kg/m^3$, $v_\infty = 5 \times 10^7 m/s$, and $c_s = 5 \times 10^5 m/s$ just to start with.

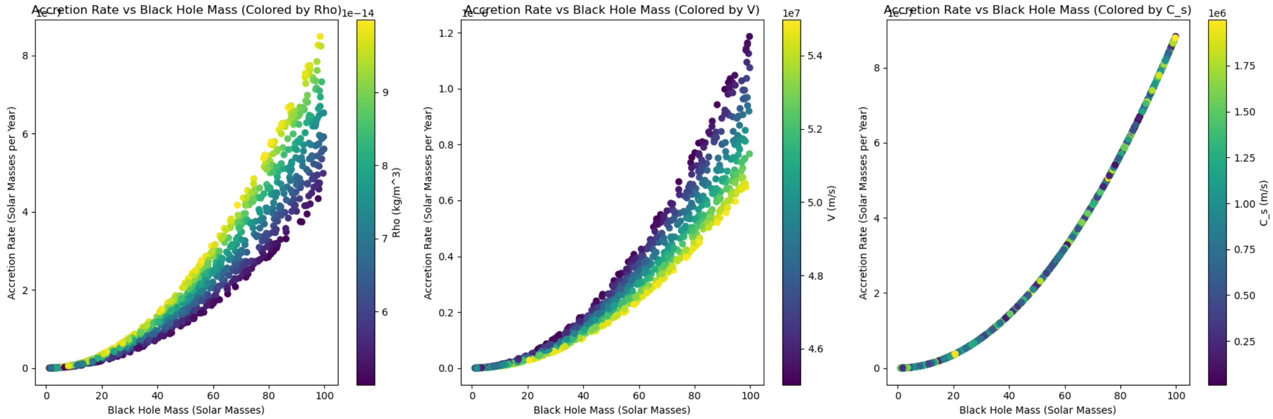
In [4]:
```
# based on this, we can define a accretion rate function called calculate_ac
# returns the accretion rate solar masses per year
def calculate_accretion_rate(black_hole_mass,
                             rho = np.random.uniform(1e-14, 1e-13),
                             v = np.random.uniform(4.5e7, 5.5e7),
                             c_s = np.random.uniform(1e4, 2e6)):
    '''
    The input black hole mass should in the unit of solar mass
    $\rho_\infty = 10^{-13} kg/m^3$, $v_\infty = 5 \times 10^7 m/s$, and $c_
    '''

    G = 6.67430e-11 * u.m**3 / u.kg / u.s**2 # gravitational constant with u
    M = black_hole_mass * 1.98847e30 * u.kg  # solar masses to kg

    rho = rho * u.kg / u.m**3  # density of the accreting material in kg/m^3
    v = v * u.m / u.s  # relative velocity in m/s
    c_s = c_s * u.m / u.s # sound speed in the material in m/s

    # Bondi-Hoyle-Lyttleton accretion rate
    accretion_rate = (4 * np.pi * G**2 * M**2 * rho) / ((v**2 + c_s**2)**(3/

    # Convert accretion rate to solar masses per year
    accretion_rate_solar_masses_per_year = accretion_rate.to(u.kg / u.s) / (

    return accretion_rate_solar_masses_per_year
```

In [5]:
```
# Test the function calculate_accretion_rate
print(calculate_accretion_rate(100))
```

```
8.744656068435424e-15 yr / s
```

To test if this function works well, and if rho, c, c_s contributes a lot to the function (if these parameters impacts it significantly, then we want to reconsider the definition of the function)

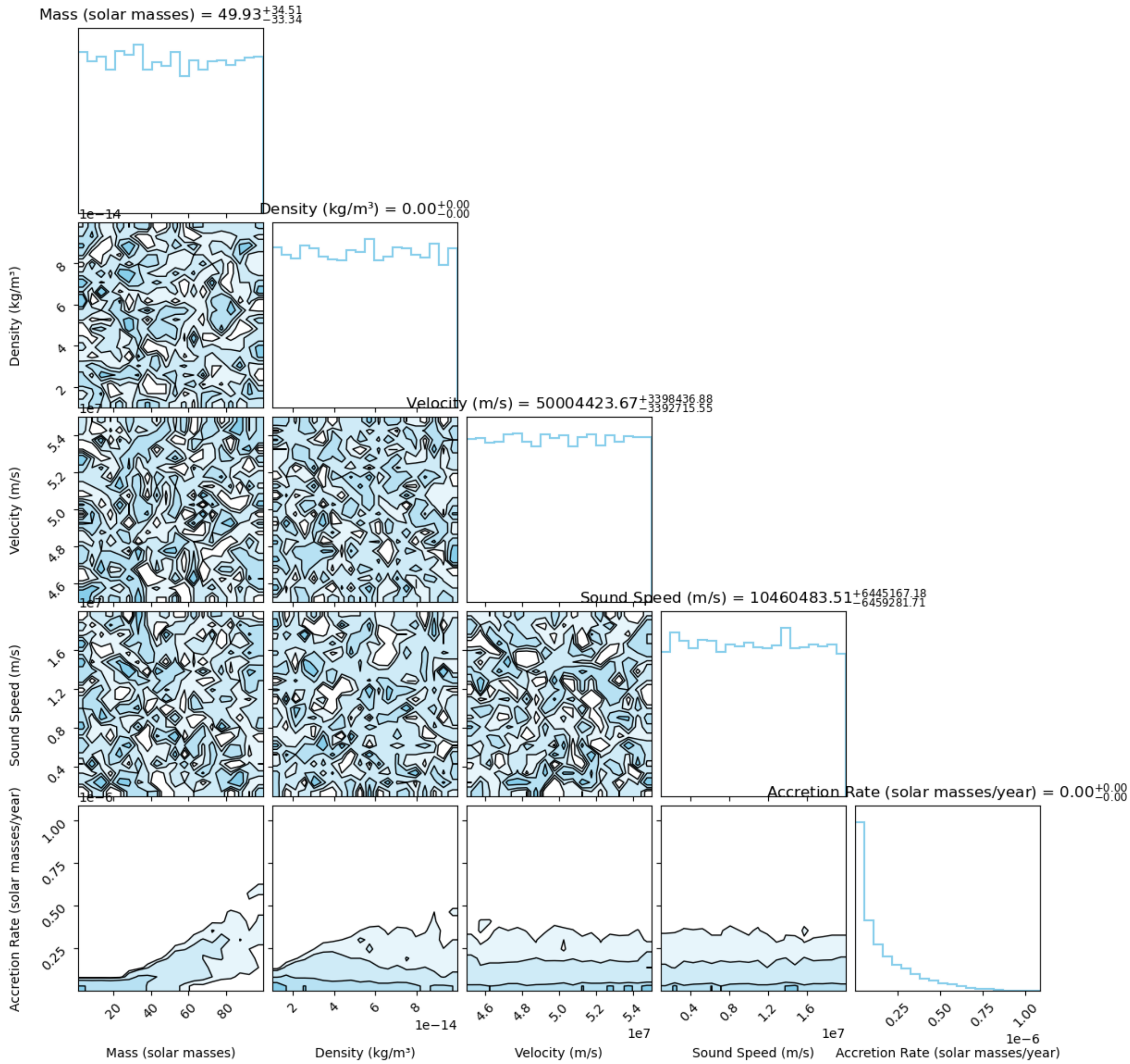
In [6]:
```
'''Plot Accretion Rate in term of Black Hole Masses with varies Parameters''
# Randomize one parameter at a time, keeping the others fixed
num_points = 1000
fixed_rho = 1e-13  # Fixed density
fixed_v = 5e7      # Fixed relative velocity
fixed_c_s = 1e5    # Fixed sound speed

# Randomize black hole masses
black_hole_masses = np.random.uniform(1, 100, num_points)

# Randomizing rho while keeping v and c_s fixed
rho_values = np.random.uniform(5e-14, 1e-13, num_points)
accretion_rates_rho = [calculate_accretion_rate(mass, rho=rho, v=fixed_v, c_

# Randomizing v while keeping rho and c_s fixed
v_values = np.random.uniform(4.5e7, 5.5e7, num_points)
accretion_rates_v = [calculate_accretion_rate(mass, rho=fixed_rho, v=v, c_s=

# Randomizing c_s while keeping rho and v fixed
c_s_values = np.random.uniform(1e4, 2e6, num_points)
accretion_rates_c_s = [calculate_accretion_rate(mass, rho=fixed_rho, v=fixed



plt.figure(figsize=(18, 6))

# Scatter plot for rho
plt.subplot(131)
plt.scatter(black_hole_masses, accretion_rates_rho, c=rho_values, cmap='viri
plt.colorbar(label='Rho (kg/m^3)')
plt.xlabel('Black Hole Mass (Solar Masses)')
plt.ylabel('Accretion Rate (Solar Masses per Year)')
plt.title('Accretion Rate vs Black Hole Mass (Colored by Rho)')

# Scatter plot for v
plt.subplot(132)
plt.scatter(black_hole_masses, accretion_rates_v, c=v_values, cmap='viridis'
plt.colorbar(label='V (m/s)')
plt.xlabel('Black Hole Mass (Solar Masses)')
plt.ylabel('Accretion Rate (Solar Masses per Year)')
plt.title('Accretion Rate vs Black Hole Mass (Colored by V)')

# Scatter plot for c_s
plt.subplot(133)
plt.scatter(black_hole_masses, accretion_rates_c_s, c=c_s_values, cmap='viri
plt.colorbar(label='C_s (m/s)')
plt.xlabel('Black Hole Mass (Solar Masses)')
plt.ylabel('Accretion Rate (Solar Masses per Year)')
plt.title('Accretion Rate vs Black Hole Mass (Colored by C_s)')

plt.tight_layout()
plt.show()
```



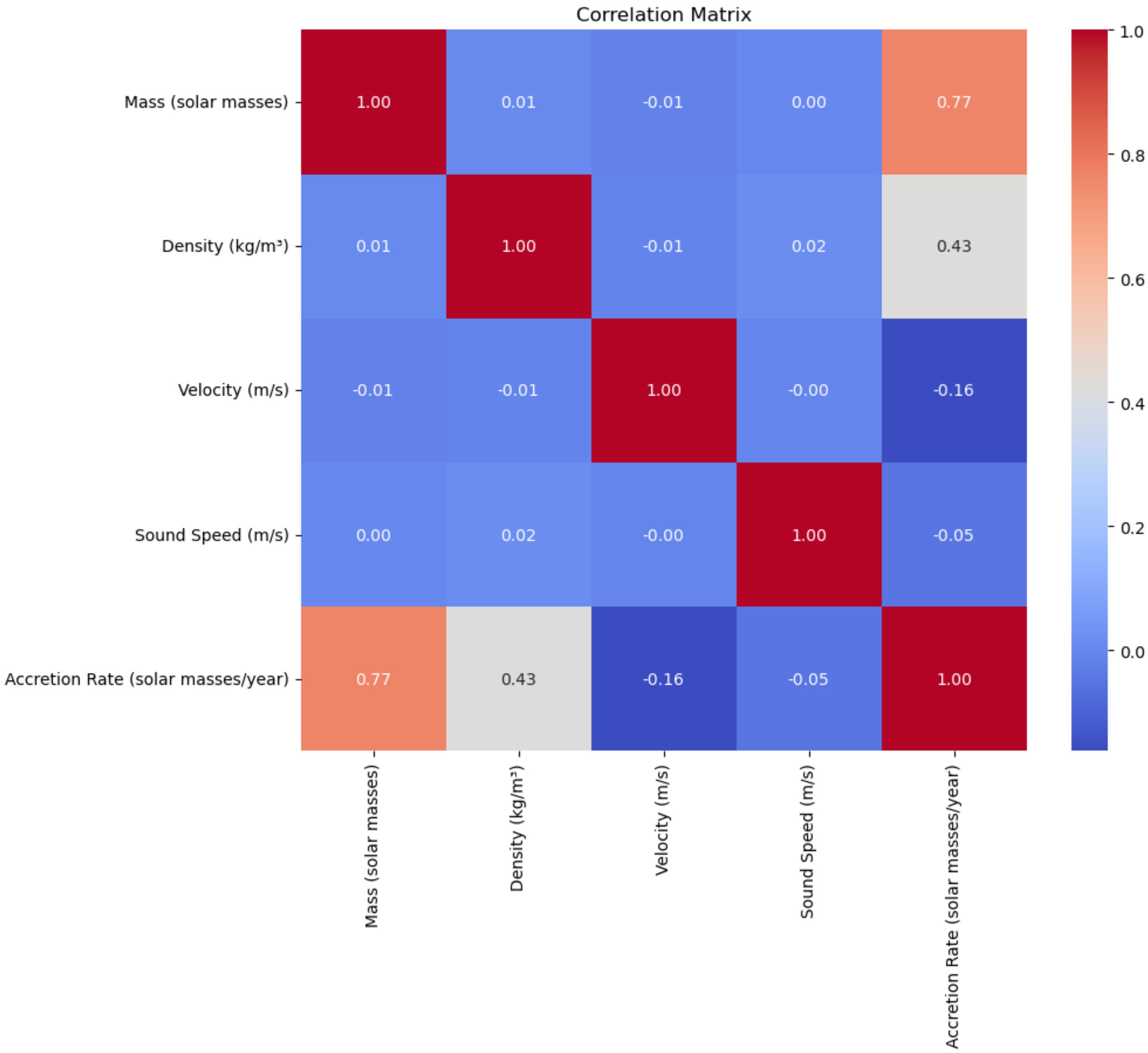
According to the output plots, adjust the randomization to an acceptable range, and update the function calculate_accretion_rate with these randomization range.

In [7]:
```
# observe the correlation between black hole mass, density of ambient object
# velocity between the object and the ambient medium, and Sound speed of the
def pearson_correlation(x, y):
    """
    Calculates the Pearson correlation coefficient between two arrays x and
    """
    mean_x = np.mean(x)
    mean_y = np.mean(y)
    numerator = np.sum((x - mean_x) * (y - mean_y))
    denominator = np.sqrt(np.sum((x - mean_x)**2) * np.sum((y - mean_y)**2))

    return numerator / denominator if denominator != 0 else 0

# Monte Carlo Simulation Parameters
num_simulations = 10000
black_hole_mass_range = (1, 100) # in solar masses
rho_range = (1e-14, 1e-13) # in kg/m^3
v_range = (4.5e7, 5.5e7) # in m/s
c_s_range = (1e6, 2e7) # in m/s

masses = np.random.uniform(*black_hole_mass_range, num_simulations)
rhos = np.random.uniform(*rho_range, num_simulations)
vs = np.random.uniform(*v_range, num_simulations)
c_ss = np.random.uniform(*c_s_range, num_simulations)
accretion_rates = np.array([calculate_accretion_rate(m, rho, v, c_s) for m,

data = np.vstack([masses, rhos, vs, c_ss, accretion_rates]).T

# Labels for each parameter
labels = ['Mass (solar masses)', 'Density (kg/m³)', 'Velocity (m/s)', 'Sound

figure = corner.corner(data, labels=labels, show_titles=True, title_kwargs={
                       color='skyblue',
                       hist_kwargs={'density': True, 'histtype': 'step', 'li
                       plot_density=True,
                       plot_datapoints=False,
```

```
                             fill_contours=True,
                             contour_kwargs={'colors': 'black', 'linewidths': 1})
plt.show()

# Monte Carlo Simulation Parameters
num_simulations = 10000
black_hole_mass_range = (1, 100)  # in solar masses
rho_range = (1e-14, 1e-13)  # in kg/m^3
v_range = (4.5e7, 5.5e7)  # in m/s
c_s_range = (1e6, 2e7)  # in m/s

# Generating random data for each parameter
masses = np.random.uniform(*black_hole_mass_range, num_simulations)
rhos = np.random.uniform(*rho_range, num_simulations)
vs = np.random.uniform(*v_range, num_simulations)
c_ss = np.random.uniform(*c_s_range, num_simulations)

# Calculating accretion rates
accretion_rates = np.array([calculate_accretion_rate(m, rho, v, c_s) for m,

# Stacking the data and calculating the correlation matrix
data = np.vstack([masses, rhos, vs, c_ss, accretion_rates]).T
correlation_matrix = np.array([[pearson_correlation(data[:, i], data[:, j])

# Labels for each parameter
labels = ['Mass (solar masses)', 'Density (kg/m³)', 'Velocity (m/s)', 'Sound

# Creating the correlation matrix heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, xticklabels=labels, yticklabels=
plt.title('Correlation Matrix')
plt.show()
```



Mass (solar masses) = $49.93^{+34.51}_{-33.34}$
Density (kg/m³) = $0.00^{+0.00}_{-0.00}$
Velocity (m/s) = $50004423.67^{+3398436.88}_{-3392715.55}$
Sound Speed (m/s) = $10460483.51^{+6445167.18}_{-6459281.71}$
Accretion Rate (solar masses/year) = $0.00^{+0.00}_{-0.00}$
Density (kg/m³)
Velocity (m/s)
Sound Speed (m/s)
Accretion Rate (solar masses/year)
Mass (solar masses)
1e−14
1e7
1e−6



Here I run 10000 times monte carlo simulation to explore the correlation between our interested parameters. This is as what was expected: the mass of black hole contributes significantly to the accretion rate. However, note that the density also has a relatively high correlation with the accretion rate comparing to other parameters, so I should keep this in mind in the future work.

2. Radiation

Since radiation is emitted by the accretion process as luminosity, it is important to consider the radiation in doing the modeling. The luminosity is typically calculated by (https://en.wikipedia.org/wiki/Luminosity#:~:text=Luminosity%20is%20an%20absolute%20r

$$Radiation = \epsilon \times \dot{M} \times 1.98847 \times 10^{30} \times c^2$$

where

$$\begin{aligned} \dot{M} &: \text{Efficiency factor - typically around 0.1 for non-rotating black holes} & (7) \\ c &: \text{speed of light} & (8) \\ R_{acc} &: \text{accretion rate} & (9) \end{aligned}$$

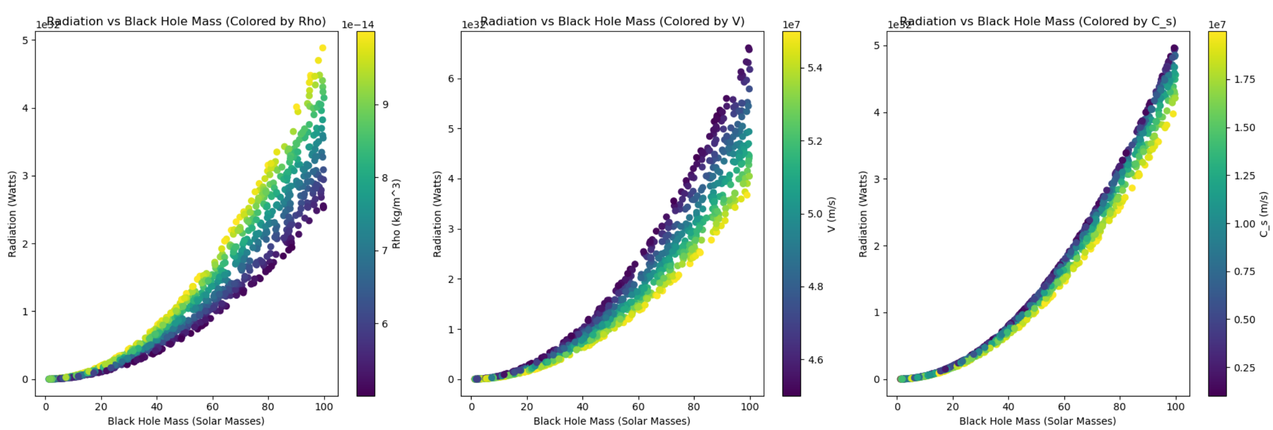
In [8]:
```
def calculate_radiation(accretion_rate, epsilon = 0.1):
    '''
    The accretion rate is obtained from the previous function calculate_accr
    '''
    c = 299792458 * u.m / u.s

    # calculate the luminosity (or radiation)
    radiation = epsilon * accretion_rate * 1.98847e30 * c**2  # Convert accr

    return radiation
```

In [9]:
```
# Test the function calculate_radiation
print(calculate_radiation(calculate_accretion_rate(100)))
```

```
1.5627992069740066e+32 m2 yr / s3
```

In [10]:
```
'''Plot Radiation in term of Black Hole Masses with varies Parameters'''
# Randomize the parameters and calculate accretion rates and radiation
num_points = 1000
fixed_rho = 1e-13  # Fixed density
fixed_v = 5e7      # Fixed relative velocity
fixed_c_s = 5e6    # Fixed sound speed

# Randomize black hole masses
black_hole_masses = np.random.uniform(1, 100, num_points)

# Randomizing rho while keeping v and c_s fixed
rho_values = np.random.uniform(5e-14, 1e-13, num_points)
accretion_rates_rho = [calculate_accretion_rate(mass, rho=rho, v=fixed_v, c_
radiation_rho = [calculate_radiation(rate) for rate in accretion_rates_rho]

# Randomizing v while keeping rho and c_s fixed
v_values = np.random.uniform(4.5e7, 5.5e7, num_points)
accretion_rates_v = [calculate_accretion_rate(mass, rho=fixed_rho, v=v, c_s=
```

```
radiation_v = [calculate_radiation(rate) for rate in accretion_rates_v]

# Randomizing c_s while keeping rho and v fixed
c_s_values = np.random.uniform(1e6, 2e7, num_points)
accretion_rates_c_s = [calculate_accretion_rate(mass, rho=fixed_rho, v=fixed
radiation_c_s = [calculate_radiation(rate) for rate in accretion_rates_c_s]

radiation_rho = [x.value for x in radiation_rho]
radiation_v = [x.value for x in radiation_v]
radiation_c_s = [x.value for x in radiation_c_s]

# Plotting the results
plt.figure(figsize=(18, 6))

# Scatter plot for radiation vs black hole mass (Colored by Rho)
plt.subplot(131)
plt.scatter(black_hole_masses, radiation_rho, c=rho_values, cmap='viridis')
plt.colorbar(label='Rho (kg/m^3)')
plt.xlabel('Black Hole Mass (Solar Masses)')
plt.ylabel('Radiation (Watts)')
plt.title('Radiation vs Black Hole Mass (Colored by Rho)')

# Scatter plot for radiation vs black hole mass (Colored by V)
plt.subplot(132)
plt.scatter(black_hole_masses, radiation_v, c=v_values, cmap='viridis')
plt.colorbar(label='V (m/s)')
plt.xlabel('Black Hole Mass (Solar Masses)')
plt.ylabel('Radiation (Watts)')
plt.title('Radiation vs Black Hole Mass (Colored by V)')

# Scatter plot for radiation vs black hole mass (Colored by C_s)
plt.subplot(133)
plt.scatter(black_hole_masses, radiation_c_s, c=c_s_values, cmap='viridis')
plt.colorbar(label='C_s (m/s)')
plt.xlabel('Black Hole Mass (Solar Masses)')
plt.ylabel('Radiation (Watts)')
plt.title('Radiation vs Black Hole Mass (Colored by C_s)')

plt.tight_layout()
plt.show()
```



3. jet production

The jet which emits energy is determined by many factor and is usually very hard to model, so we here assume a simpler case where the jets are produced when the accretion rate and radiation exceed certain limit. We assume the limits to be:

1. accretion rate thershold = 10^{-4} solar mass per year
2. radiation thershold = 10^{36} watts

In [11]:
```
def determine_jet_production(accretion_rate, radiation):
    # Limits when jet production takes place
    accretion_rate_threshold = 1e-4  # in solar masses per year
    radiation_threshold = 1e36

    # Determine jet production
    jets = accretion_rate > accretion_rate_threshold and radiation > radiati
    return jets
    # returns true or false
```

In [12]:
```
# Examples for determine_jet_production
print(determine_jet_production(1e-7, 1e20)) # neither parameter exceeds the
print(determine_jet_production(1, 1e30)) # the accretion rate exceeds the li
print(determine_jet_production(1e-5, 1e40)) # the radiation exceeds the limi
print(determine_jet_production(1, 1e40)) # both exceeds the limit
```

```
False
False
False
True
```

combining the three function

In [13]:
```
def accretion_model(black_hole_mass, accretion_radius, other_parameters):
    accretion_rate = calculate_accretion_rate(black_hole_mass, accretion_rad
    radiation = calculate_radiation(accretion_rate, other_parameters)
    jets = determine_jet_production(accretion_rate, radiation, other_paramet

    return accretion_rate, radiation, jets
```

Now I can add these functions to the original class as the first part of the code:

In [14]:
```
class Galaxy:
    '''initialize the galaxy'''
    def __init__(self, mass, position, velocity, black_hole_mass):
        self.mass = mass
        self.position = position
        self.velocity = velocity
```

```
        self.black_hole_mass = black_hole_mass

    '''black hole accretion model'''
    def calculate_accretion_rate(black_hole_mass,
                                 rho = np.random.uniform(1e-14, 1e-13),
                                 v = np.random.uniform(4.5e7, 5.5e7),
                                 c_s = np.random.uniform(1e6, 2e7)):
        # The input black hole mass should in the unit of solar mass
        G = 6.67430e-11 * u.m**3 / u.kg / u.s**2 # gravitational constant wi
        M = black_hole_mass * 1.98847e30 * u.kg  # solar masses to kg
        rho = rho * u.kg / u.m**3  # density of the accreting material in kg
        v = v * u.m / u.s  # relative velocity in m/s
        c_s = c_s * u.m / u.s # sound speed in the material in m/s
        # Bondi-Hoyle-Lyttleton accretion rate
        accretion_rate = (4 * np.pi * G**2 * M**2 * rho) / ((v**2 + c_s**2)*
        # Convert accretion rate to solar masses per year
        accretion_rate_solar_masses_per_year = accretion_rate.to(u.kg / u.s)
        return accretion_rate_solar_masses_per_year

    def calculate_radiation(accretion_rate, epsilon = 0.1):
        # The accretion rate is obtained from the previous function calculat
        c = 299792458 * u.m / u.s
        # calculate the luminosity (or radiation)
        radiation = epsilon * accretion_rate * 1.98847e30 * c**2  # Convert
        return radiation

    def accretion_model(black_hole_mass, accretion_radius, other_parameters)
        accretion_rate = calculate_accretion_rate(black_hole_mass, accretion
        radiation = calculate_radiation(accretion_rate, other_parameters)
        jets = determine_jet_production(accretion_rate, radiation, other_par
        return accretion_rate, radiation, jets

    # Gas Dynamics and Interstellar Medium
    def gas_dynamics(galaxy):
        '''
        This models the impliment model of the dynamics of gas in and around
        The processes like cooling, heating, and interaction with AGN radiat
        '''
        pass

    # Galaxy Interaction and Dynamics
    def update_galaxy_dynamics(galaxies):
        '''
        Implement gravitational interactions between galaxies.
        Update positions and velocities based on gravitational forces.
        '''
        pass
```

B) Cooling and Heating

According to my research of other people's work, the fully modeled dynamic of mass cannot be easily computed from a local computer and requires much more computation work. Thus, for my work, I will be creating a easier model with more idealized condition than what is actually happening in AGN galaxies but keep all the key concepts in AGN gas dynamics simulation.

The temperature change within the galaxy is another important thing to consider. From previous study (Li et al 2016), they analyzed a battle between radiative cooling and AGN heating. Thus the cooling and heating are happening at the same time within the AGN host galaxy. That being said, we want to set different rate of cooling and heating in terms of other parameters.

This is complex as a lot of parameters could contribute to the cooling/ heating rates, such as mass, cooling coefficient, environment, AGN activity, and many more. We here consider some of the most significant factor in cooling and heating, which are:

1. **density**: Cooling processes in the ISM are significantly influenced by the gas's density, and it is usually proportional to density square
2. **cooling coefficient**
3. **temperature**: Cooling processes in the ISM are inversely proportional to temperature
4. **time**: The rate might decrease over time as the galaxy loses energy
5. **composition**: The composition of AGN galaxies are really hard to determine, I will keep this away from the simulation for now, but may come back to it as this project becomes more developed
6. **metallicity**
7. **star formation rate**
8. **AGN luminosity**: this was calculated above with the radiation

In [15]:
```
# combining what has been discussed above, cooling rate is proportional to c
# and inversely proportional to time and temperature
def cooling_function(temperature):
    if temperature < 1e4:
        # Low-temperature regime
        new_temp = 1e-22 * temperature
        return new_temp
    else:
        # High-temperature regime
        new_temp = 1e-24 * temperature**1.5
        return new_temp

def heating_function(star_formation_initial, agn_luminosity_initial, density
    decay_factor = 0.05  # Represents a decay over time
    # Star formation rate and AGN luminosity decrease over time
    star_formation_rate = star_formation_initial / (1 + decay_factor * time)
    agn_luminosity = agn_luminosity_initial / (1 + decay_factor * time)

    return 1e-35 * (star_formation_rate + agn_luminosity) * density

def cooling_rate(new_temp, density):
    cooling_rate = new_temp * density**2
    return cooling_rate

def ism_temperature_evolution(initial_temperature, density, metallicity, sta
    temperature = initial_temperature

    for time in range(total_time):
        # cooling and heating rates
        cooling_rate = cooling_function(temperature) * density**2
        heating_rate = heating_function(star_formation_rate, agn_luminosity,

        # Update temperature
        temperature_change = (heating_rate - cooling_rate) * timestep
        temperature += temperature_change

        if temperature < 10:  # 10 K is a reasonable lower limit for ISM tem
            temperature = 10

    return temperature
```

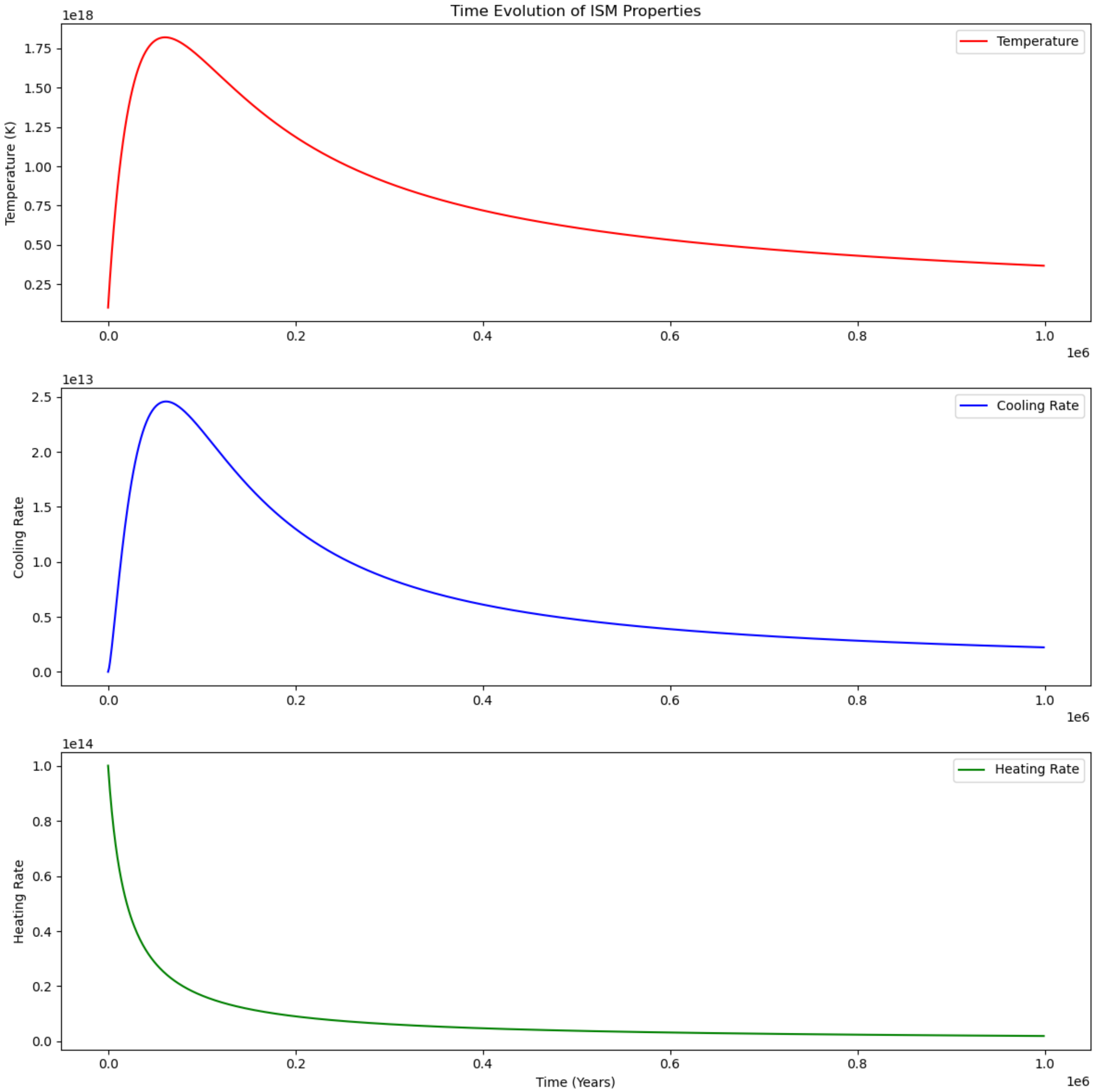
In [16]:
```
# Test the function
initial_temperature = 1000000  # in Kelvin
density = 1.0e5  # in particles per cubic centimeter
metallicity = 1.0  # Solar metallicity
star_formation_rate = 2.0  # Solar masses per year
agn_luminosity = 1e44  # in ergs per second
timestep = 1e3  # in years
total_time = 1000  # number of timesteps
final_temperature = ism_temperature_evolution(initial_temperature, density,
print(f"Final ISM Temperature: {final_temperature} K")
```

```
Final ISM Temperature: 3.669852783696508e+17 K
```

Week 2 (Milestone2 start)

In [17]:
```
def ism_temperature_evolution_over_time(initial_temperature,
                                        density,
                                        metallicity,
                                        star_formation_initial,
                                        agn_luminosity_initial,
                                        timestep,
                                        total_time):
    temperature = initial_temperature
    temperatures, cooling_rates, heating_rates = [], [], []

    for time in range(total_time):
        cooling_rate = cooling_function(temperature) * density**2
        heating_rate = heating_function(star_formation_initial, agn_luminosi

        temperature_change = (heating_rate - cooling_rate) * timestep
        # print(heating_rate, cooling_rate)
        temperature += temperature_change
        if temperature < 10:
            temperature = 10

        temperatures.append(temperature)
        cooling_rates.append(cooling_rate)
        heating_rates.append(heating_rate)

    return temperatures, cooling_rates, heating_rates
```

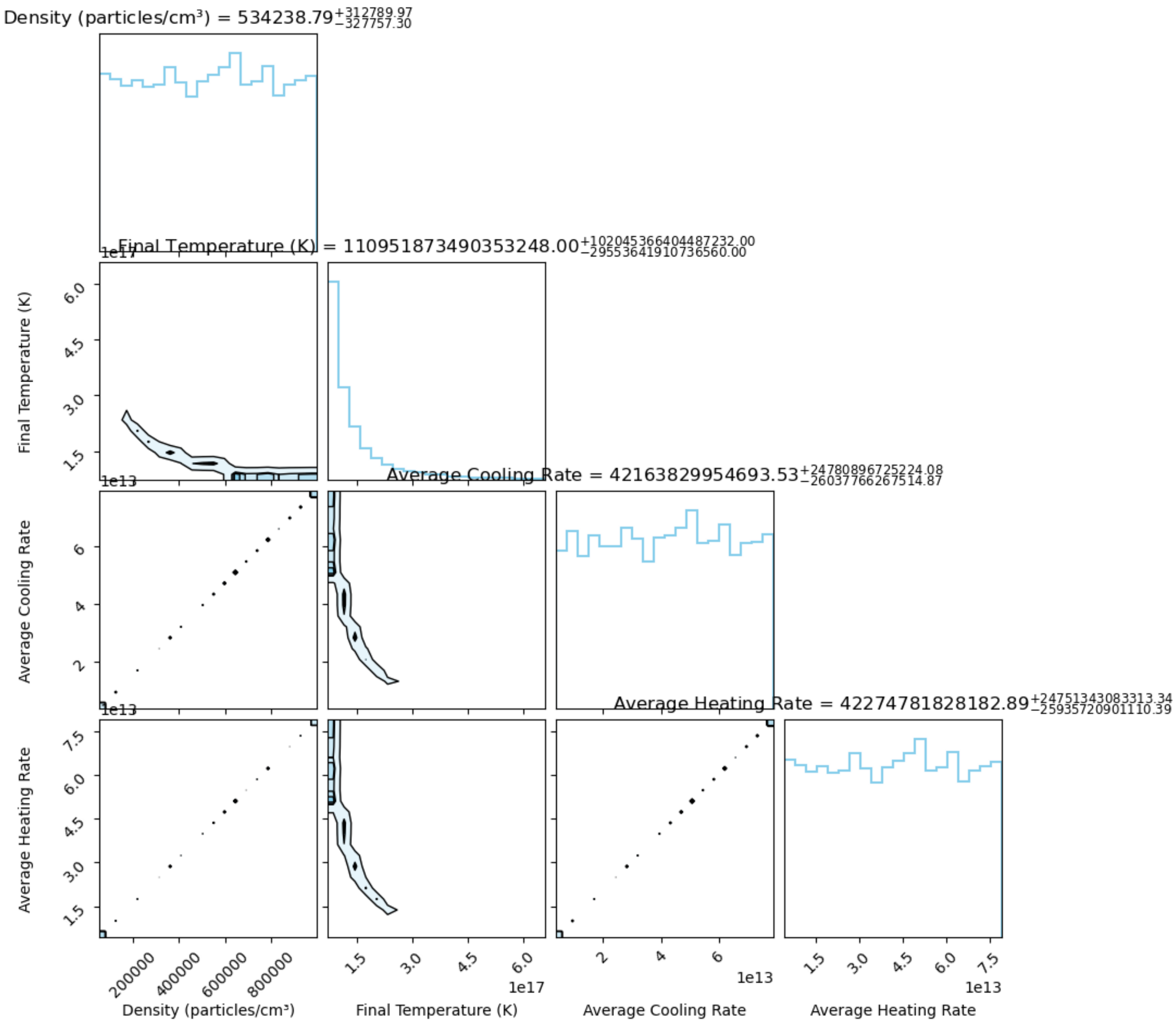
In [18]:
```
# simulation
temperatures, cooling_rates, heating_rates = ism_temperature_evolution_over_
time_axis = [timestep * i for i in range(total_time)]
plt.figure(figsize=(12, 12))

# Temperature
plt.subplot(3, 1, 1)
plt.plot(time_axis, temperatures, label='Temperature', color='red')
plt.ylabel('Temperature (K)')
plt.title('Time Evolution of ISM Properties')
plt.legend()

# Cooling
plt.subplot(3, 1, 2)
plt.plot(time_axis, cooling_rates, label='Cooling Rate', color='blue')
plt.ylabel('Cooling Rate')
plt.legend()

# Heating
plt.subplot(3, 1, 3)
plt.plot(time_axis, heating_rates, label='Heating Rate', color='green')
plt.ylabel('Heating Rate')
plt.xlabel('Time (Years)')
plt.legend()

plt.tight_layout()
plt.show()
```



From this plot, it seems like the temperature follows similar evolution as the cooling rate, however, when I did the attempts of changing the value of density, this trend varies a lot. This is impressive and should be further explored, so I want to plot out the evolution of ISM temperature and cooling/heating rates corresponding to different density environment. This is crucial as it reveals potential environmental impact on AGN galaxy evolution. Some methods I could use is to

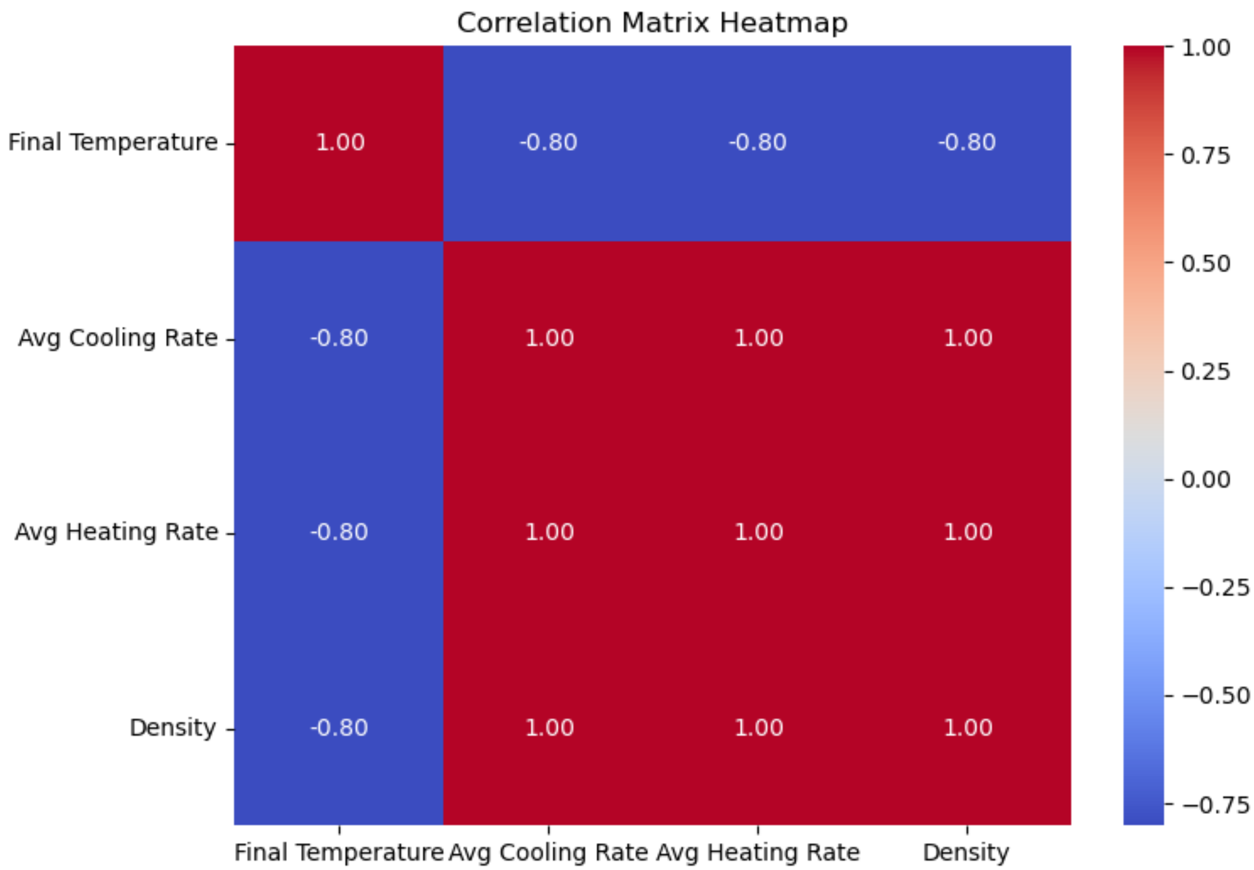
1. again, use the monte carlo simulation to find the correlation, and
2. plot out the rates and temperature at different density to observe the trends.

In [19]:
```
# monte carlo simulation to find the correlation between density and rates/t
num_simulations = 10000
density_range = (5.5e4, 1e6)

initial_temperature = 1000000  # in Kelvin
metallicity = 1.0  # Solar metallicity
star_formation_rate = 2.0  # Solar masses per year
agn_luminosity = 1e44  # in ergs per second
timestep = 1e3  # in years
total_time = 1000  # number of timesteps

densities = np.random.uniform(*density_range, num_simulations)
final_temperatures = []
average_cooling_rates = []
average_heating_rates = []

for density in densities:
    temperatures, cooling_rates, heating_rates = ism_temperature_evolution_c
        initial_temperature, density, metallicity, star_formation_rate,
        agn_luminosity, timestep, total_time
    )
    final_temperatures.append(temperatures[-1])
    average_cooling_rates.append(np.mean(cooling_rates))
    average_heating_rates.append(np.mean(heating_rates))
simulation_data = np.vstack([densities, final_temperatures, average_cooling_
corner_labels = ['Density (particles/cm³)', 'Final Temperature (K)', 'Averag
corner_figure = corner.corner(simulation_data, labels=corner_labels, show_ti
                              color='skyblue',
                              hist_kwargs={'density': True, 'histtype': 'ste
                              plot_density=True,
                              plot_datapoints=False,
                              fill_contours=True,
                              contour_kwargs={'colors': 'black', 'linewidths
plt.show()
```

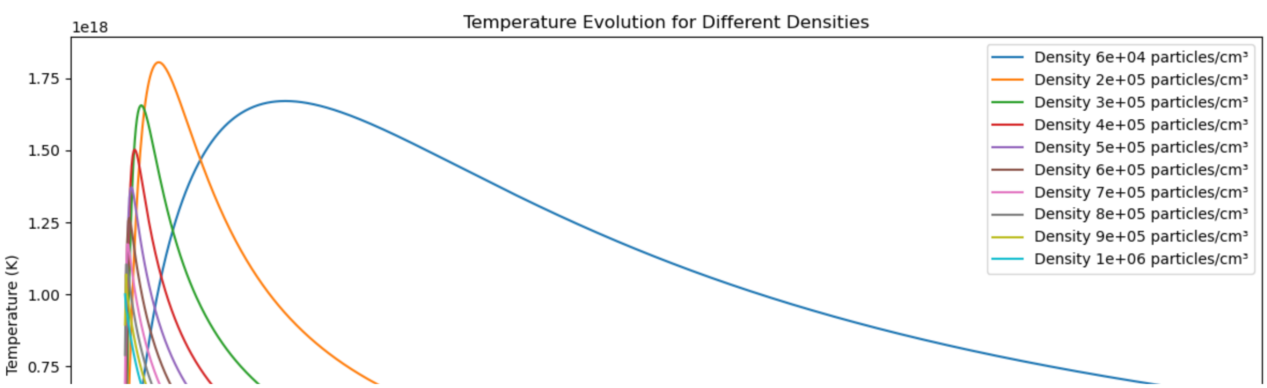


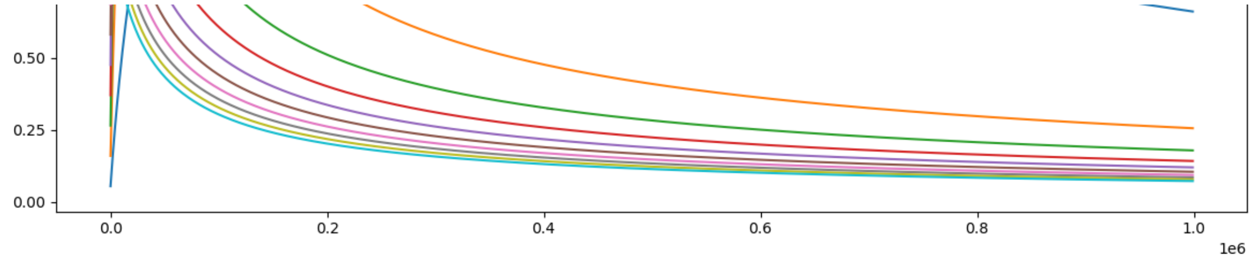
In [20]:

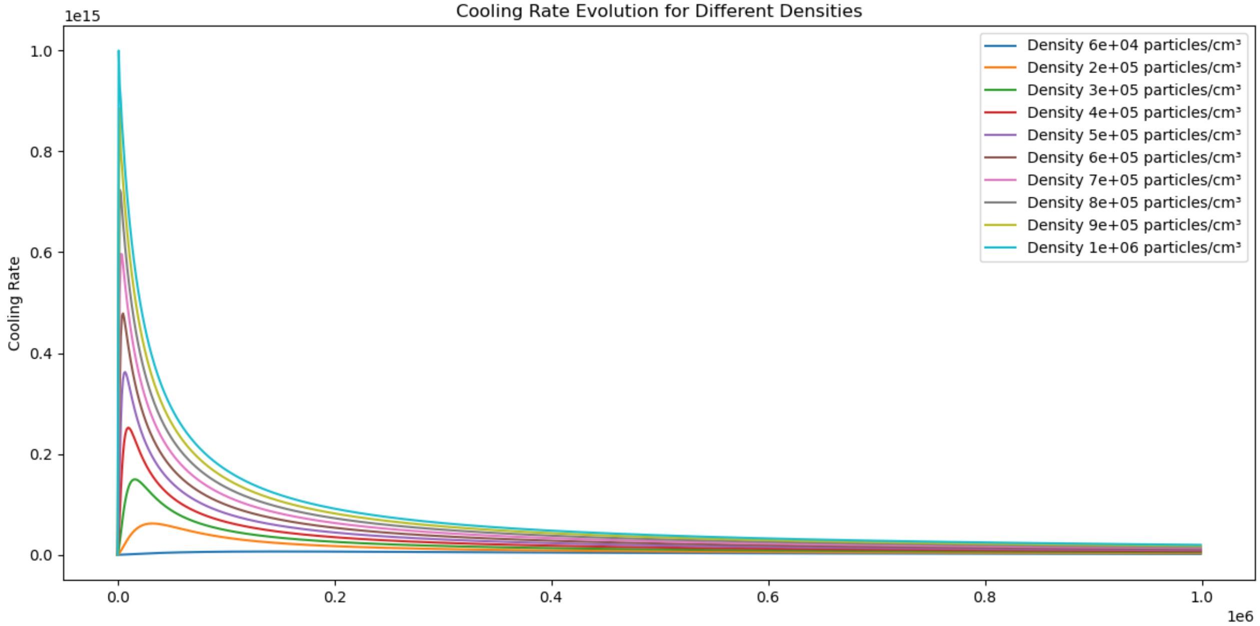
```
# the heatmap for correlations between
# temperature, cooling rate, heating rate, and density
simulation_data_array = np.array([final_temperatures, average_cooling_rates,
correlation_matrix = np.corrcoef(simulation_data_array)
heatmap_labels = ['Final Temperature', 'Avg Cooling Rate', 'Avg Heating Rate

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, xticklabels=heatmap_labels, ytic
plt.title('Correlation Matrix Heatmap')
plt.show()
```

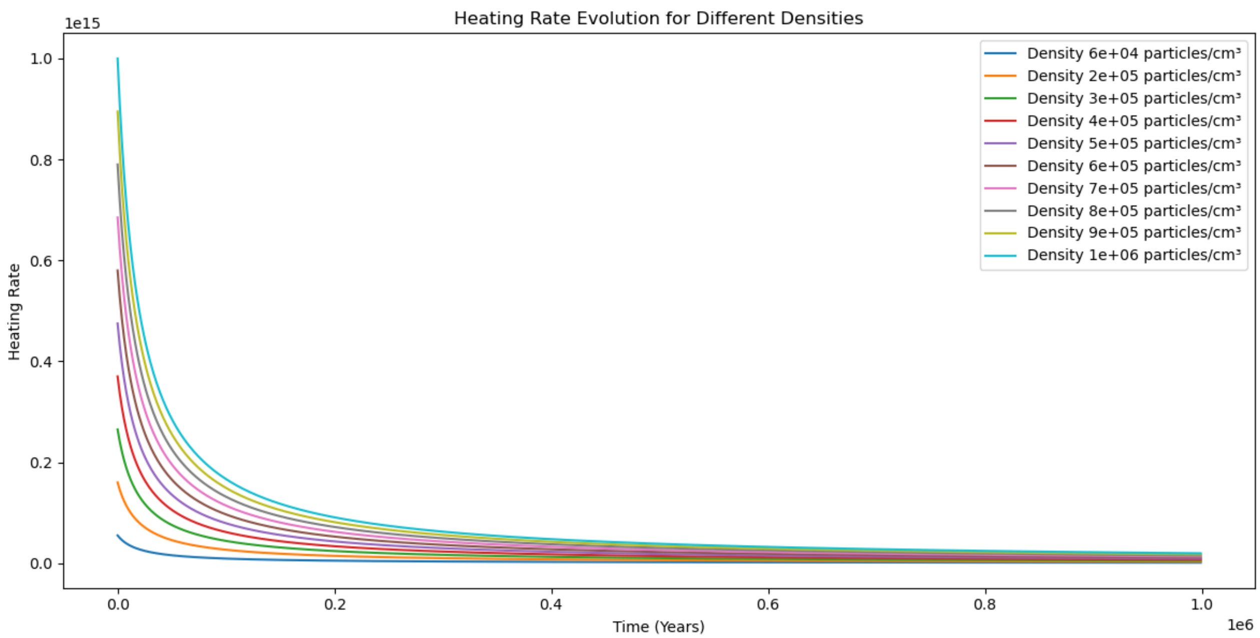

Correlation Matrix Heatmap
Final Temperature
Avg Cooling Rate
Avg Heating Rate
Density
1.00
-0.80
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

In [21]:
```
selected_densities = np.linspace(density_range[0], density_range[1], 10)

plt.figure(figsize=(12, 18))

for density in selected_densities:
    temperatures, cooling_rates, heating_rates = ism_temperature_evolution_c
        initial_temperature, density, metallicity, star_formation_rate,
        agn_luminosity, timestep, total_time
    )
    time_axis = [timestep * i for i in range(total_time)]

    # Temperature
    plt.subplot(3, 1, 1)
    plt.plot(time_axis, temperatures, label=f'Density {density:.0e} particle

    # cooling
    plt.subplot(3, 1, 2)
    plt.plot(time_axis, cooling_rates, label=f'Density {density:.0e} particl

    # heating
    plt.subplot(3, 1, 3)
    plt.plot(time_axis, heating_rates, label=f'Density {density:.0e} particl

plt.subplot(3, 1, 1)
plt.ylabel('Temperature (K)')
plt.title('Temperature Evolution for Different Densities')
plt.legend()

plt.subplot(3, 1, 2)
plt.ylabel('Cooling Rate')
plt.title('Cooling Rate Evolution for Different Densities')
plt.legend()

plt.subplot(3, 1, 3)
plt.ylabel('Heating Rate')
plt.xlabel('Time (Years)')
plt.title('Heating Rate Evolution for Different Densities')
plt.legend()

plt.tight_layout()
plt.show()
```






Cooling Rate Evolution for Different Densities
1e15
Cooling Rate
1e6
Density 6e+04 particles/cm³
Density 2e+05 particles/cm³
Density 3e+05 particles/cm³
Density 4e+05 particles/cm³
Density 5e+05 particles/cm³
Density 6e+05 particles/cm³
Density 7e+05 particles/cm³
Density 8e+05 particles/cm³
Density 9e+05 particles/cm³
Density 1e+06 particles/cm³


Heating Rate Evolution for Different Densities
1e15
Heating Rate
Time (Years)
1e6
Density 6e+04 particles/cm³
Density 2e+05 particles/cm³
Density 3e+05 particles/cm³
Density 4e+05 particles/cm³
Density 5e+05 particles/cm³
Density 6e+05 particles/cm³
Density 7e+05 particles/cm³
Density 8e+05 particles/cm³
Density 9e+05 particles/cm³
Density 1e+06 particles/cm³

As density increases, the heating rate increases, the cooling rate increases as well, and the overall temperature peaks lower and decays faster.

Also, from the correlation matrix heat map, it is clear how things are closely related in this cooling and heading system. It is as expected because environment itself is a very deterministic factor.

The density = $1e5$ to $1e6$ is a fair estimation for the average density inside of an AGN galaxy, but note that the density varies a lot in different parts of a galaxy. For my targets which are AGN galaxies, they are extremely powerful towards the center mainly due to the super massive blackhole. Thus the density at broad line region (the region close to black hole) of AGN galaxies can exceed $1e9$ particles per cubic centimeter. As the radius moves away from the center of the galaxy, the narrow line region typically has densities in the range of $1e2$ to $1e6$.

To make the system more realistic, I can write a function of density in terms of the radius. The typical relationship between density and radius can be expressed as

$$\rho(r) = \rho_0(\frac{r}{r_0})^{-\alpha}$$

where

$$\begin{aligned} \rho(r) &: \text{The density as a function of radius from the center} \\ \rho_0 &: \text{the reference density, I here take it to be } 1.0 \times 10^5 \text{ particles per cubic c} \\ r_0 &: \text{reference radius, I here take it to be 1 parsec} \\ r &: \text{the radius from the center, in unit of parsec} \\ \alpha &: \text{the index of this power law, I here take it to be 1.5, this varies for diffe} \end{aligned}$$

In [22]:
```
# we can define the function density
def density_radius(r, rho_0 = 1e5, r_0 = 1, alpha = 1.5):
    """
    Density as a function of radius for an AGN environment.
    r: Radius in parsec
    rho_0: Reference density at reference radius r_0
    r_0: Reference radius
    alpha: Power-law index
    """
    rho_r = rho_0 * (r / r_0) ** (-alpha)
    return rho_r
```

Then, I can update the previous functions:

In [23]:
```
def heating_function(star_formation_initial,
```

```
                    r,
                    time,
                    agn_luminosity_initial = 1e44,
                    rho_0 = 1e5,
                    r_0 = 1,
                    alpha = 1.5):
    decay_factor = 0.05
    star_formation_rate = star_formation_initial / (1 + decay_factor * time)
    agn_luminosity = agn_luminosity_initial / (1 + decay_factor * time)
    density = density_radius(r, rho_0, r_0, alpha)
    new_temp = 1e-35 * (star_formation_rate + agn_luminosity) * density
    return new_temp

def cooling_function(temperature, r, rho_0 = 1e5, r_0 = 1, alpha = 1.5):
    density = density_radius(r, rho_0, r_0, alpha)
    if temperature < 1e4:
        new_temp = 1e-22 * temperature * density**2
    else:
        new_temp = 1e-24 * temperature**1.5 * density**2
    return new_temp

def cooling_rate(new_temp, density):
    cooling_rate = new_temp * density**2
    return cooling_rate

def ism_temperature_evolution(initial_temperature,
                              r,
                              metallicity,
                              star_formation_rate,
                              agn_luminosity, timestep, total_time):
    temperature = initial_temperature
    density = density_radius(r, rho_0, r_0, alpha)
    for time in range(total_time):
        # cooling and heating rates
        cooling_rate = cooling_function(temperature, r, rho_0 = 1e5, r_0 = 1
        heating_rate = heating_function(star_formation_rate, agn_luminosity,
        temperature_change = (heating_rate - cooling_rate) * timestep
        temperature += temperature_change
        if temperature < 10:  # 10 K is a reasonable lower limit for ISM tem
            temperature = 10
    return temperature

def ism_temperature_evolution_over_time(initial_temperature,
                                        r,
                                        rho_0=1e5,
                                        r_0=1,
                                        alpha=1.5,
                                        metallicity=1.0,
                                        star_formation_initial=2.0,
                                        agn_luminosity_initial=1e44,
                                        timestep=1e3,
                                        total_time=10000):
    temperature = initial_temperature
```

```
    temperatures, cooling_rates, heating_rates = [], [], []
    for time in range(total_time):
        density = density_radius(r, rho_0, r_0, alpha)
        cooling_rate = cooling_function(temperature, r, rho_0, r_0, alpha) *
        heating_rate = heating_function(star_formation_initial, agn_luminosi
        temperature_change = (heating_rate - cooling_rate) * timestep
        temperature += temperature_change
        if temperature < 10:
            temperature = 10
        temperatures.append(temperature)
        cooling_rates.append(cooling_rate)
        heating_rates.append(heating_rate)
    return temperatures, cooling_rates, heating_rates
```

In [24]:

```
# Example
r = 2.0  # radius in parsecs
initial_temperature = 100000  # in Kelvin
temperatures, cooling_rates, heating_rates = ism_temperature_evolution_over_
temperatures[:10]
```

Out[24]:

```
[50589.411559869055,
 32810.32404018419,
 23524.171671272197,
 17886.6070103183,
 14148.842176469958,
 11519.171404028086,
 9587.417107373523,
 8089.3831843464095,
 6825.417061792283,
 5758.945645887238]
```

Add these to the previously defined class:

In [25]:

```
class Galaxy:
    '''initialize the galaxy'''
    def __init__(self, mass, position, velocity, black_hole_mass):
        self.mass = mass
        self.position = position
        self.velocity = velocity
        self.black_hole_mass = black_hole_mass

    '''black hole accretion model'''
    def calculate_accretion_rate(black_hole_mass,
                                 rho = np.random.uniform(1e-14, 1e-13),
                                 v = np.random.uniform(4.5e7, 5.5e7),
                                 c_s = np.random.uniform(1e6, 2e7)):
        # The input black hole mass should in the unit of solar mass
        G = 6.67430e-11 * u.m**3 / u.kg / u.s**2 # gravitational constant wi
        M = black_hole_mass * 1.98847e30 * u.kg  # solar masses to kg
        rho = rho * u.kg / u.m**3  # density of the accreting material in kg
        v = v * u.m / u.s  # relative velocity in m/s
        c_s = c_s * u.m / u.s # sound speed in the material in m/s
        # Bondi-Hoyle-Lyttleton accretion rate
```

```
        accretion_rate = (4 * np.pi * G**2 * M**2 * rho) / ((v**2 + c_s**2)*
        # Convert accretion rate to solar masses per year
        accretion_rate_solar_masses_per_year = accretion_rate.to(u.kg / u.s)
        return accretion_rate_solar_masses_per_year

    def calculate_radiation(accretion_rate, epsilon = 0.1):
        # The accretion rate is obtained from the previous function calculat
        c = 299792458 * u.m / u.s
        # calculate the luminosity (or radiation)
        radiation = epsilon * accretion_rate * 1.98847e30 * c**2  # Convert
        return radiation

    def accretion_model(black_hole_mass, accretion_radius, other_parameters)
        accretion_rate = calculate_accretion_rate(black_hole_mass, accretion
        radiation = calculate_radiation(accretion_rate, other_parameters)
        jets = determine_jet_production(accretion_rate, radiation, other_par
        return accretion_rate, radiation, jets

    '''cooling and heating process in different radius of the galaxy'''
    def heating_function(star_formation_initial,
                         r,
                         time,
                         agn_luminosity_initial = 1e44,
                         rho_0 = 1e5,
                         r_0 = 1,
                         alpha = 1.5):
        decay_factor = 0.05
        star_formation_rate = star_formation_initial / (1 + decay_factor * t
        agn_luminosity = agn_luminosity_initial / (1 + decay_factor * time)
        density = density_radius(r, rho_0, r_0, alpha)
        new_temp = 1e-35 * (star_formation_rate + agn_luminosity) * density
        return new_temp

    def cooling_function(temperature, r, rho_0 = 1e5, r_0 = 1, alpha = 1.5):
        density = density_radius(r, rho_0, r_0, alpha)
        if temperature < 1e4:
            new_temp = 1e-22 * temperature * density**2
        else:
            new_temp = 1e-24 * temperature**1.5 * density**2
        return new_temp

    def cooling_rate(new_temp, density):
        cooling_rate = new_temp * density**2
        return cooling_rate

    def ism_temperature_evolution(initial_temperature,
                                  r,
                                  metallicity,
                                  star_formation_rate,
                                  agn_luminosity, timestep, total_time):
        temperature = initial_temperature
        density = density_radius(r, rho_0, r_0, alpha)
        for time in range(total_time):
```

```
            # cooling and heating rates
            cooling_rate = cooling_function(temperature, r, rho_0 = 1e5, r_0
            heating_rate = heating_function(star_formation_rate, agn_luminos
            temperature_change = (heating_rate - cooling_rate) * timestep
            temperature += temperature_change
            if temperature < 10:  # 10 K is a reasonable lower limit for ISM
                temperature = 10
        return temperature

    def ism_temperature_evolution_over_time(initial_temperature,
                                            r,
                                            rho_0=1e5,
                                            r_0=1,
                                            alpha=1.5,
                                            metallicity=1.0,
                                            star_formation_initial=2.0,
                                            agn_luminosity_initial=1e44,
                                            timestep=1e3,
                                            total_time=10000):
        temperature = initial_temperature
        temperatures, cooling_rates, heating_rates = [], [], []
        for time in range(total_time):
            density = density_radius(r, rho_0, r_0, alpha)
            cooling_rate = cooling_function(temperature, r, rho_0, r_0, alph
            heating_rate = heating_function(star_formation_initial, agn_lumi
            temperature_change = (heating_rate - cooling_rate) * timestep
            temperature += temperature_change
            if temperature < 10:
                temperature = 10
            temperatures.append(temperature)
            cooling_rates.append(cooling_rate)
            heating_rates.append(heating_rate)
        return temperatures, cooling_rates, heating_rates

    # Gas Dynamics and Interstellar Medium
    def gas_dynamics(galaxy):
        '''
        This models the impliment model of the dynamics of gas in and around
        The processes like cooling, heating, and interaction with AGN radiat
        '''
        pass

    # Galaxy Interaction and Dynamics
    def update_galaxy_dynamics(galaxies):
        '''
        Implement gravitational interactions between galaxies.
        Update positions and velocities based on gravitational forces.
        '''
        pass
```

C) Spectra Modeling for AGN galaxies

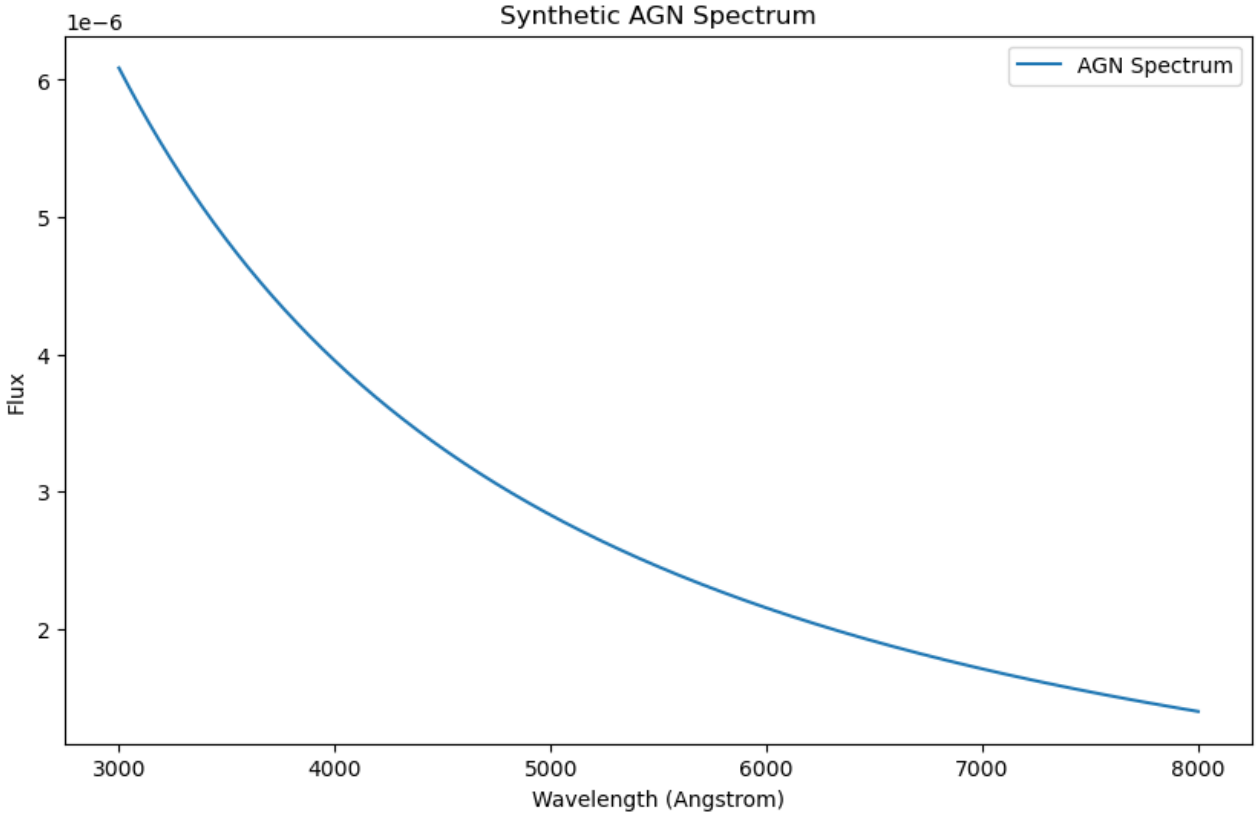
Spectra modeling is significant for studying gakaxies because it gives us information about a various things. Some typical ones includes:

1. **composition and chemical elemental abundances**: from analyzing a spectra of a galaxy, we can tell from the emission and absorption lines which components exists in the galaxies, also, the ratio between certain emission lines give us the chemical elemental abundances, which reveals the history of the galaxy because some element are more produced in certain age or phase of galaxy evolution.
2. **redshift**: by looking at the obvious emission and absorption lines from a galaxy's spectrum, we can compare it with the wavelength of that element in the rest frame spectrum, and by using $1+z=\frac{\lambda_{observed}}{\lambda_e}$ we can find the redshift of the galaxy, and thus how far the galaxy is from us.
3. **stellar population**: since different stars have very different physical properties, we can tell from the spectra which kind of stars do the galaxy have and thus better understand the stellar populations of certain types of galaxies.
4. Spectra modeling also plays critical roles in **understanding black holes of AGN**, **studying dark matter and dark matter halo**, and **providing insights into the temperature and density**. Thus we want to understand how AGN galaxies look like and potentially build AGN galaxy spectrum.

There are several things and steps I need to consider towards doing spectra modeling:

1. I need to **generate a power-law continuum**
2. consider **emission lines**
3. add in **noise**
4. consider **absorption feature** I here use units that is typically used in experimental astrophysics, which are arbitrary unit for flux (we do not actually care about the unit of flux, what we care is the intensity and the ratios)

In [26]:
```
# Step 1
# we first assumed redshift 0 and generalize the power-law continuum
# most of the emission lines and absorption features are within wavelength r
# so I choose the wavelength range from 3000 to 10000
# alpha is the power-law slope
wavelengths = np.linspace(3000, 8000, 10000)
alpha = -1.5
flux_continuum = wavelengths ** alpha

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(wavelengths, flux_continuum, label='AGN Spectrum')
plt.xlabel('Wavelength (Angstrom)')
plt.ylabel('Flux')
plt.title('Synthetic AGN Spectrum')
plt.legend()
plt.show()
```

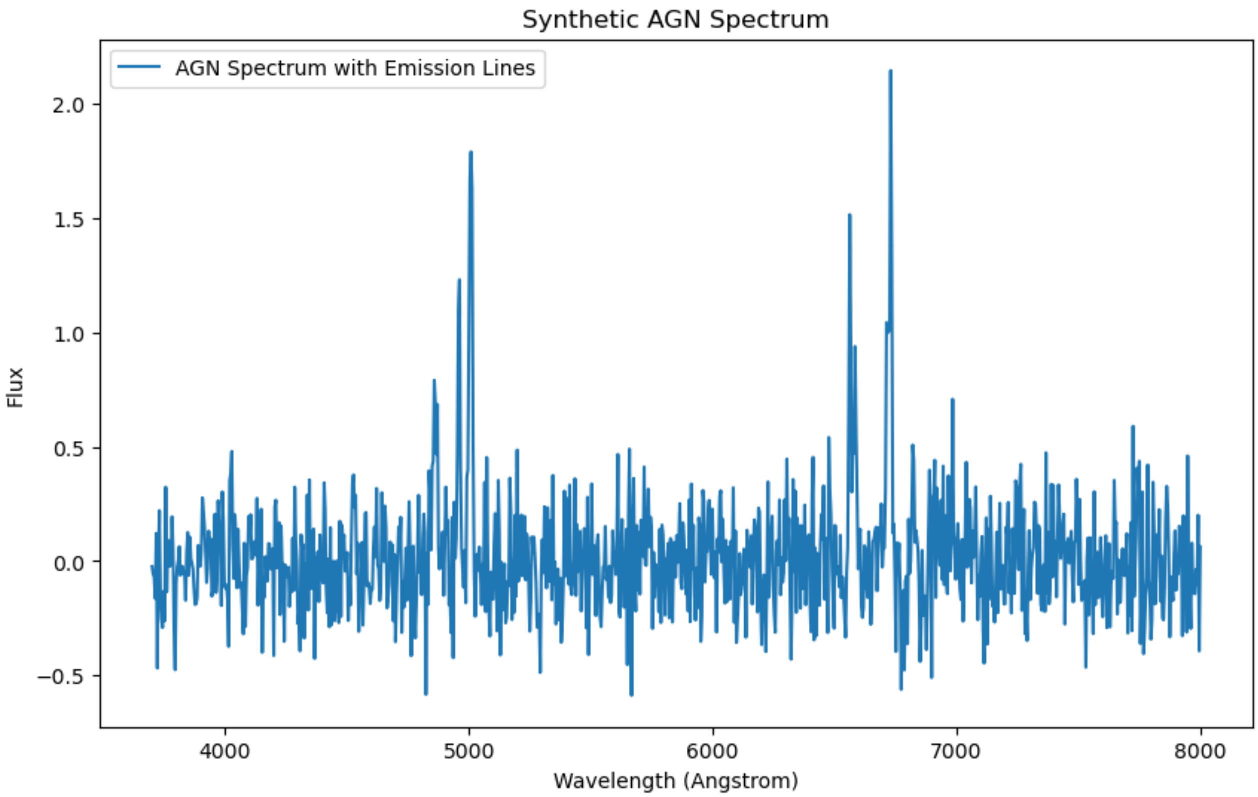


The **width** and **height** of each emission line is usually based on observation, these are determined by a various factor such as pressure, abundance, temperature, density, and ionization state. I here randomize them, and for next week, I want to use the randomized spectra to observe the correlation between these parameters. The emission lines that are usually visible are:

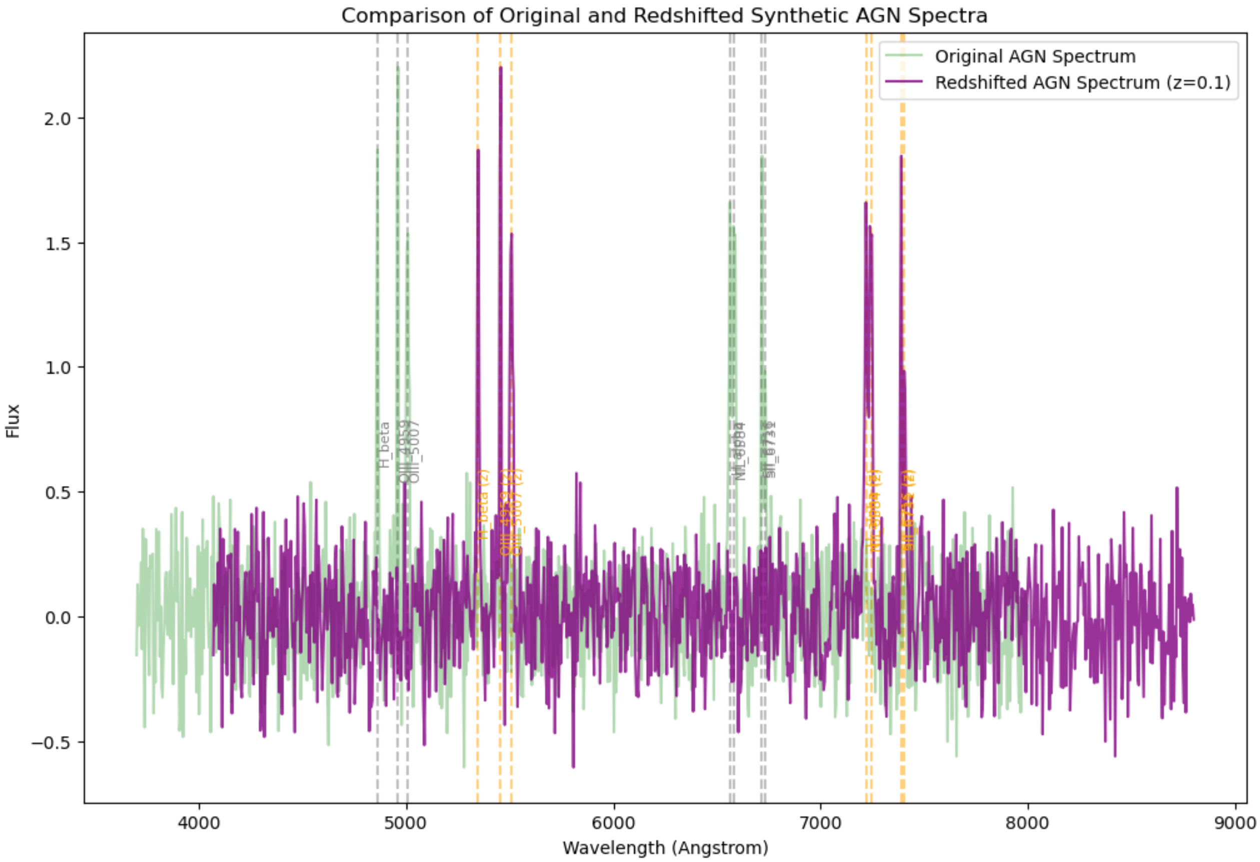
1. H_alpha = 6563 # H-alpha
2. H_beta = 4861 # H-beta
3. OIII_5007 = 5007 # O III line
4. OIII_4959 = 4959 # O III line
5. NII_6584 = 6584 # N II line
6. SII_6716 = 6716 # S II line
7. SII_6731 = 6731 # S II line

among them, for AGN host galaxies, elements like Hydrogen, Oxygen, and Nitrogen are commonly observed with relatively intense lines, so I time their flux by a factor of 1.5

In [27]:
```
# Step 2
# now I want to add in some typical emission lines for an AGN host galaxies
wavelengths = np.linspace(3700, 8000, 1000)
alpha = -15
flux_continuum = wavelengths ** alpha
# the emission lines include:
# Randomization ranges
width_range = (2, 10)  # Width range in Angstroms
base_height_range = (10, 20)  # Base height range for less intense lines

emission_lines = {
    "H_alpha": {"center": 6563, "element": "H"},
    "H_beta": {"center": 4861, "element": "H"},
    "OIII_5007": {"center": 5007, "element": "O"},
    "OIII_4959": {"center": 4959, "element": "O"},
    "NII_6584": {"center": 6584, "element": "N"},
    "SII_6716": {"center": 6716, "element": "S"},
    "SII_6731": {"center": 6731, "element": "S"}
}


def add_emission_line(wavelength_center, width, height):
    return height * norm.pdf(wavelengths, wavelength_center, width)

for line in emission_lines.values():
    width = random.uniform(*width_range)
    height = random.uniform(*base_height_range)

    if line["element"] in ["H", "O", "N"]:
        height *= 1.5

    flux_continuum += add_emission_line(line["center"], width, height)

# Step 3: The spectrum usually also have noise, I also want to include that:
noise_level = 0.2
noise = np.random.normal(0, noise_level, wavelengths.shape)
flux_continuum += noise

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(wavelengths, flux_continuum, label='AGN Spectrum with Emission Line
plt.xlabel('Wavelength (Angstrom)')
plt.ylabel('Flux')
plt.title('Synthetic AGN Spectrum')
plt.legend()
plt.show()
```


Synthetic AGN Spectrum
AGN Spectrum with Emission Lines
Flux
Wavelength (Angstrom)
2.0
1.5
1.0
0.5
0.0
−0.5
4000
5000
6000
7000
8000

In [28]:
```
wavelengths = np.linspace(3700, 8000, 1000)
alpha = -15
flux_continuum = wavelengths ** alpha

emission_lines = {
    "H_alpha": {"center": 6563, "element": "H"},
    "H_beta": {"center": 4861, "element": "H"},
    "OIII_5007": {"center": 5007, "element": "O"},
    "OIII_4959": {"center": 4959, "element": "O"},
    "NII_6584": {"center": 6584, "element": "N"},
    "SII_6716": {"center": 6716, "element": "S"},
    "SII_6731": {"center": 6731, "element": "S"}
}

def add_emission_line(wavelength_center, width, height):
    return height * norm.pdf(wavelengths, wavelength_center, width)

width_range = (2, 10)
base_height_range = (10, 20)

for line in emission_lines.values():
    width = random.uniform(*width_range)
    height = random.uniform(*base_height_range)
    if line["element"] in ["H", "O", "N"]:
        height *= 1.5
    flux_continuum += add_emission_line(line["center"], width, height)

noise_level = 0.2
noise = np.random.normal(0, noise_level, wavelengths.shape)
flux_continuum += noise

z = 0.1
original_flux_continuum = flux_continuum.copy()
redshifted_wavelengths = wavelengths * (1 + z)
redshifted_flux_continuum = original_flux_continuum

plt.figure(figsize=(12, 8))
for line_name, line in emission_lines.items():
    original_center = line["center"]
    redshifted_center = original_center * (1 + z)
    plt.axvline(x=original_center, color='gray', linestyle='--', alpha=0.5)
    plt.axvline(x=redshifted_center, color='orange', linestyle='--', alpha=0
    plt.text(original_center, plt.ylim()[1]*0.8, line_name, rotation=90, ver
    plt.text(redshifted_center, plt.ylim()[1]*0.6, f'{line_name} (z)', rotat

plt.plot(wavelengths, original_flux_continuum, label='Original AGN Spectrum'
plt.plot(redshifted_wavelengths, redshifted_flux_continuum, label=f'Redshift
plt.xlabel('Wavelength (Angstrom)')
plt.ylabel('Flux')
plt.title('Comparison of Original and Redshifted Synthetic AGN Spectra')
plt.legend()
plt.show()
```

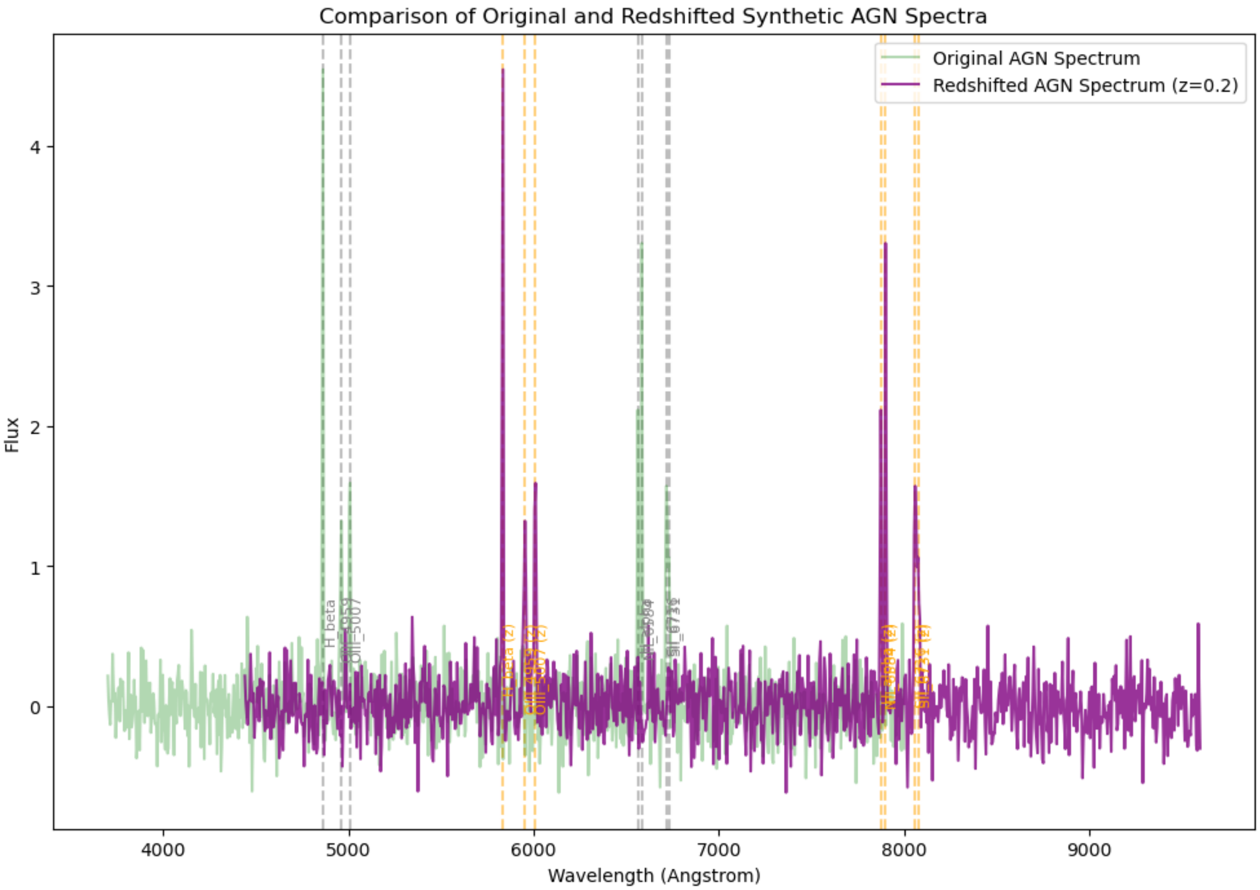


In [29]:
```
# define a function that takes a single input of redshift and produce the sp
def generate_redshifted_spectrum(redshift):
    wavelengths = np.linspace(3700, 8000, 1000)
    alpha = -15
    flux_continuum = wavelengths ** alpha

    emission_lines = {
        "H_alpha": {"center": 6563, "element": "H"},
        "H_beta": {"center": 4861, "element": "H"},
        "OIII_5007": {"center": 5007, "element": "O"},
        "OIII_4959": {"center": 4959, "element": "O"},
        "NII_6584": {"center": 6584, "element": "N"},
        "SII_6716": {"center": 6716, "element": "S"},
        "SII_6731": {"center": 6731, "element": "S"}
    }

    def add_emission_line(wavelength_center, width, height):
        return height * norm.pdf(wavelengths, wavelength_center, width)

    width_range = (2, 10)
    base_height_range = (10, 20)

    for line in emission_lines.values():
        width = random.uniform(*width_range)
        height = random.uniform(*base_height_range)
        if line["element"] in ["H", "O", "N"]:
```

```
            height *= 1.5
        flux_continuum += add_emission_line(line["center"], width, height)

    noise_level = 0.2
    noise = np.random.normal(0, noise_level, wavelengths.shape)
    flux_continuum += noise

    original_flux_continuum = flux_continuum.copy()
    redshifted_wavelengths = wavelengths * (1 + redshift)
    redshifted_flux_continuum = original_flux_continuum

    plt.figure(figsize=(12, 8))
    for line_name, line in emission_lines.items():
        original_center = line["center"]
        redshifted_center = original_center * (1 + redshift)
        plt.axvline(x=original_center, color='gray', linestyle='--', alpha=0
        plt.axvline(x=redshifted_center, color='orange', linestyle='--', alp
        plt.text(original_center, plt.ylim()[1]*0.8, line_name, rotation=90,
        plt.text(redshifted_center, plt.ylim()[1]*0.6, f'{line_name} (z)', r

    plt.plot(wavelengths, original_flux_continuum, label='Original AGN Spect
    plt.plot(redshifted_wavelengths, redshifted_flux_continuum, label=f'Reds
    plt.xlabel('Wavelength (Angstrom)')
    plt.ylabel('Flux')
    plt.title('Comparison of Original and Redshifted Synthetic AGN Spectra')
    plt.legend()
    plt.show()
    return redshifted_wavelengths, flux_continuum
```

In [30]:

```
# example
generate_redshifted_spectrum(0.2)
```



Out[30]:

```
(array([4440.        , 4445.16516517, 4450.33033033, 4455.4954955 ,
        4460.66066066, 4465.82582583, 4470.99099099, 4476.15615616,
        4481.32132132, 4486.48648649, 4491.65165165, 4496.81681682,
        4501.98198198, 4507.14714715, 4512.31231231, 4517.47747748,
        4522.64264264, 4527.80780781, 4532.97297297, 4538.13813814,
        4543.3033033 , 4548.46846847, 4553.63363363, 4558.7987988 ,
        4563.96396396, 4569.12912913, 4574.29429429, 4579.45945946,
        4584.62462462, 4589.78978979, 4594.95495495, 4600.12012012,
        4605.28528529, 4610.45045045, 4615.61561562, 4620.78078078,
        4625.94594595, 4631.11111111, 4636.27627628, 4641.44144144,
        4646.60660661, 4651.77177177, 4656.93693694, 4662.1021021 ,
        4667.26726727, 4672.43243243, 4677.5975976 , 4682.76276276,
        4687.92792793, 4693.09309309, 4698.25825826, 4703.42342342,
        4708.58858859, 4713.75375375, 4718.91891892, 4724.08408408,
        4729.24924925, 4734.41441441, 4739.57957958, 4744.74474474,
        4749.90990991, 4755.07507508, 4760.24024024, 4765.40540541,
        4770.57057057, 4775.73573574, 4780.9009009 , 4786.06606607,
        4791.23123123, 4796.3963964 , 4801.56156156, 4806.72672673,
        4811.89189189, 4817.05705706, 4822.22222222, 4827.38738739,
        4832.55255255, 4837.71771772, 4842.88288288, 4848.04804805,
        4853.21321321, 4858.37837838, 4863.54354354, 4868.70870871,
        4873.87387387, 4879.03903904, 4884.2042042 , 4889.36936937,
        4894.53453453, 4899.6996997 , 4904.86486486, 4910.03003003,
        4915.1951952 , 4920.36036036, 4925.52552553, 4930.69069069,
        4935.85585586, 4941.02102102, 4946.18618619, 4951.35135135,
        4956.51651652, 4961.68168168, 4966.84684685, 4972.01201201,
```

```
4977.17717718, 4982.34234234, 4987.50750751, 4992.67267267,
4997.83783784, 5003.003003  , 5008.16816817, 5013.33333333,
5018.4984985 , 5023.66366366, 5028.82882883, 5033.99399399,
5039.15915916, 5044.32432432, 5049.48948949, 5054.65465465,
5059.81981982, 5064.98498498, 5070.15015015, 5075.31531532,
5080.48048048, 5085.64564565, 5090.81081081, 5095.97597598,
5101.14114114, 5106.30630631, 5111.47147147, 5116.63663664,
5121.8018018 , 5126.96696697, 5132.13213213, 5137.2972973 ,
5142.46246246, 5147.62762763, 5152.79279279, 5157.95795796,
5163.12312312, 5168.28828829, 5173.45345345, 5178.61861862,
5183.78378378, 5188.94894895, 5194.11411411, 5199.27927928,
5204.44444444, 5209.60960961, 5214.77477477, 5219.93993994,
5225.10510511, 5230.27027027, 5235.43543544, 5240.6006006 ,
5245.76576577, 5250.93093093, 5256.0960961 , 5261.26126126,
5266.42642643, 5271.59159159, 5276.75675676, 5281.92192192,
5287.08708709, 5292.25225225, 5297.41741742, 5302.58258258,
5307.74774775, 5312.91291291, 5318.07807808, 5323.24324324,
5328.40840841, 5333.57357357, 5338.73873874, 5343.9039039 ,
5349.06906907, 5354.23423423, 5359.3993994 , 5364.56456456,
5369.72972973, 5374.89489489, 5380.06006006, 5385.22522523,
5390.39039039, 5395.55555556, 5400.72072072, 5405.88588589,
5411.05105105, 5416.21621622, 5421.38138138, 5426.54654655,
5431.71171171, 5436.87687688, 5442.04204204, 5447.20720721,
5452.37237237, 5457.53753754, 5462.7027027 , 5467.86786787,
5473.03303303, 5478.1981982 , 5483.36336336, 5488.52852853,
5493.69369369, 5498.85885886, 5504.02402402, 5509.18918919,
5514.35435435, 5519.51951952, 5524.68468468, 5529.84984985,
5535.01501502, 5540.18018018, 5545.34534535, 5550.51051051,
5555.67567568, 5560.84084084, 5566.00600601, 5571.17117117,
5576.33633634, 5581.5015015 , 5586.66666667, 5591.83183183,
5596.996997  , 5602.16216216, 5607.32732733, 5612.49249249,
5617.65765766, 5622.82282282, 5627.98798799, 5633.15315315,
5638.31831832, 5643.48348348, 5648.64864865, 5653.81381381,
5658.97897898, 5664.14414414, 5669.30930931, 5674.47447447,
5679.63963964, 5684.8048048 , 5689.96996997, 5695.13513514,
5700.3003003 , 5705.46546547, 5710.63063063, 5715.7957958 ,
5720.96096096, 5726.12612613, 5731.29129129, 5736.45645646,
5741.62162162, 5746.78678679, 5751.95195195, 5757.11711712,
5762.28228228, 5767.44744745, 5772.61261261, 5777.77777778,
5782.94294294, 5788.10810811, 5793.27327327, 5798.43843844,
5803.6036036 , 5808.76876877, 5813.93393393, 5819.0990991 ,
5824.26426426, 5829.42942943, 5834.59459459, 5839.75975976,
5844.92492492, 5850.09009009, 5855.25525526, 5860.42042042,
5865.58558559, 5870.75075075, 5875.91591592, 5881.08108108,
5886.24624625, 5891.41141141, 5896.57657658, 5901.74174174,
5906.90690691, 5912.07207207, 5917.23723724, 5922.4024024 ,
5927.56756757, 5932.73273273, 5937.8978979 , 5943.06306306,
5948.22822823, 5953.39339339, 5958.55855856, 5963.72372372,
5968.88888889, 5974.05405405, 5979.21921922, 5984.38438438,
5989.54954955, 5994.71471471, 5999.87987988, 6005.04504505,
6010.21021021, 6015.37537538, 6020.54054054, 6025.70570571,
6030.87087087, 6036.03603604, 6041.2012012 , 6046.36636637,
6051.53153153, 6056.6966967 , 6061.86186186, 6067.02702703,
```

```
6072.19219219, 6077.35735736, 6082.52252252, 6087.68768769,
6092.85285285, 6098.01801802, 6103.18318318, 6108.34834835,
6113.51351351, 6118.67867868, 6123.84384384, 6129.00900901,
6134.17417417, 6139.33933934, 6144.5045045 , 6149.66966967,
6154.83483483, 6160.        , 6165.16516517, 6170.33033033,
6175.4954955 , 6180.66066066, 6185.82582583, 6190.99099099,
6196.15615616, 6201.32132132, 6206.48648649, 6211.65165165,
6216.81681682, 6221.98198198, 6227.14714715, 6232.31231231,
6237.47747748, 6242.64264264, 6247.80780781, 6252.97297297,
6258.13813814, 6263.3033033 , 6268.46846847, 6273.63363363,
6278.7987988 , 6283.96396396, 6289.12912913, 6294.29429429,
6299.45945946, 6304.62462462, 6309.78978979, 6314.95495495,
6320.12012012, 6325.28528529, 6330.45045045, 6335.61561562,
6340.78078078, 6345.94594595, 6351.11111111, 6356.27627628,
6361.44144144, 6366.60660661, 6371.77177177, 6376.93693694,
6382.1021021 , 6387.26726727, 6392.43243243, 6397.5975976 ,
6402.76276276, 6407.92792793, 6413.09309309, 6418.25825826,
6423.42342342, 6428.58858859, 6433.75375375, 6438.91891892,
6444.08408408, 6449.24924925, 6454.41441441, 6459.57957958,
6464.74474474, 6469.90990991, 6475.07507508, 6480.24024024,
6485.40540541, 6490.57057057, 6495.73573574, 6500.9009009 ,
6506.06606607, 6511.23123123, 6516.3963964 , 6521.56156156,
6526.72672673, 6531.89189189, 6537.05705706, 6542.22222222,
6547.38738739, 6552.55255255, 6557.71771772, 6562.88288288,
6568.04804805, 6573.21321321, 6578.37837838, 6583.54354354,
6588.70870871, 6593.87387387, 6599.03903904, 6604.2042042 ,
6609.36936937, 6614.53453453, 6619.6996997 , 6624.86486486,
6630.03003003, 6635.1951952 , 6640.36036036, 6645.52552553,
6650.69069069, 6655.85585586, 6661.02102102, 6666.18618619,
6671.35135135, 6676.51651652, 6681.68168168, 6686.84684685,
6692.01201201, 6697.17717718, 6702.34234234, 6707.50750751,
6712.67267267, 6717.83783784, 6723.003003  , 6728.16816817,
6733.33333333, 6738.4984985 , 6743.66366366, 6748.82882883,
6753.99399399, 6759.15915916, 6764.32432432, 6769.48948949,
6774.65465465, 6779.81981982, 6784.98498498, 6790.15015015,
6795.31531532, 6800.48048048, 6805.64564565, 6810.81081081,
6815.97597598, 6821.14114114, 6826.30630631, 6831.47147147,
6836.63663664, 6841.8018018 , 6846.96696697, 6852.13213213,
6857.2972973 , 6862.46246246, 6867.62762763, 6872.79279279,
6877.95795796, 6883.12312312, 6888.28828829, 6893.45345345,
6898.61861862, 6903.78378378, 6908.94894895, 6914.11411411,
6919.27927928, 6924.44444444, 6929.60960961, 6934.77477477,
6939.93993994, 6945.10510511, 6950.27027027, 6955.43543544,
6960.6006006 , 6965.76576577, 6970.93093093, 6976.0960961 ,
6981.26126126, 6986.42642643, 6991.59159159, 6996.75675676,
7001.92192192, 7007.08708709, 7012.25225225, 7017.41741742,
7022.58258258, 7027.74774775, 7032.91291291, 7038.07807808,
7043.24324324, 7048.40840841, 7053.57357357, 7058.73873874,
7063.9039039 , 7069.06906907, 7074.23423423, 7079.3993994 ,
7084.56456456, 7089.72972973, 7094.89489489, 7100.06006006,
7105.22522523, 7110.39039039, 7115.55555556, 7120.72072072,
7125.88588589, 7131.05105105, 7136.21621622, 7141.38138138,
7146.54654655, 7151.71171171, 7156.87687688, 7162.04204204,
```

```
7167.20720721, 7172.37237237, 7177.53753754, 7182.7027027 ,
7187.86786787, 7193.03303303, 7198.1981982 , 7203.36336336,
7208.52852853, 7213.69369369, 7218.85885886, 7224.02402402,
7229.18918919, 7234.35435435, 7239.51951952, 7244.68468468,
7249.84984985, 7255.01501502, 7260.18018018, 7265.34534535,
7270.51051051, 7275.67567568, 7280.84084084, 7286.00600601,
7291.17117117, 7296.33633634, 7301.5015015 , 7306.66666667,
7311.83183183, 7316.996997  , 7322.16216216, 7327.32732733,
7332.49249249, 7337.65765766, 7342.82282282, 7347.98798799,
7353.15315315, 7358.31831832, 7363.48348348, 7368.64864865,
7373.81381381, 7378.97897898, 7384.14414414, 7389.30930931,
7394.47447447, 7399.63963964, 7404.8048048 , 7409.96996997,
7415.13513514, 7420.3003003 , 7425.46546547, 7430.63063063,
7435.7957958 , 7440.96096096, 7446.12612613, 7451.29129129,
7456.45645646, 7461.62162162, 7466.78678679, 7471.95195195,
7477.11711712, 7482.28228228, 7487.44744745, 7492.61261261,
7497.77777778, 7502.94294294, 7508.10810811, 7513.27327327,
7518.43843844, 7523.6036036 , 7528.76876877, 7533.93393393,
7539.0990991 , 7544.26426426, 7549.42942943, 7554.59459459,
7559.75975976, 7564.92492492, 7570.09009009, 7575.25525526,
7580.42042042, 7585.58558559, 7590.75075075, 7595.91591592,
7601.08108108, 7606.24624625, 7611.41141141, 7616.57657658,
7621.74174174, 7626.90690691, 7632.07207207, 7637.23723724,
7642.4024024 , 7647.56756757, 7652.73273273, 7657.8978979 ,
7663.06306306, 7668.22822823, 7673.39339339, 7678.55855856,
7683.72372372, 7688.88888889, 7694.05405405, 7699.21921922,
7704.38438438, 7709.54954955, 7714.71471471, 7719.87987988,
7725.04504505, 7730.21021021, 7735.37537538, 7740.54054054,
7745.70570571, 7750.87087087, 7756.03603604, 7761.2012012 ,
7766.36636637, 7771.53153153, 7776.6966967 , 7781.86186186,
7787.02702703, 7792.19219219, 7797.35735736, 7802.52252252,
7807.68768769, 7812.85285285, 7818.01801802, 7823.18318318,
7828.34834835, 7833.51351351, 7838.67867868, 7843.84384384,
7849.00900901, 7854.17417417, 7859.33933934, 7864.5045045 ,
7869.66966967, 7874.83483483, 7880.        , 7885.16516517,
7890.33033033, 7895.4954955 , 7900.66066066, 7905.82582583,
7910.99099099, 7916.15615616, 7921.32132132, 7926.48648649,
7931.65165165, 7936.81681682, 7941.98198198, 7947.14714715,
7952.31231231, 7957.47747748, 7962.64264264, 7967.80780781,
7972.97297297, 7978.13813814, 7983.3033033 , 7988.46846847,
7993.63363363, 7998.7987988 , 8003.96396396, 8009.12912913,
8014.29429429, 8019.45945946, 8024.62462462, 8029.78978979,
8034.95495495, 8040.12012012, 8045.28528529, 8050.45045045,
8055.61561562, 8060.78078078, 8065.94594595, 8071.11111111,
8076.27627628, 8081.44144144, 8086.60660661, 8091.77177177,
8096.93693694, 8102.1021021 , 8107.26726727, 8112.43243243,
8117.5975976 , 8122.76276276, 8127.92792793, 8133.09309309,
8138.25825826, 8143.42342342, 8148.58858859, 8153.75375375,
8158.91891892, 8164.08408408, 8169.24924925, 8174.41441441,
8179.57957958, 8184.74474474, 8189.90990991, 8195.07507508,
8200.24024024, 8205.40540541, 8210.57057057, 8215.73573574,
8220.9009009 , 8226.06606607, 8231.23123123, 8236.3963964 ,
8241.56156156, 8246.72672673, 8251.89189189, 8257.05705706,
```

```
8262.22222222, 8267.38738739, 8272.55255255, 8277.71771772,
8282.88288288, 8288.04804805, 8293.21321321, 8298.37837838,
8303.54354354, 8308.70870871, 8313.87387387, 8319.03903904,
8324.2042042 , 8329.36936937, 8334.53453453, 8339.6996997 ,
8344.86486486, 8350.03003003, 8355.1951952 , 8360.36036036,
8365.52552553, 8370.69069069, 8375.85585586, 8381.02102102,
8386.18618619, 8391.35135135, 8396.51651652, 8401.68168168,
8406.84684685, 8412.01201201, 8417.17717718, 8422.34234234,
8427.50750751, 8432.67267267, 8437.83783784, 8443.003003  ,
8448.16816817, 8453.33333333, 8458.4984985 , 8463.66366366,
8468.82882883, 8473.99399399, 8479.15915916, 8484.32432432,
8489.48948949, 8494.65465465, 8499.81981982, 8504.98498498,
8510.15015015, 8515.31531532, 8520.48048048, 8525.64564565,
8530.81081081, 8535.97597598, 8541.14114114, 8546.30630631,
8551.47147147, 8556.63663664, 8561.8018018 , 8566.96696697,
8572.13213213, 8577.2972973 , 8582.46246246, 8587.62762763,
8592.79279279, 8597.95795796, 8603.12312312, 8608.28828829,
8613.45345345, 8618.61861862, 8623.78378378, 8628.94894895,
8634.11411411, 8639.27927928, 8644.44444444, 8649.60960961,
8654.77477477, 8659.93993994, 8665.10510511, 8670.27027027,
8675.43543544, 8680.6006006 , 8685.76576577, 8690.93093093,
8696.0960961 , 8701.26126126, 8706.42642643, 8711.59159159,
8716.75675676, 8721.92192192, 8727.08708709, 8732.25225225,
8737.41741742, 8742.58258258, 8747.74774775, 8752.91291291,
8758.07807808, 8763.24324324, 8768.40840841, 8773.57357357,
8778.73873874, 8783.9039039 , 8789.06906907, 8794.23423423,
8799.3993994 , 8804.56456456, 8809.72972973, 8814.89489489,
8820.06006006, 8825.22522523, 8830.39039039, 8835.55555556,
8840.72072072, 8845.88588589, 8851.05105105, 8856.21621622,
8861.38138138, 8866.54654655, 8871.71171171, 8876.87687688,
8882.04204204, 8887.20720721, 8892.37237237, 8897.53753754,
8902.7027027 , 8907.86786787, 8913.03303303, 8918.1981982 ,
8923.36336336, 8928.52852853, 8933.69369369, 8938.85885886,
8944.02402402, 8949.18918919, 8954.35435435, 8959.51951952,
8964.68468468, 8969.84984985, 8975.01501502, 8980.18018018,
8985.34534535, 8990.51051051, 8995.67567568, 9000.84084084,
9006.00600601, 9011.17117117, 9016.33633634, 9021.5015015 ,
9026.66666667, 9031.83183183, 9036.996997  , 9042.16216216,
9047.32732733, 9052.49249249, 9057.65765766, 9062.82282282,
9067.98798799, 9073.15315315, 9078.31831832, 9083.48348348,
9088.64864865, 9093.81381381, 9098.97897898, 9104.14414414,
9109.30930931, 9114.47447447, 9119.63963964, 9124.8048048 ,
9129.96996997, 9135.13513514, 9140.3003003 , 9145.46546547,
9150.63063063, 9155.7957958 , 9160.96096096, 9166.12612613,
9171.29129129, 9176.45645646, 9181.62162162, 9186.78678679,
9191.95195195, 9197.11711712, 9202.28228228, 9207.44744745,
9212.61261261, 9217.77777778, 9222.94294294, 9228.10810811,
9233.27327327, 9238.43843844, 9243.6036036 , 9248.76876877,
9253.93393393, 9259.0990991 , 9264.26426426, 9269.42942943,
9274.59459459, 9279.75975976, 9284.92492492, 9290.09009009,
9295.25525526, 9300.42042042, 9305.58558559, 9310.75075075,
9315.91591592, 9321.08108108, 9326.24624625, 9331.41141141,
9336.57657658, 9341.74174174, 9346.90690691, 9352.07207207,
```

```
        9357.23723724, 9362.4024024 , 9367.56756757, 9372.73273273,
        9377.8978979 , 9383.06306306, 9388.22822823, 9393.39339339,
        9398.55855856, 9403.72372372, 9408.88888889, 9414.05405405,
        9419.21921922, 9424.38438438, 9429.54954955, 9434.71471471,
        9439.87987988, 9445.04504505, 9450.21021021, 9455.37537538,
        9460.54054054, 9465.70570571, 9470.87087087, 9476.03603604,
        9481.2012012 , 9486.36636637, 9491.53153153, 9496.6966967 ,
        9501.86186186, 9507.02702703, 9512.19219219, 9517.35735736,
        9522.52252252, 9527.68768769, 9532.85285285, 9538.01801802,
        9543.18318318, 9548.34834835, 9553.51351351, 9558.67867868,
        9563.84384384, 9569.00900901, 9574.17417417, 9579.33933934,
        9584.5045045 , 9589.66966967, 9594.83483483, 9600.        ]),
array([ 2.17936350e-01,  3.94994044e-02, -5.67191964e-02, -1.30187083e-01,
        3.73651013e-02,  4.23066767e-02,  3.74427025e-01,  1.57046040e-01,
        3.20938839e-02,  9.42710802e-02, -2.23440174e-01, -7.46713323e-02,
       -7.09310961e-02,  1.29166151e-01,  1.36650175e-01, -1.05659149e-01,
        1.96383252e-01,  7.61453701e-02,  1.19899060e-01,  1.77458231e-01,
       -5.12846253e-02, -1.74575202e-01, -2.87430756e-02, -1.53914059e-01,
       -1.43105663e-01,  1.13642390e-02,  3.84524077e-01, -2.32292524e-01,
        1.29103871e-02,  8.16498967e-02, -1.48287626e-01, -1.02287261e-01,
       -2.89693726e-02,  1.88548483e-01,  4.87910412e-03,  6.87618496e-02,
       -3.69300067e-01, -2.78449627e-01, -7.70771312e-02, -3.18552911e-02,
       -3.12493372e-01,  8.26262961e-03,  4.18512082e-01, -3.47870688e-02,
        4.01378092e-01,  6.64803220e-02, -1.16848243e-01,  1.79610799e-01,
        3.26573774e-01, -7.43681337e-03, -6.90276693e-02,  1.64795114e-01,
        5.56319127e-02,  1.68239180e-01, -1.87278844e-01, -9.09798756e-02,
       -7.98676929e-02,  9.24623067e-02,  3.72904102e-02, -1.11920677e-01,
       -1.30331042e-01, -1.21002273e-01, -4.26755738e-01, -6.21241276e-02,
       -5.27116775e-02, -7.27832958e-02,  3.32238834e-01, -2.54808225e-01,
        1.67354991e-02,  2.24637612e-01, -1.36787771e-01, -1.86025576e-01,
       -3.08082029e-01,  7.95792250e-03, -1.93251921e-01, -3.35235688e-01,
       -2.95748689e-01,  5.25781255e-02,  1.99445608e-01, -5.97454472e-02,
        1.46193999e-01,  1.72217181e-01, -9.86244467e-02, -3.46568073e-01,
        1.17646207e-01, -4.74531596e-02, -1.21853439e-01,  1.86485378e-01,
        2.32225147e-01,  2.94389048e-01,  8.45729648e-02, -6.72883874e-02,
        1.00344580e-01,  1.82126715e-01,  1.30087818e-01,  7.38569677e-02,
       -4.06061163e-02, -1.60174387e-01,  1.15098536e-01,  1.00017788e-01,
       -2.09664349e-01,  3.18791707e-03, -4.28645379e-01, -6.39821218e-02,
        5.01096607e-02,  5.45265486e-01,  3.07507820e-01, -1.17766276e-02,
        3.85197558e-02, -7.62096106e-02,  1.59463980e-01, -1.57853375e-01,
       -3.09131780e-02, -2.40602908e-02, -1.00070726e-02, -1.16689468e-01,
        2.45123795e-01, -2.92285672e-01,  2.41217273e-01, -9.60099879e-02,
        5.82042932e-02, -3.75579634e-01,  2.90545311e-01,  3.64524798e-02,
        5.15483487e-02,  4.54295515e-02, -6.18047285e-02, -3.25836597e-02,
       -2.70035762e-01,  7.38730441e-02,  9.95587217e-02,  3.54645533e-02,
        2.31106989e-01, -4.60642559e-02,  2.13034900e-02, -7.35207261e-02,
        1.18764806e-01, -3.39641554e-01,  1.15992980e-01,  1.84315751e-01,
       -2.02723332e-01,  2.23360658e-01, -4.62859491e-01, -2.87481071e-01,
        1.18761142e-01, -1.26641040e-02,  6.37521744e-02,  2.22283881e-01,
       -7.03365296e-02,  1.10559309e-02, -2.25593541e-01,  3.85987581e-02,
        8.67976992e-02, -2.15532396e-01, -1.31355790e-01, -8.01164210e-02,
       -1.86937450e-01,  1.04183399e-02,  1.53116916e-01, -9.71393702e-02,
       -1.53456627e-01, -1.61569640e-01,  2.27717619e-01,  2.14205584e-01,
```

```
-2.27915155e-01,  4.30481486e-02,  4.99627459e-03, -7.78345118e-02,
 3.05674252e-01, -1.53090639e-01, -6.84769642e-02, -1.18413604e-01,
 2.62988262e-01,  2.06244552e-01, -7.62247480e-02,  6.38363896e-01,
 3.09731592e-01,  1.90770343e-01, -2.59545615e-01,  3.22719612e-02,
-1.36834351e-03, -6.06037366e-01, -1.59841282e-01,  1.51460442e-01,
 1.76606079e-01,  3.16307071e-02,  2.71628261e-01,  1.90979429e-01,
 4.29956130e-01,  2.73179049e-02,  1.68302210e-01, -2.99870184e-02,
-7.32954213e-02,  3.04558048e-01,  7.91408435e-02, -1.24575289e-01,
-3.69594596e-01, -3.92113526e-01, -6.89197931e-02,  1.26718934e-01,
 1.19873970e-02, -6.33691210e-02,  5.30953915e-02,  2.24293529e-01,
-1.02550207e-01, -1.42786092e-01, -7.90399841e-02, -2.30741913e-01,
 1.49365387e-01,  1.32712153e-01, -1.25500067e-02, -7.51688096e-02,
-4.97383872e-01,  6.98637509e-02, -5.99456587e-02,  3.18645017e-01,
 1.29113303e-01, -9.65280948e-02, -7.66983820e-02,  1.75067629e-01,
-1.59807930e-01, -3.86905408e-02,  3.13443096e-01, -3.76614792e-02,
 2.30574266e-02, -1.22101095e-01, -1.22044096e-01, -3.67982472e-02,
 3.55296274e-02, -3.47610386e-02, -4.45791571e-02,  4.51397688e-01,
 3.75646162e-02, -5.68307861e-02, -2.87004626e-02, -2.46003679e-02,
 1.58421105e-01,  1.56945912e-01, -1.66657568e-02,  1.38384573e-01,
 4.93234886e-01,  5.04338764e-02,  3.29373920e-02,  4.21082578e-01,
 6.33073317e-02,  9.71739495e-02, -2.13429857e-01, -9.68816118e-02,
-2.04753568e-02,  2.34355533e-02, -4.01421459e-02, -6.88224227e-02,
-3.29223332e-01,  1.61017929e-01, -9.76630978e-03,  1.29046942e-01,
 3.77730518e-02, -1.13134569e-01, -2.50848405e-01, -2.69882374e-01,
 2.17134189e-01,  2.84136252e-01, -6.14477718e-02,  4.78209794e-01,
 3.50182433e-02, -2.05414638e-01, -1.10991922e-01,  2.85253642e-01,
 3.35007730e-01,  2.40597339e+00,  4.54528833e+00,  2.81377693e-01,
-2.39777683e-01, -2.44040819e-01, -3.23180564e-02, -9.77253128e-02,
-3.49510807e-02, -9.79555822e-02,  8.11023093e-02,  1.59902449e-01,
 2.30167921e-01,  3.25181770e-01, -1.31581337e-01, -1.97866518e-01,
-4.32704016e-02, -1.03836736e-01, -2.39819415e-01, -8.25573338e-02,
 1.29228429e-01, -7.57755590e-02,  3.04095290e-01,  6.06460081e-01,
 8.69274344e-01,  1.32332408e+00,  8.91843389e-01,  1.87901037e-01,
 5.25630770e-02,  1.98593976e-01,  3.70348742e-01, -3.97487363e-02,
 2.68120536e-01,  3.15423611e-01,  4.12572886e-01,  1.39897265e+00,
 1.59406795e+00,  7.38565198e-01, -1.15847819e-01, -8.40707380e-02,
 2.71453915e-01, -1.42639967e-01,  6.69878377e-02, -2.01529239e-01,
 1.98050565e-01,  5.40613637e-03,  2.16758499e-01, -1.48158253e-01,
 3.33770109e-01,  2.78330919e-01,  9.73493205e-03,  2.02546245e-01,
-9.60106571e-03,  8.09697275e-02,  1.49284875e-01,  3.18515734e-01,
 5.13067199e-02,  2.58404591e-01,  7.67693771e-02, -9.72627656e-02,
-1.65226307e-01, -5.20482224e-02, -2.07165569e-01,  1.56470365e-01,
 4.35516423e-02,  2.32355006e-01,  3.60281423e-02,  2.26573549e-01,
-1.41287421e-01, -2.28105763e-01,  2.09479448e-01,  1.28116911e-01,
-1.39057789e-01, -4.20195459e-01, -1.25787180e-01,  7.56595846e-02,
 3.71187990e-01,  6.05223661e-02,  7.60373725e-03,  3.95941574e-01,
 9.83987549e-02, -7.25956924e-02, -1.57206821e-01,  4.30804438e-03,
 7.58655823e-02, -3.09179283e-01, -3.21005734e-02,  2.33365562e-02,
 3.18829339e-01, -6.96263671e-02, -2.37500084e-01, -3.96938860e-02,
-1.70716072e-01, -1.90608632e-01,  5.24410211e-01,  2.64262876e-01,
-1.62597087e-01,  5.24119546e-02,  1.05950207e-02,  8.83986002e-02,
-1.32621824e-01,  2.32504358e-01, -2.08429798e-01,  5.94198319e-02,
-1.83152406e-01,  2.16199926e-01,  1.46011364e-01, -2.57904972e-01,
```

```
-2.69069909e-01,  9.69860522e-02, -1.45836914e-01,  1.87750126e-01,
-9.12978539e-02,  1.06844434e-01,  1.49466821e-01,  3.49477200e-01,
-9.32149925e-03, -2.42197847e-02, -1.04528311e-01,  1.75870074e-01,
 4.71691922e-02, -1.37577014e-01,  1.57510296e-01,  1.33604304e-01,
-2.83530084e-02,  2.91343285e-01,  1.90178313e-01, -1.01781882e-01,
-1.15056749e-01,  1.47040651e-01,  2.69727675e-01, -7.07276413e-02,
 3.96135504e-01, -2.44104274e-02, -3.49487106e-01,  1.22593471e-01,
-7.79824987e-02, -2.42620642e-01, -8.08421003e-02,  1.31787397e-01,
 2.82758652e-01, -4.64212633e-03, -1.01186240e-01,  3.51892430e-01,
-3.00280096e-01,  1.75322554e-01, -3.61643968e-02, -1.21365018e-01,
-2.53016626e-01, -2.04019374e-01, -1.67317444e-01, -1.54598983e-01,
-9.99855605e-02, -9.22121906e-02,  5.74893338e-01, -9.53833015e-03,
 3.77006365e-01,  1.67113075e-01, -4.36736362e-02, -1.09287103e-01,
 3.82175856e-01, -2.55434989e-01, -1.18206729e-01,  1.10044407e-01,
 1.26220527e-01, -4.84834691e-02, -4.05263971e-02, -9.28839854e-02,
-7.49957436e-02,  3.60049430e-02, -1.40236099e-01, -1.33936403e-01,
-2.08686391e-01, -1.66171073e-02,  1.75670727e-01,  2.77890182e-01,
 1.56250656e-01, -4.13129220e-02,  2.54830949e-01,  1.66439352e-01,
-2.12984198e-01, -1.70395650e-01,  3.33857939e-01,  1.71108937e-01,
 3.81411307e-02,  1.67156152e-02,  4.89882476e-02,  1.26606502e-01,
 3.04199929e-02,  1.00056611e-01,  2.19036949e-01,  3.84502963e-02,
 4.24499209e-01,  2.23376951e-01,  2.09390172e-01, -1.15789347e-01,
 1.13584469e-01,  2.53310821e-01, -2.55871450e-02, -4.31603787e-01,
 5.89601657e-02,  1.78700825e-01, -3.67230315e-01, -1.31447707e-01,
 9.12703808e-02,  1.50806231e-02, -2.70532531e-01,  9.44781684e-02,
 1.65216857e-01,  3.79538427e-01,  1.17560530e-01, -8.65413356e-02,
 2.05448141e-01,  6.06134396e-02, -1.73709421e-01, -3.68332238e-02,
 1.90027558e-01,  2.71480589e-01,  7.34748879e-02,  7.90365332e-02,
 1.02233854e-01,  4.87262134e-01, -4.11722961e-01,  6.20850569e-02,
 3.53186742e-01, -2.88207910e-01,  2.64764480e-01,  2.21490692e-02,
-3.77172206e-01,  4.54168873e-02, -1.94874931e-01, -1.76201267e-01,
-1.02709991e-02,  1.36675903e-01, -1.03894210e-01, -2.23896727e-02,
-4.08029472e-01,  1.73201080e-02,  2.75263797e-01,  1.49264338e-01,
 1.14466033e-01,  1.80010649e-01, -8.24374232e-02,  2.75430860e-02,
 2.28285768e-03,  2.36224450e-01, -1.49670779e-01, -1.23288506e-01,
-1.11483333e-01,  6.75059976e-04,  1.98053856e-02,  4.03148897e-01,
-4.76617550e-02,  4.31720864e-01, -1.26476416e-02, -1.80888539e-01,
-3.48850468e-01, -9.84693538e-02, -2.39886131e-02,  7.30581123e-02,
 3.98191003e-01, -4.63405103e-01, -9.47872081e-02, -7.02795419e-02,
-3.49867885e-02, -3.18039297e-01, -1.13371706e-02, -1.29651639e-01,
-1.17520659e-01,  1.95094775e-01,  1.80589957e-02,  1.72029699e-01,
-2.41200950e-01, -1.37371908e-01, -4.06901109e-01, -2.11842222e-01,
 1.78608981e-01,  8.80614641e-02, -6.41482123e-02, -5.78918063e-02,
 1.38311587e-01,  1.57524514e-02,  1.46847977e-01,  1.91651144e-01,
-1.91732150e-01, -4.48797082e-02,  1.30289214e-02,  1.59043635e-01,
-5.65465478e-02, -2.73778250e-01, -9.94046809e-02,  1.91708767e-01,
 1.90403928e-01,  7.00038768e-02,  2.15090745e-01,  2.49668285e-01,
 1.63861505e-02,  7.99443717e-02, -6.14873028e-01,  2.40968826e-01,
-7.75078483e-02,  8.60896732e-02, -2.04142601e-01, -4.29299225e-01,
 1.00026048e-01, -3.05290997e-02,  1.00122517e-03, -2.74034523e-01,
 8.75875588e-02, -1.77723555e-02,  2.47467342e-01,  1.43587113e-01,
 3.64869252e-01, -2.03404493e-01, -5.77763478e-02,  2.02571724e-01,
 1.35576148e-01, -1.31547734e-02,  1.23111751e-01,  1.01325197e-01,
```

```
-6.76791430e-02,  3.62059660e-01, -1.53391046e-01, -2.83906185e-01,
 1.64641897e-01,  1.68254457e-01, -1.51182578e-02,  1.74112078e-01,
 9.14906375e-03, -2.05345179e-01, -7.38722152e-02,  4.13333519e-02,
-4.70022302e-02,  3.31869526e-01,  4.63975031e-01, -4.90894794e-01,
-1.69186351e-01,  1.50994780e-02,  3.48193933e-01,  2.52588156e-01,
 2.10625158e-02, -1.70492858e-01, -2.44998758e-01, -1.68504986e-01,
 3.76273875e-01, -2.26291582e-02,  1.99772816e-02, -3.67994613e-01,
-1.77580467e-01,  1.67315912e-01,  2.02528079e-01, -1.84125353e-01,
 3.41027744e-03,  1.21641848e-01,  3.65360274e-02,  5.17029088e-03,
 1.55798637e-01,  2.73246704e-01, -8.66965887e-02, -1.72764805e-01,
 7.91661759e-02, -1.46592431e-01,  3.47169941e-02,  3.05541067e-01,
-6.21319479e-02, -2.43346578e-01, -1.54568711e-01, -2.47021538e-02,
 6.04911508e-02,  3.69111236e-01, -2.92288720e-01,  1.11619302e-01,
 4.76446261e-01, -1.10294442e-01,  2.03597867e-02, -2.28511601e-01,
-2.13246836e-01, -2.59674644e-01,  9.13995489e-02, -2.79592188e-01,
-1.60194317e-01,  1.34109408e-01,  3.48406293e-01,  2.12967522e-01,
-1.96004975e-01,  6.87619041e-02, -2.14637933e-01,  2.55530020e-01,
 6.30292568e-02, -7.65720287e-02,  2.51669435e-01,  1.35182868e-01,
 1.75276156e-01,  5.40250679e-02, -1.15555455e-01, -1.94592001e-01,
 8.24424208e-01,  2.11532913e+00,  1.16787956e+00, -1.13480642e-01,
 1.95369491e-02,  9.16966161e-01,  3.30588493e+00,  1.25763667e+00,
 7.53031263e-02, -2.91038313e-02, -2.73828905e-02, -4.94182915e-02,
 1.58547546e-02,  1.02776554e-02,  2.60021141e-02, -4.27710758e-03,
 4.02463978e-02, -4.10101130e-01,  2.69993190e-01,  3.30137159e-01,
-2.36939348e-01,  5.64391094e-02,  1.23793225e-01, -1.04059285e-01,
 3.95938407e-02,  1.76201985e-02,  1.96018149e-01,  2.26008204e-01,
-1.61483576e-01, -5.77500743e-01, -1.30863609e-01,  3.32867228e-01,
-1.05319641e-02, -1.95942820e-01, -1.81485150e-01,  2.51774057e-01,
 1.21952124e+00,  1.57254052e+00,  1.37117619e+00,  9.94175126e-01,
 1.06289657e+00,  7.36300054e-01,  6.16909626e-01, -1.49849301e-01,
 2.48227530e-01, -2.04200891e-02,  1.12860418e-01, -3.64633508e-02,
 6.15341264e-02, -1.15550245e-01,  1.77563610e-01, -3.30275771e-01,
 3.00963209e-01,  1.64579992e-02,  8.72072532e-02, -5.27943421e-01,
 1.00258731e-01, -8.26731031e-02,  2.32328341e-04, -6.57523467e-02,
-2.02131740e-03, -1.10278322e-01, -1.14352373e-03,  1.55154479e-01,
-9.75946451e-03,  8.46109611e-02,  1.25271155e-01,  2.56525656e-01,
 6.23779734e-02,  1.42279916e-01, -2.34649104e-02, -7.41829568e-02,
 2.98739203e-01, -3.57278262e-02, -2.38486883e-01, -8.10319581e-02,
-2.09036804e-01,  1.94801084e-01, -1.67381902e-01,  4.40053726e-02,
 1.38095426e-01,  1.15783986e-01, -3.86664286e-01, -1.25496008e-01,
 1.33470987e-01,  3.69569011e-02,  2.06720105e-01,  1.98734590e-01,
 1.70460210e-01, -2.50814148e-01,  1.93462949e-01, -8.36076827e-03,
-8.53990179e-02, -2.91598874e-01, -1.79787919e-01, -2.52030957e-01,
-1.30209217e-01,  1.50455081e-03, -2.30189168e-01,  1.95369039e-01,
-4.44338764e-01,  1.88291366e-02,  2.81222909e-01,  6.91391845e-02,
 1.37444777e-01, -2.18324148e-02, -4.76003678e-02, -4.11037913e-02,
 4.78786922e-02,  2.62844511e-01, -1.18828212e-01, -3.41750399e-01,
 1.45612929e-01,  5.75128597e-01,  4.50407921e-02, -1.93141420e-01,
 1.76400012e-01, -2.69952775e-01, -2.53821920e-01, -1.83662737e-01,
-1.95612829e-01, -4.39516079e-02, -9.99838647e-02, -1.10626567e-01,
 4.26766155e-02,  1.17976096e-01,  1.68320546e-01, -3.85938625e-02,
 2.38283100e-01, -1.24923745e-01,  6.41495923e-02,  3.69739862e-01,
-3.17075020e-02,  2.50074366e-02,  9.72335783e-03,  1.54447324e-01,
```

```
        2.12934329e-01,  1.03948726e-01, -1.73343706e-01, -5.37401543e-02,
        1.72128491e-01,  2.96468326e-01, -1.90294600e-02, -1.88028582e-01,
       -3.02961623e-01,  5.17590970e-02,  1.27681197e-01, -9.43564036e-02,
       -8.96284168e-02,  4.72994660e-03,  1.72362902e-01, -2.87490813e-01,
        3.56689618e-02,  2.13525997e-01,  4.24638383e-02,  3.48042175e-01,
       -2.58928062e-01, -4.75616430e-02, -3.57015803e-01, -1.56729981e-01,
        9.43768708e-02,  4.26144451e-01,  9.33714798e-02, -1.62515456e-01,
       -2.44379092e-01, -3.33107604e-01,  2.66189200e-01, -1.24631005e-01,
        1.01211452e-01, -1.45257932e-01, -3.78942405e-01, -4.04472810e-01,
        1.65080451e-01,  1.06944420e-01,  1.75579614e-01, -1.77670191e-01,
       -7.96313351e-02, -3.31333958e-01,  1.59069216e-01, -2.57217079e-01,
        1.21228105e-01,  1.48528192e-01, -7.03231290e-02,  2.38417179e-01,
        8.15077575e-03, -2.69057425e-01,  1.08341680e-01,  2.67081350e-02,
        4.24267350e-02, -5.23549859e-02,  8.09309084e-02,  3.17922750e-01,
       -2.80300722e-02, -1.37572163e-01, -3.03894751e-01,  4.31688014e-02,
        7.39512064e-02,  4.87384170e-01, -3.46702868e-01,  1.52301553e-01,
       -3.50047563e-02,  9.83756840e-02,  2.53537236e-01, -1.38730862e-01,
        4.54589889e-02,  9.38341753e-02, -1.81441277e-01, -8.50300511e-02,
       -4.27041662e-02, -1.02888769e-02,  1.05808127e-01, -2.47687145e-02,
       -5.11370676e-01, -2.14097950e-01, -1.57757556e-01, -2.63902998e-02,
       -1.53463430e-01, -1.92711074e-01, -1.22746401e-01,  1.02159537e-01,
       -2.26267519e-01,  1.32578865e-01,  1.41437542e-01,  2.76331128e-02,
       -4.35941373e-01,  3.10335899e-01, -1.56419634e-01, -9.84582743e-03,
       -1.55307894e-01,  8.83870128e-02, -2.99400128e-01, -7.63911052e-02,
       -1.19783653e-01, -1.09748464e-01,  8.76607589e-02, -1.17738021e-02,
       -7.83155037e-02, -5.04185031e-02, -1.95681826e-01,  8.27137292e-03,
       -5.92389561e-02,  3.35033585e-02, -3.22928797e-01,  1.96645421e-01,
        9.63587808e-02,  2.15791790e-01,  4.10898833e-02,  2.92664265e-01,
       -1.69333015e-01, -1.26905786e-01,  1.16476747e-01, -4.93162075e-02,
        1.43544121e-01,  2.01119914e-02, -8.19131896e-02,  1.14117493e-01,
       -3.01016758e-01,  1.82979765e-02,  4.74072987e-01, -2.39767641e-02,
       -1.72706556e-01, -7.66260870e-02,  4.99203685e-01, -2.09450335e-01,
       -1.71282676e-02,  1.64066573e-01,  2.14281759e-01, -1.05559490e-01,
       -3.71066933e-02,  1.13765204e-01,  7.61300519e-02,  1.54894384e-01,
       -5.01861499e-02,  3.39044305e-02,  3.28827518e-01, -5.45867246e-01,
       -5.83260228e-02,  2.03569053e-02, -1.20708898e-02,  9.60980830e-02,
        3.33183484e-01,  1.33007263e-02,  2.67571742e-01,  6.31606178e-02,
       -2.67598344e-02,  6.65253388e-02, -1.55726360e-01,  2.07648851e-01,
       -8.71996192e-02, -2.56724629e-01, -1.48292435e-01,  1.08600528e-01,
        1.11683426e-02, -7.13029099e-02,  1.75144304e-01, -7.27418388e-02,
        2.20124687e-02, -4.08824828e-01, -1.57281505e-01,  5.33036119e-02,
        8.61857304e-02, -3.52940666e-01, -9.02578388e-02, -1.13690382e-01,
       -8.67652548e-02, -2.24443420e-01,  3.93362159e-02,  9.71562689e-02,
       -1.71484387e-01,  2.00617587e-01,  2.92491388e-02,  1.87253955e-01,
       -1.37634579e-01,  2.23600788e-01,  2.57963986e-02, -1.79837833e-02,
        1.35056891e-01,  1.38971660e-01,  3.63906549e-02, -2.45788917e-01,
        1.11147303e-01, -7.69312816e-02, -2.87757861e-01, -2.09005930e-01,
       -1.76918854e-01, -2.07378483e-01,  8.65385764e-02, -6.98265887e-02,
        4.42561997e-02, -5.13436222e-02,  8.76158714e-02, -2.82627650e-01,
       -3.15084666e-01,  5.91144944e-01,  8.53821114e-02, -3.00324828e-01])
)
```

Add this into the class:

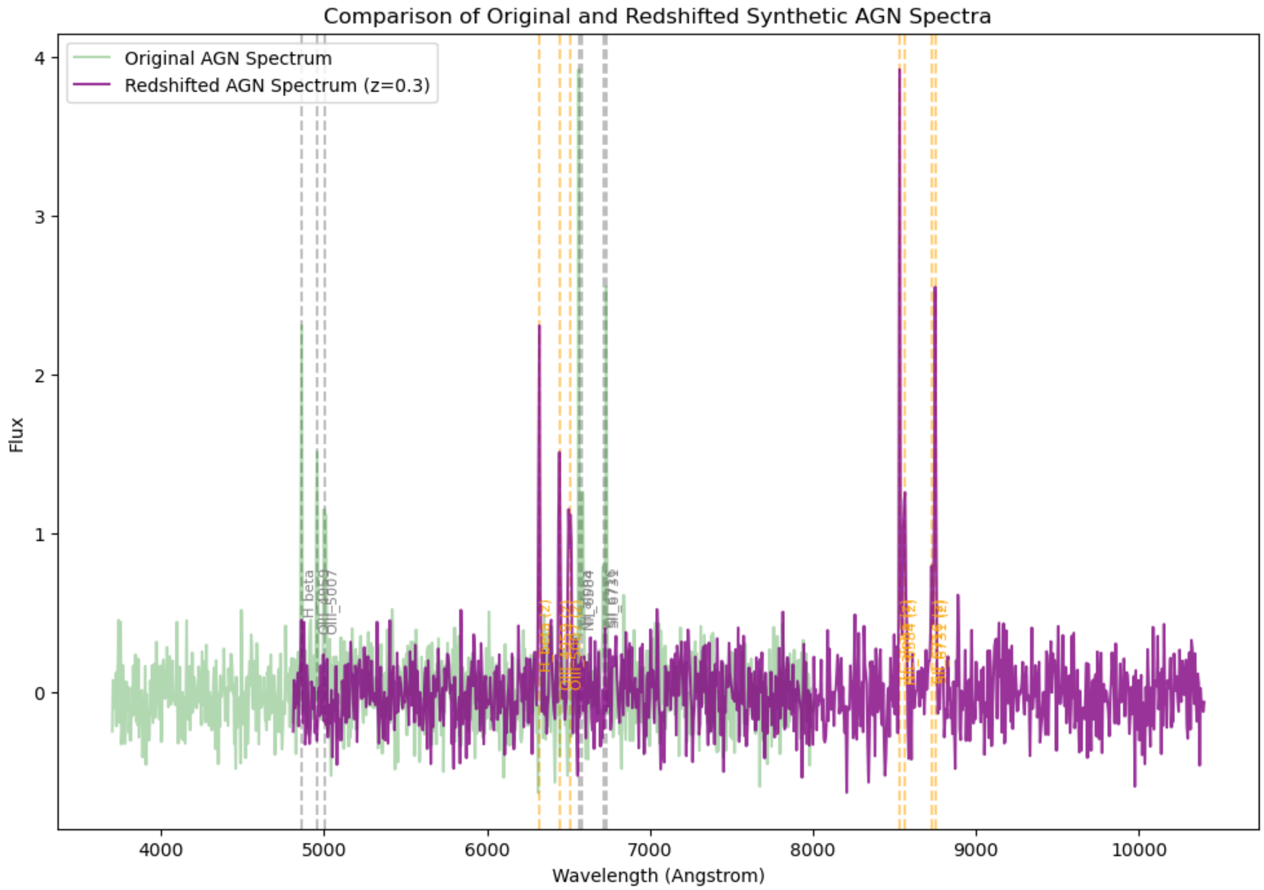
In [31]:
```
class Galaxy:
    '''initialize the galaxy'''
    def __init__(self, mass, position, velocity, black_hole_mass, initial_te
        self.mass = mass
        self.position = position
        self.velocity = velocity
        self.black_hole_mass = black_hole_mass

    '''black hole accretion model'''
    def calculate_accretion_rate(black_hole_mass,
                                 rho = np.random.uniform(1e-14, 1e-13),
                                 v = np.random.uniform(4.5e7, 5.5e7),
                                 c_s = np.random.uniform(1e6, 2e7)):
        # The input black hole mass should in the unit of solar mass
        G = 6.67430e-11 * u.m**3 / u.kg / u.s**2 # gravitational constant wi
        M = black_hole_mass * 1.98847e30 * u.kg  # solar masses to kg
        rho = rho * u.kg / u.m**3  # density of the accreting material in kg
        v = v * u.m / u.s  # relative velocity in m/s
        c_s = c_s * u.m / u.s # sound speed in the material in m/s
        # Bondi-Hoyle-Lyttleton accretion rate
        accretion_rate = (4 * np.pi * G**2 * M**2 * rho) / ((v**2 + c_s**2)*
        # Convert accretion rate to solar masses per year
        accretion_rate_solar_masses_per_year = accretion_rate.to(u.kg / u.s)
        return accretion_rate_solar_masses_per_year

    def calculate_radiation(accretion_rate, epsilon = 0.1):
        # The accretion rate is obtained from the previous function calculat
        c = 299792458 * u.m / u.s
        # calculate the luminosity (or radiation)
        radiation = epsilon * accretion_rate * 1.98847e30 * c**2  # Convert
        return radiation

    def accretion_model(black_hole_mass, accretion_radius, other_parameters)
        accretion_rate = calculate_accretion_rate(black_hole_mass, accretion
        radiation = calculate_radiation(accretion_rate, other_parameters)
        jets = determine_jet_production(accretion_rate, radiation, other_par
        return accretion_rate, radiation, jets

    '''cooling and heating process in different radius of the galaxy'''
    def heating_function(r,
                         time,
                         agn_luminosity_initial = 1e44,
                         star_formation_initial = 2.0,
                         rho_0 = 1e5,
                         r_0 = 1,
                         alpha = 1.5):
        decay_factor = 0.05
        star_formation_rate = star_formation_initial / (1 + decay_factor * t
        agn_luminosity = agn_luminosity_initial / (1 + decay_factor * time)
        density = density_radius(r, rho_0, r_0, alpha)
        new_temp = 1e-35 * (star_formation_rate + agn_luminosity) * density
        return new_temp
```

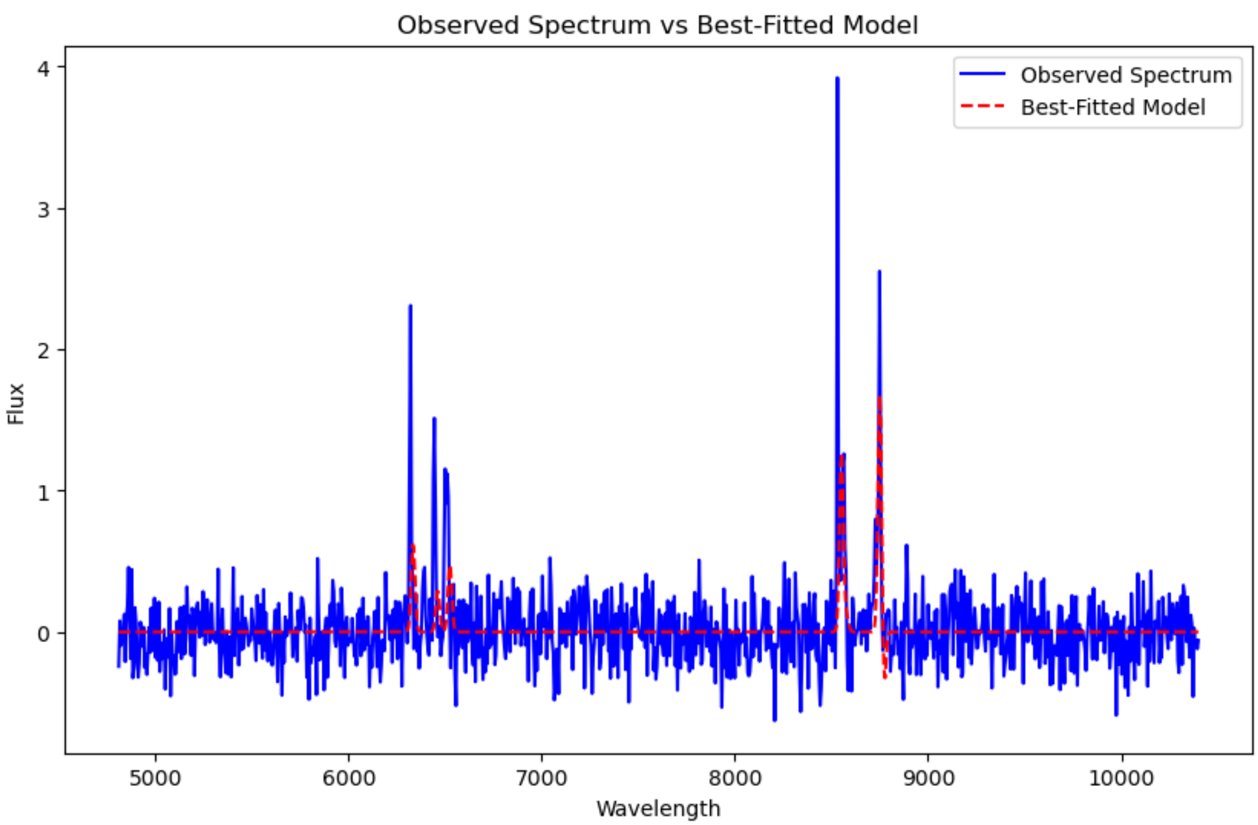
```
def cooling_function(temperature, r, rho_0 = 1e5, r_0 = 1, alpha = 1.5):
    density = density_radius(r, rho_0, r_0, alpha)
    if temperature < 1e4:
        new_temp = 1e-22 * temperature * density**2
    else:
        new_temp = 1e-24 * temperature**1.5 * density**2
    return new_temp

def cooling_rate(new_temp, density):
    cooling_rate = new_temp * density**2
    return cooling_rate

def ism_temperature_evolution(initial_temperature,
                              r,
                              metallicity,
                              star_formation_rate,
                              agn_luminosity, timestep, total_time):
    temperature = initial_temperature
    density = density_radius(r, rho_0, r_0, alpha)
    for time in range(total_time):
        # cooling and heating rates
        cooling_rate = cooling_function(temperature, r, rho_0 = 1e5, r_0
        heating_rate = heating_function(star_formation_rate, agn_luminos
        temperature_change = (heating_rate - cooling_rate) * timestep
        temperature += temperature_change
        if temperature < 10:  # 10 K is a reasonable lower limit for ISM
            temperature = 10
    return temperature

def ism_temperature_evolution_over_time(initial_temperature,
                                        r,
                                        rho_0=1e5,
                                        r_0=1,
                                        alpha=1.5,
                                        metallicity=1.0,
                                        star_formation_initial=2.0,
                                        agn_luminosity_initial=1e44,
                                        timestep=1e3,
                                        total_time=10000):
    temperature = initial_temperature
    temperatures, cooling_rates, heating_rates = [], [], []
    for time in range(total_time):
        density = density_radius(r, rho_0, r_0, alpha)
        cooling_rate = cooling_function(temperature, r, rho_0, r_0, alph
        heating_rate = heating_function(star_formation_initial, agn_lumi
        temperature_change = (heating_rate - cooling_rate) * timestep
        temperature += temperature_change
        if temperature < 10:
            temperature = 10
        temperatures.append(temperature)
        cooling_rates.append(cooling_rate)
        heating_rates.append(heating_rate)
```

```
        return temperatures, cooling_rates, heating_rates

    '''spectra modeling'''
    def generate_redshifted_spectrum(redshift):
        wavelengths = np.linspace(3700, 8000, 1000)
        alpha = -15
        flux_continuum = wavelengths ** alpha

        emission_lines = {
            "H_alpha": {"center": 6563, "element": "H"},
            "H_beta": {"center": 4861, "element": "H"},
            "OIII_5007": {"center": 5007, "element": "O"},
            "OIII_4959": {"center": 4959, "element": "O"},
            "NII_6584": {"center": 6584, "element": "N"},
            "SII_6716": {"center": 6716, "element": "S"},
            "SII_6731": {"center": 6731, "element": "S"}
        }

        def add_emission_line(wavelength_center, width, height):
            return height * norm.pdf(wavelengths, wavelength_center, width)

        width_range = (2, 10)
        base_height_range = (10, 20)

        for line in emission_lines.values():
            width = random.uniform(*width_range)
            height = random.uniform(*base_height_range)
            if line["element"] in ["H", "O", "N"]:
                height *= 1.5
            flux_continuum += add_emission_line(line["center"], width, heigh

        noise_level = 0.2
        noise = np.random.normal(0, noise_level, wavelengths.shape)
        flux_continuum += noise

        original_flux_continuum = flux_continuum.copy()
        redshifted_wavelengths = wavelengths * (1 + redshift)
        redshifted_flux_continuum = original_flux_continuum

        plt.figure(figsize=(12, 8))
        for line_name, line in emission_lines.items():
            original_center = line["center"]
            redshifted_center = original_center * (1 + redshift)
            plt.axvline(x=original_center, color='gray', linestyle='--', alp
            plt.axvline(x=redshifted_center, color='orange', linestyle='--',
            plt.text(original_center, plt.ylim()[1]*0.8, line_name, rotation
            plt.text(redshifted_center, plt.ylim()[1]*0.6, f'{line_name} (z)

        plt.plot(wavelengths, original_flux_continuum, label='Original AGN S
        plt.plot(redshifted_wavelengths, redshifted_flux_continuum, label=f'
        plt.xlabel('Wavelength (Angstrom)')
        plt.ylabel('Flux')
        plt.title('Comparison of Original and Redshifted Synthetic AGN Spect
```

```
        plt.legend()
        plt.show()
        return redshifted_wavelengths, flux_continuum

    # Gas Dynamics and Interstellar Medium
    def gas_dynamics(galaxy):
        '''
        This models the impliment model of the dynamics of gas in and around
        The processes like cooling, heating, and interaction with AGN radiat
        '''
        pass

    # Galaxy Interaction and Dynamics
    def update_galaxy_dynamics(galaxies):
        '''
        Implement gravitational interactions between galaxies.
        Update positions and velocities based on gravitational forces.
        '''
        pass
```

In [32]:
```
observed_wavelengths, observed_flux = generate_redshifted_spectrum(0.3)
emission_line_centers = [6563, 4861, 5007, 4959, 6584, 6716, 6731]
line_width = 10

def model_spectrum(wavelengths, redshift, line_intensities):
    model_flux = wavelengths ** -1.5
    for i, line_center in enumerate(emission_line_centers):
        shifted_center = line_center * (1 + redshift)
        model_flux += line_intensities[i] * norm.pdf(wavelengths, shifted_ce
    return model_flux

def chi_squared(obs_flux, model_flux):
    return np.sum((obs_flux - model_flux) ** 2)

def fit_spectrum(line_intensities, redshift):
    model_flux = model_spectrum(observed_wavelengths, redshift, line_intensi
    return chi_squared(observed_flux, model_flux)


N = len(emission_line_centers)  # number of emission lines
initial_line_intensities = [10] * N  # Initial guess for each line intensity

# Optimization Model
best_fit = None
for redshift in np.linspace(0, 1, 100):
    result = minimize(fit_spectrum, initial_line_intensities, args=(redshift
    if best_fit is None or result.fun < best_fit[1]:
        best_fit = (redshift, result.fun, result.x)

best_fit_flux = model_spectrum(observed_wavelengths, best_fit[0], best_fit[2

plt.figure(figsize=(10, 6))
plt.plot(observed_wavelengths, observed_flux, label="Observed Spectrum", col
plt.plot(observed_wavelengths, best_fit_flux, label="Best-Fitted Model", col
plt.xlabel("Wavelength")
plt.ylabel("Flux")
plt.title("Observed Spectrum vs Best-Fitted Model")
plt.legend()
plt.show()

print(f"Best fit redshift: {best_fit[0]}")
print(f"Best fit line intensities: {best_fit[2]}")
```

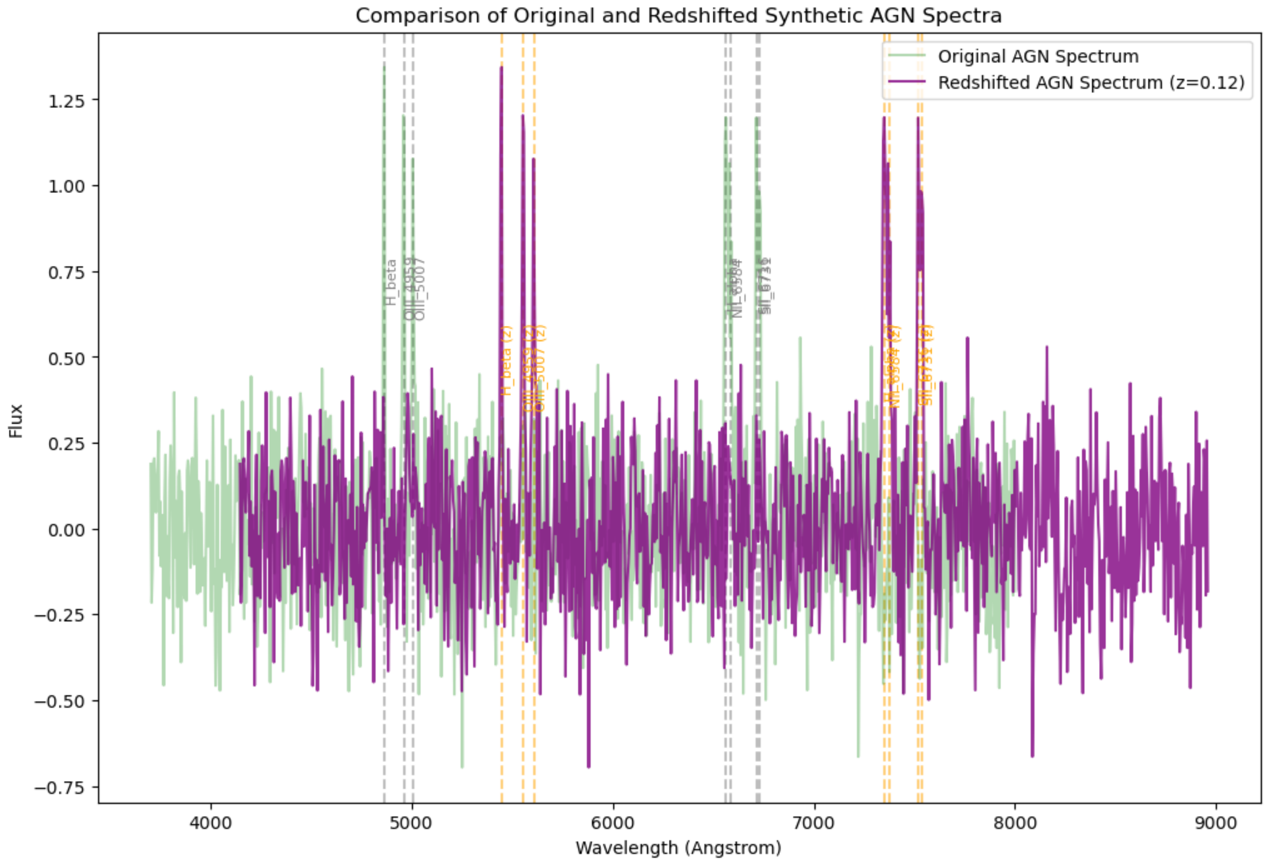

Comparison of Original and Redshifted Synthetic AGN Spectra
Original AGN Spectrum
Redshifted AGN Spectrum (z=0.3)
Flux
Wavelength (Angstrom)

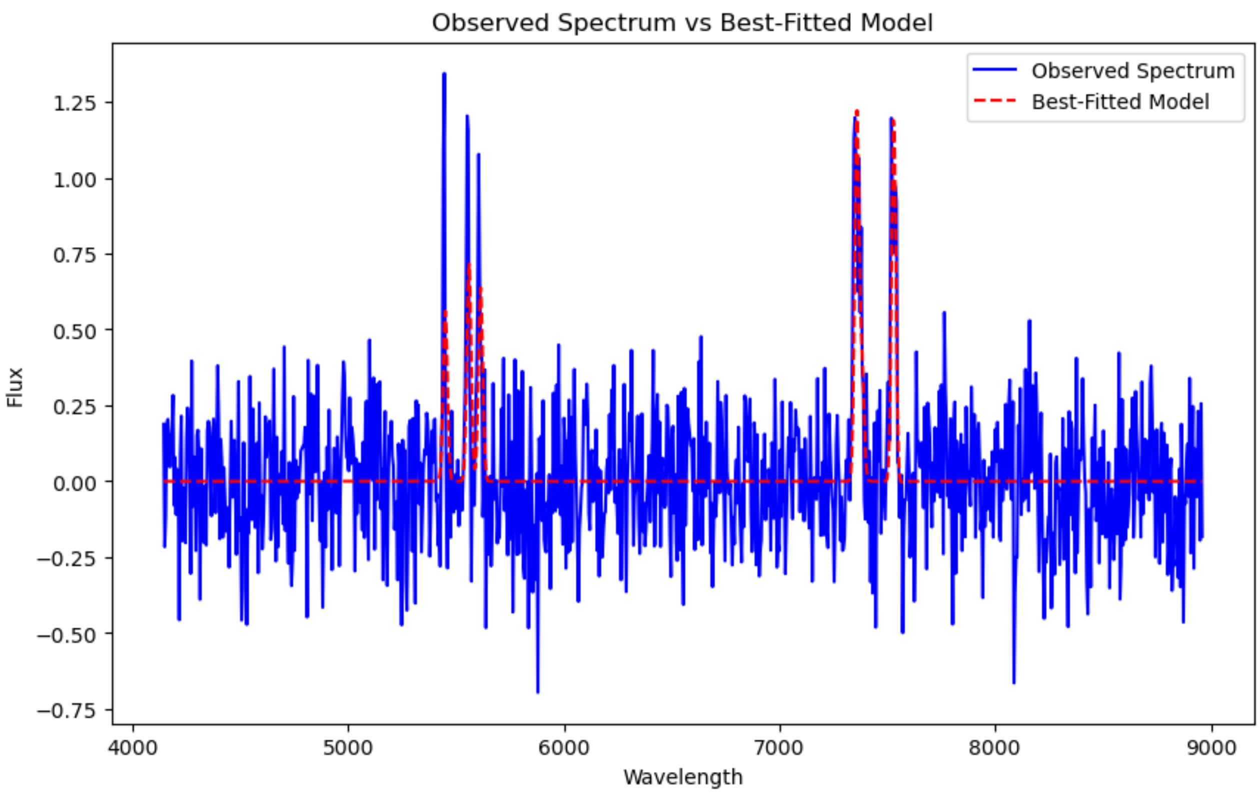

Observed Spectrum vs Best-Fitted Model
Observed Spectrum
Best-Fitted Model
Flux
Wavelength

```
Best fit redshift: 0.30303030303030304
Best fit line intensities: [ 32.10930647  15.80354273  12.08655795   7.30040
496   2.64035827
  43.95218399 -11.92880199]
```

In [33]:

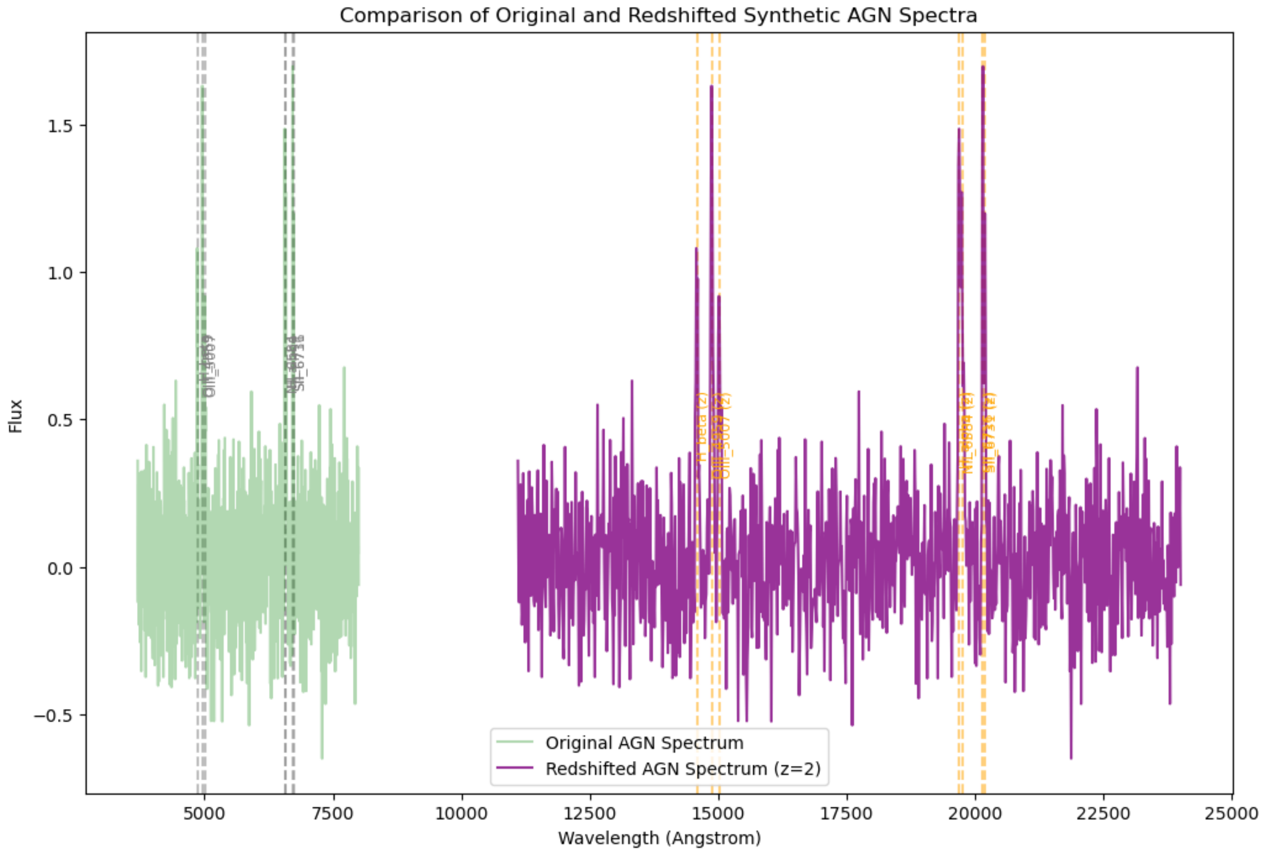
```
# do another attempt
observed_wavelengths, observed_flux = generate_redshifted_spectrum(0.12)
emission_line_centers = [6563, 4861, 5007, 4959, 6584, 6716, 6731]
line_width = 10

def model_spectrum(wavelengths, redshift, line_intensities):
    model_flux = wavelengths ** -1.5
    for i, line_center in enumerate(emission_line_centers):
        shifted_center = line_center * (1 + redshift)
        model_flux += line_intensities[i] * norm.pdf(wavelengths, shifted_ce
    return model_flux

def chi_squared(obs_flux, model_flux):
    return np.sum((obs_flux - model_flux) ** 2)

def fit_spectrum(line_intensities, redshift):
    model_flux = model_spectrum(observed_wavelengths, redshift, line_intensi
    return chi_squared(observed_flux, model_flux)


N = len(emission_line_centers)  # number of emission lines
initial_line_intensities = [10] * N  # Initial guess for each line intensity

# Optimization Model
best_fit = None
for redshift in np.linspace(0, 1, 100):
    result = minimize(fit_spectrum, initial_line_intensities, args=(redshift
    if best_fit is None or result.fun < best_fit[1]:
        best_fit = (redshift, result.fun, result.x)

best_fit_flux = model_spectrum(observed_wavelengths, best_fit[0], best_fit[2

plt.figure(figsize=(10, 6))
plt.plot(observed_wavelengths, observed_flux, label="Observed Spectrum", col
plt.plot(observed_wavelengths, best_fit_flux, label="Best-Fitted Model", col
plt.xlabel("Wavelength")
plt.ylabel("Flux")
plt.title("Observed Spectrum vs Best-Fitted Model")
plt.legend()
plt.show()

print(f"Best fit redshift: {best_fit[0]}")
print(f"Best fit line intensities: {best_fit[2]}")
```

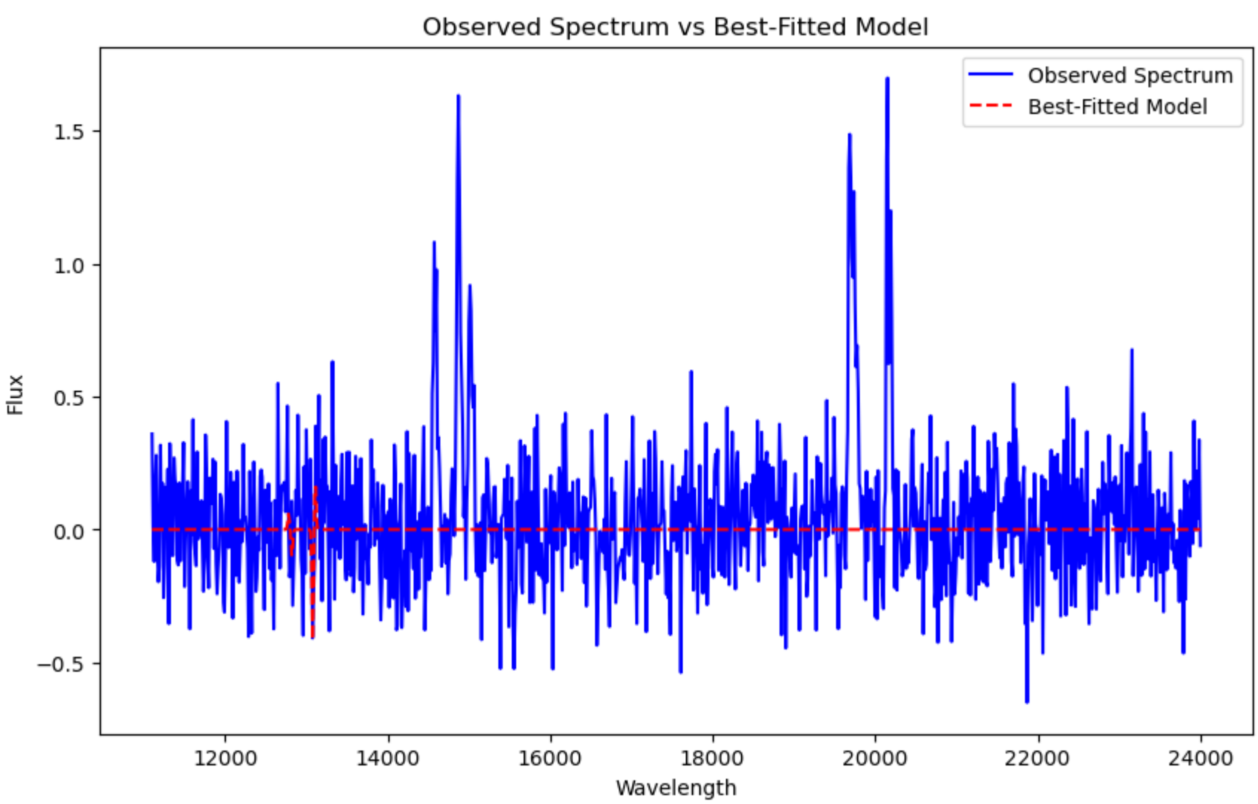





```
Best fit redshift: 0.12121212121212122
Best fit line intensities: [30.027984   14.06055572 16.00710874 18.0805335
9.34198316 30.43715122
 -0.68771816]
```

In [42]:
```
# this does not work for extremely large z (example z = 2) It cannot be corr
observed_wavelengths, observed_flux = generate_redshifted_spectrum(2)
emission_line_centers = [6563, 4861, 5007, 4959, 6584, 6716, 6731]
line_width = 10

def model_spectrum(wavelengths, redshift, line_intensities):
    model_flux = wavelengths ** -1.5
    for i, line_center in enumerate(emission_line_centers):
        shifted_center = line_center * (1 + redshift)
        model_flux += line_intensities[i] * norm.pdf(wavelengths, shifted_ce
    return model_flux

def chi_squared(obs_flux, model_flux):
    return np.sum((obs_flux - model_flux) ** 2)

def fit_spectrum(line_intensities, redshift):
    model_flux = model_spectrum(observed_wavelengths, redshift, line_intensi
    return chi_squared(observed_flux, model_flux)


N = len(emission_line_centers)  # number of emission lines
initial_line_intensities = [10] * N  # Initial guess for each line intensity

# Optimization Model
best_fit = None
for redshift in np.linspace(0, 1, 20):
    result = minimize(fit_spectrum, initial_line_intensities, args=(redshift
    if best_fit is None or result.fun < best_fit[1]:
        best_fit = (redshift, result.fun, result.x)

best_fit_flux = model_spectrum(observed_wavelengths, best_fit[0], best_fit[2

plt.figure(figsize=(10, 6))
plt.plot(observed_wavelengths, observed_flux, label="Observed Spectrum", col
plt.plot(observed_wavelengths, best_fit_flux, label="Best-Fitted Model", col
plt.xlabel("Wavelength")
plt.ylabel("Flux")
plt.title("Observed Spectrum vs Best-Fitted Model")
plt.legend()
plt.show()

print(f"Best fit redshift: {best_fit[0]}")
print(f"Best fit line intensities: {best_fit[2]}")
```



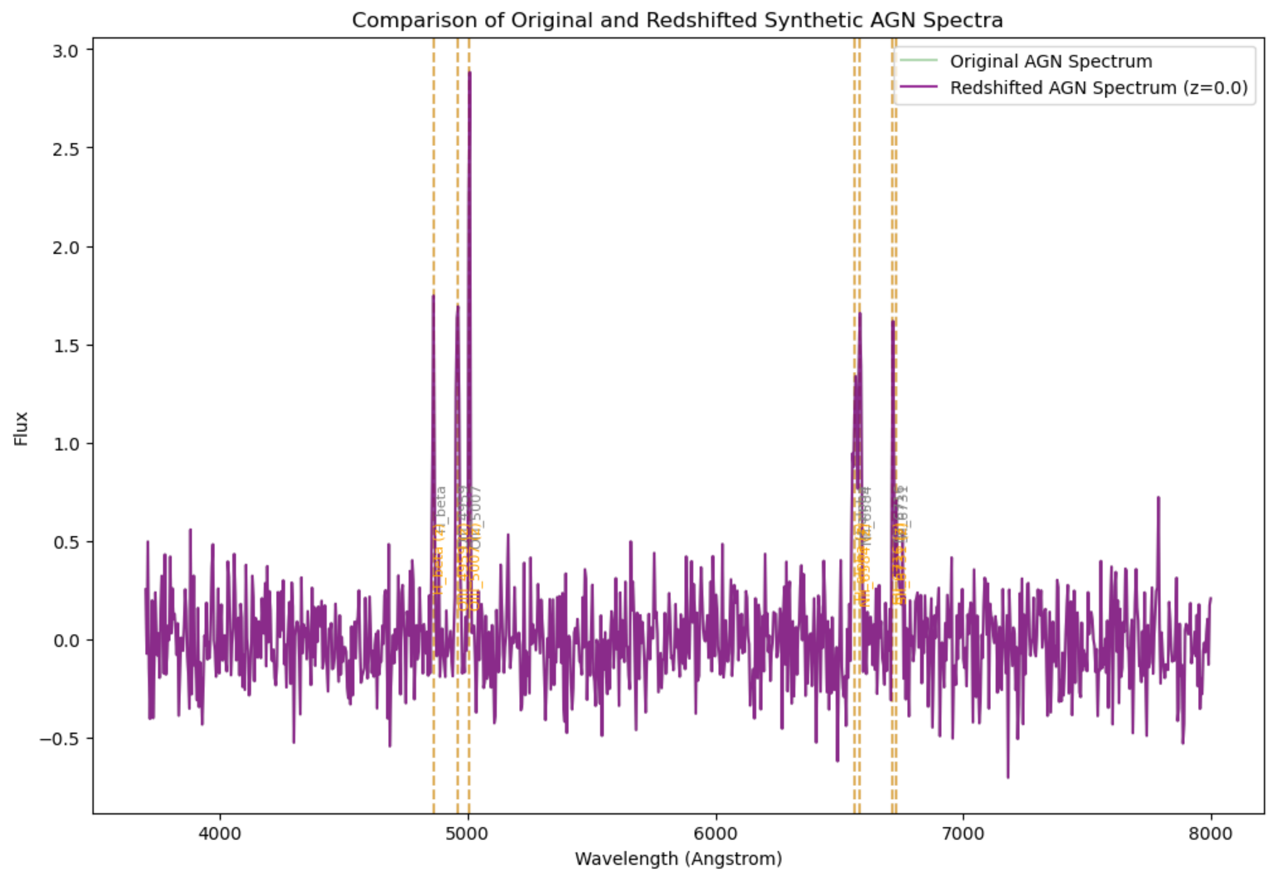


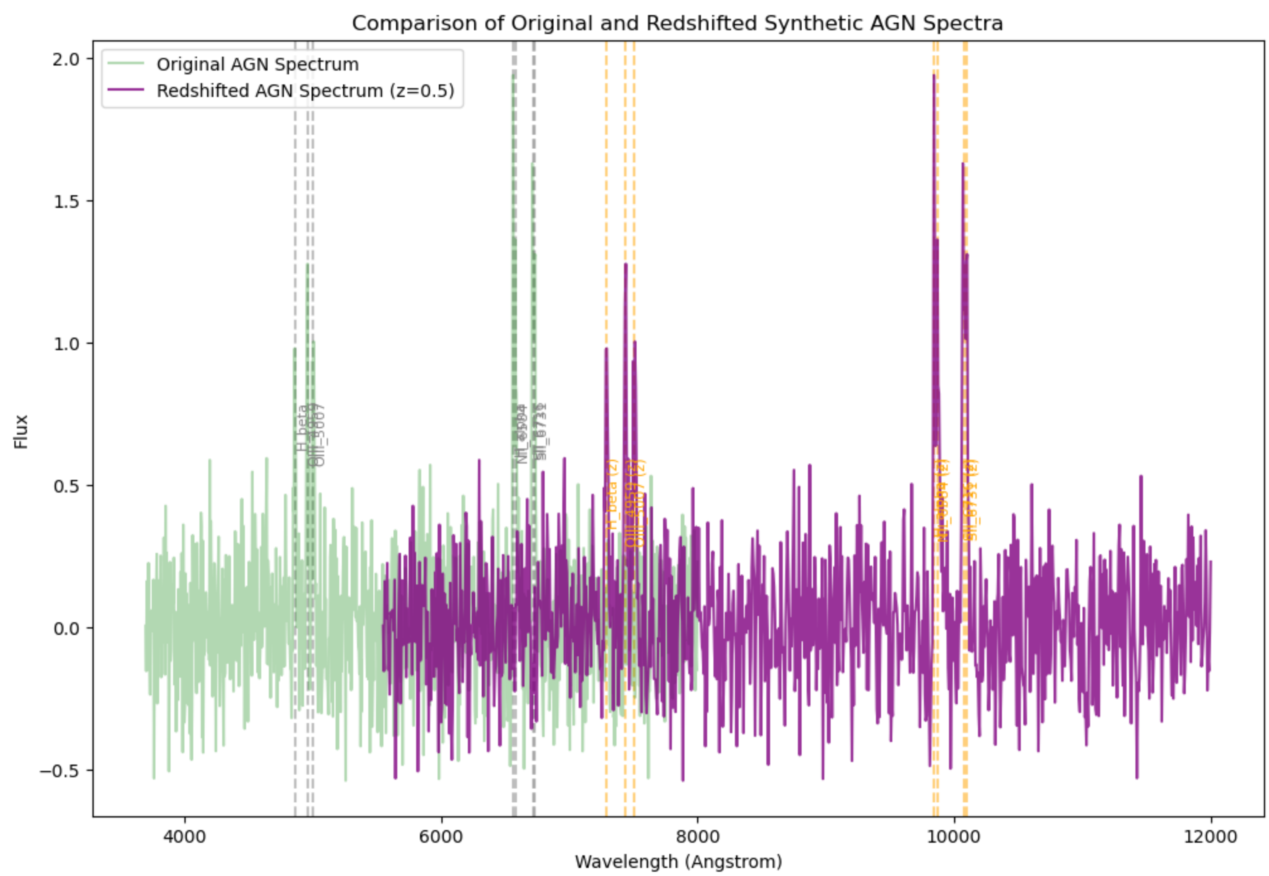

```
Best fit redshift: 0.9473684210526315
Best fit line intensities: [  1.49175322  10.          10.          10.
-2.6171938
 -10.83645787   5.03885183]
```

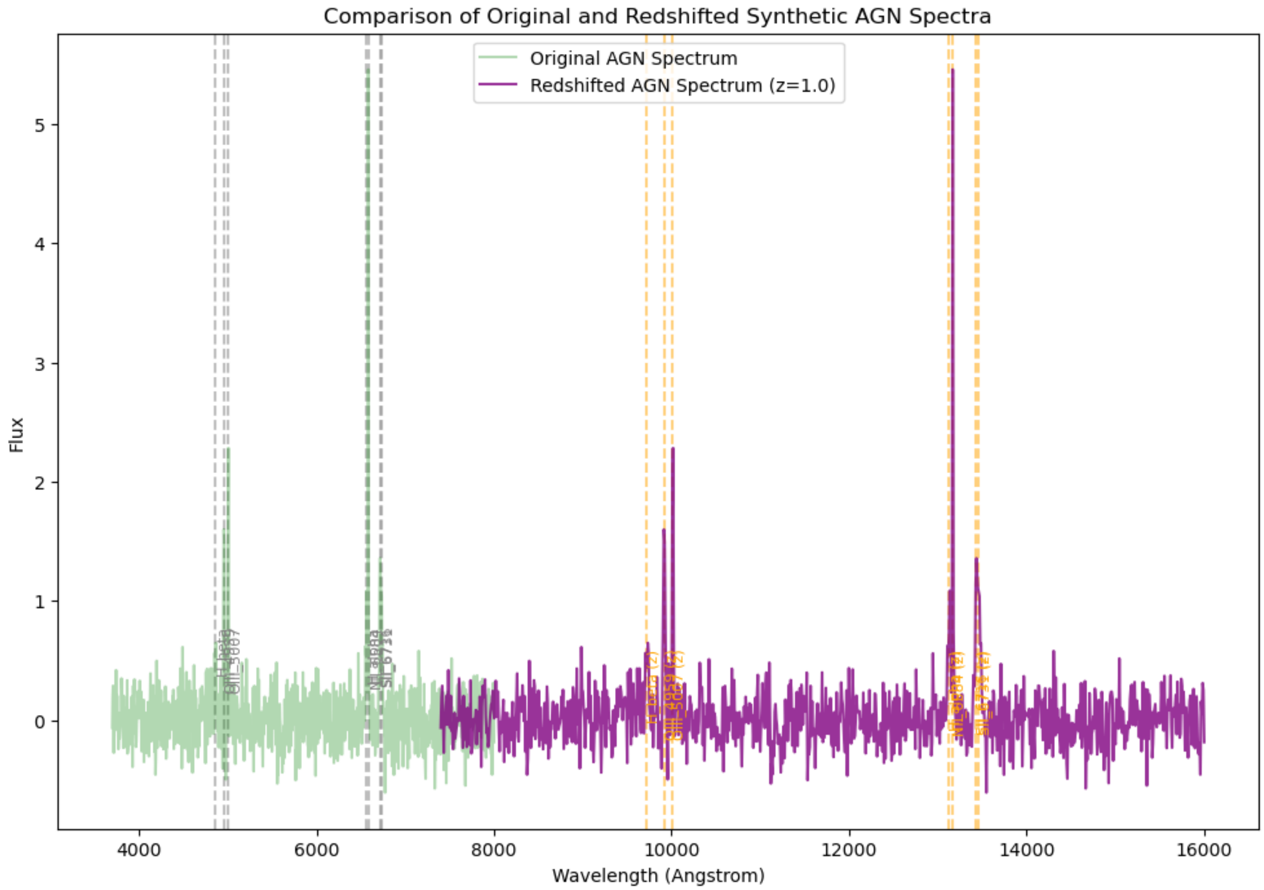
Week 3

Last time, I generated the spectrum for an AGN galaxy by specifying the emission lines and intensity. I then used optimization to find the best fit of the spectra and compute the redshift with a known "observed" spectra. However, as I discovered at the end of last time that the estimation becomes inaccurate at certain point. I here want to plot the error and the comparison between actural redshift and my estimated redshift out, to see where the accuracy "breaks".

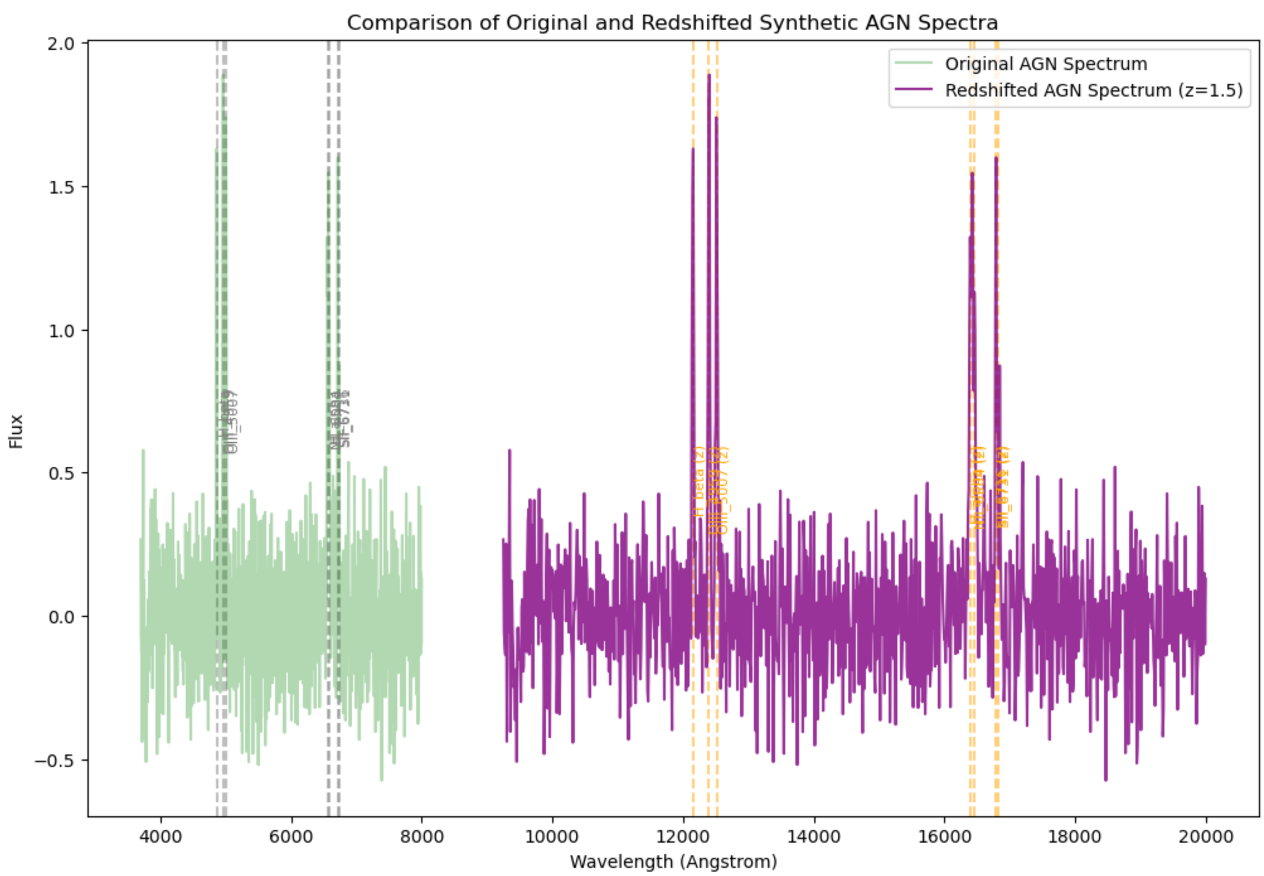
In [41]:
```
test_redshifts = np.linspace(0, 2, 5)  #test redshifts with this range
estimated_redshifts = []
errors = []

N = len(emission_line_centers)  # Number of emission lines

for test_redshift in test_redshifts:
    observed_wavelengths, observed_flux = generate_redshifted_spectrum(test_
    initial_line_intensities = [10] * N  # Initial guess for line intensitie

    # Optimization Model
    best_fit = None
    for redshift in np.linspace(0, 1, 20):  # Fewer steps in the redshift ra
        result = minimize(fit_spectrum, initial_line_intensities, args=(reds
        if best_fit is None or result.fun < best_fit[1]:
            best_fit = (redshift, result.fun, result.x)

    estimated_redshifts.append(best_fit[0])
    errors.append(abs(best_fit[0] - test_redshift))

plt.figure(figsize=(10, 6))
plt.plot(test_redshifts, estimated_redshifts, label='Estimated Redshift', ma
plt.plot(test_redshifts, test_redshifts, label='Actual Redshift', linestyle=
plt.xlabel('Input Redshift')
plt.ylabel('Estimated Redshift')
plt.title('Estimated vs Actual Redshift')
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(test_redshifts, errors, label='Error in Redshift Estimation', marke
plt.xlabel('Input Redshift')
plt.ylabel('Error')
plt.title('Error in Redshift Estimation')
plt.legend()
plt.grid()
plt.show()
```


Comparison of Original and Redshifted Synthetic AGN Spectra
Original AGN Spectrum
Redshifted AGN Spectrum (z=0.0)
Flux
Wavelength (Angstrom)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
4000
5000
6000
7000
8000

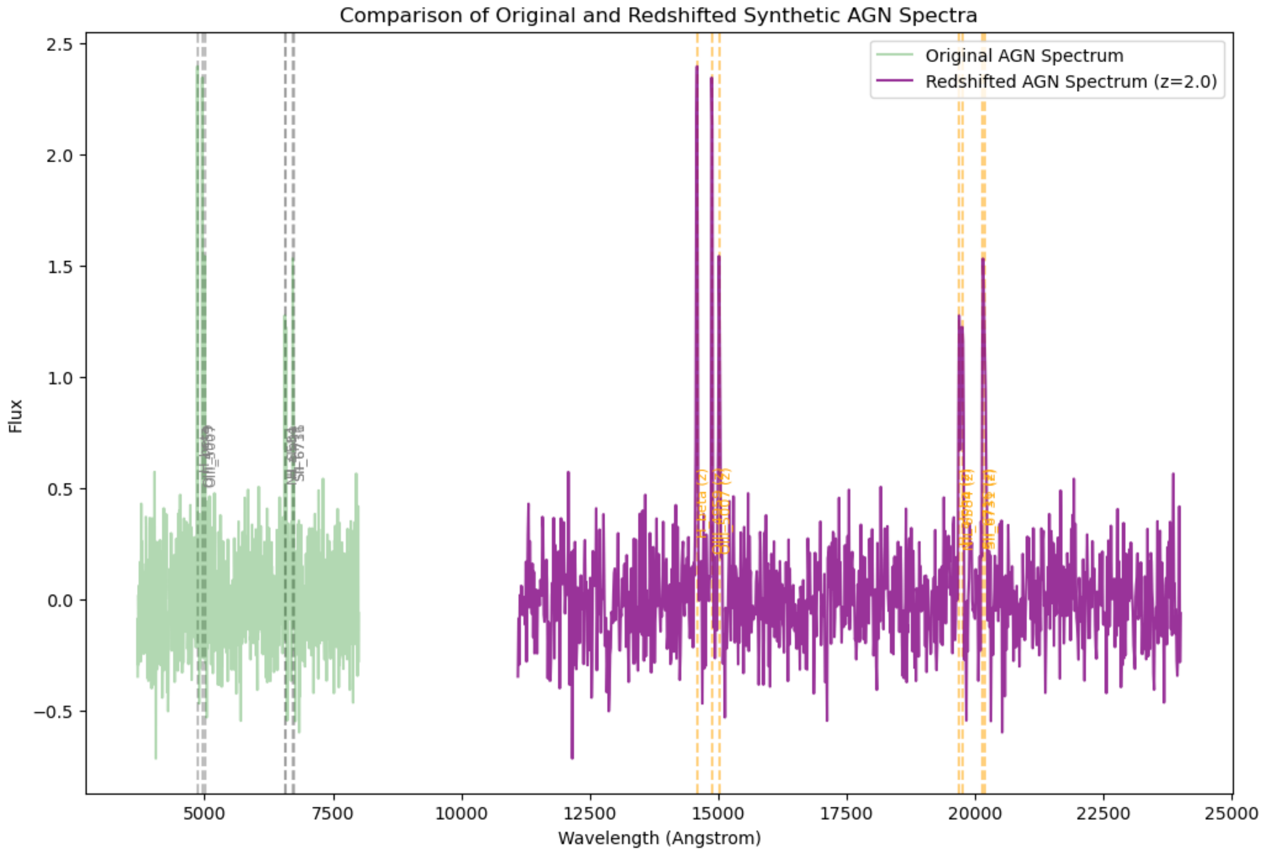

Comparison of Original and Redshifted Synthetic AGN Spectra
Original AGN Spectrum
Redshifted AGN Spectrum (z=0.5)
Flux
Wavelength (Angstrom)
2.0
1.5
1.0
0.5
0.0
−0.5
4000
6000
8000
10000
12000

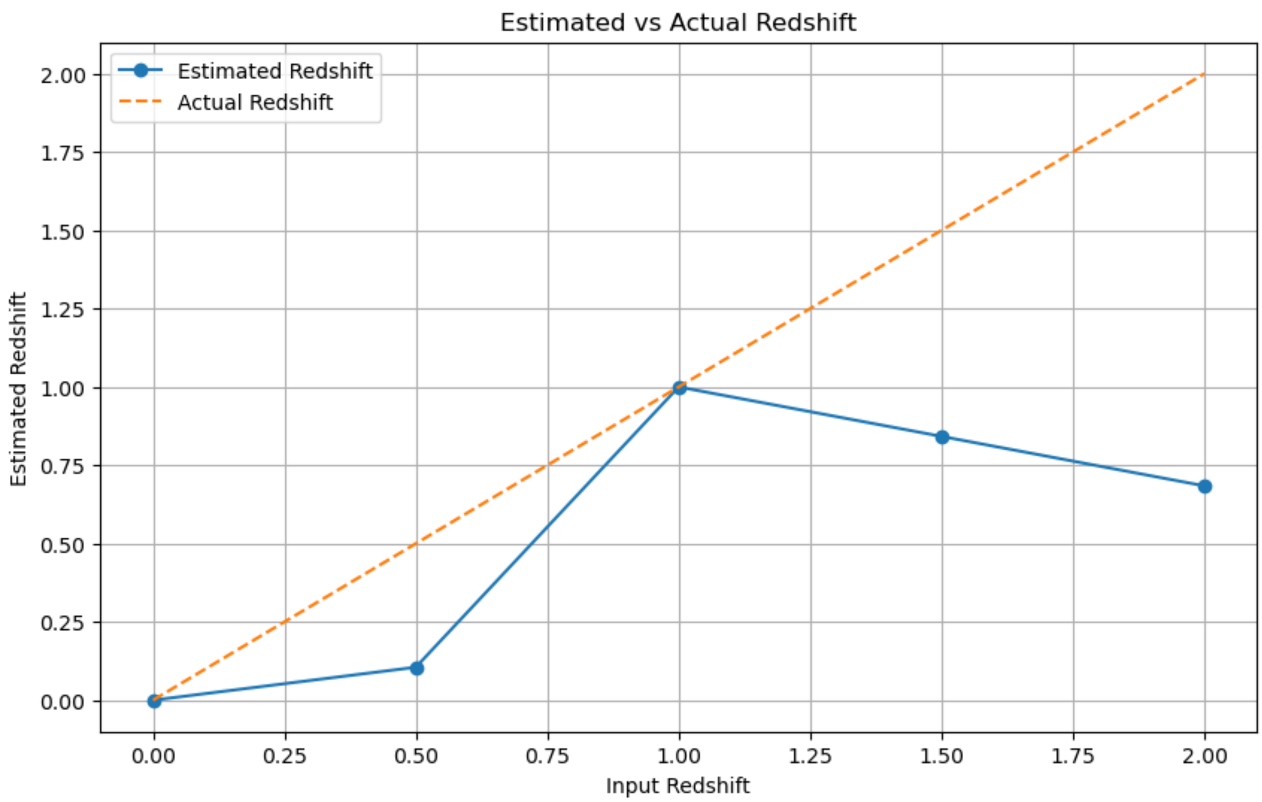


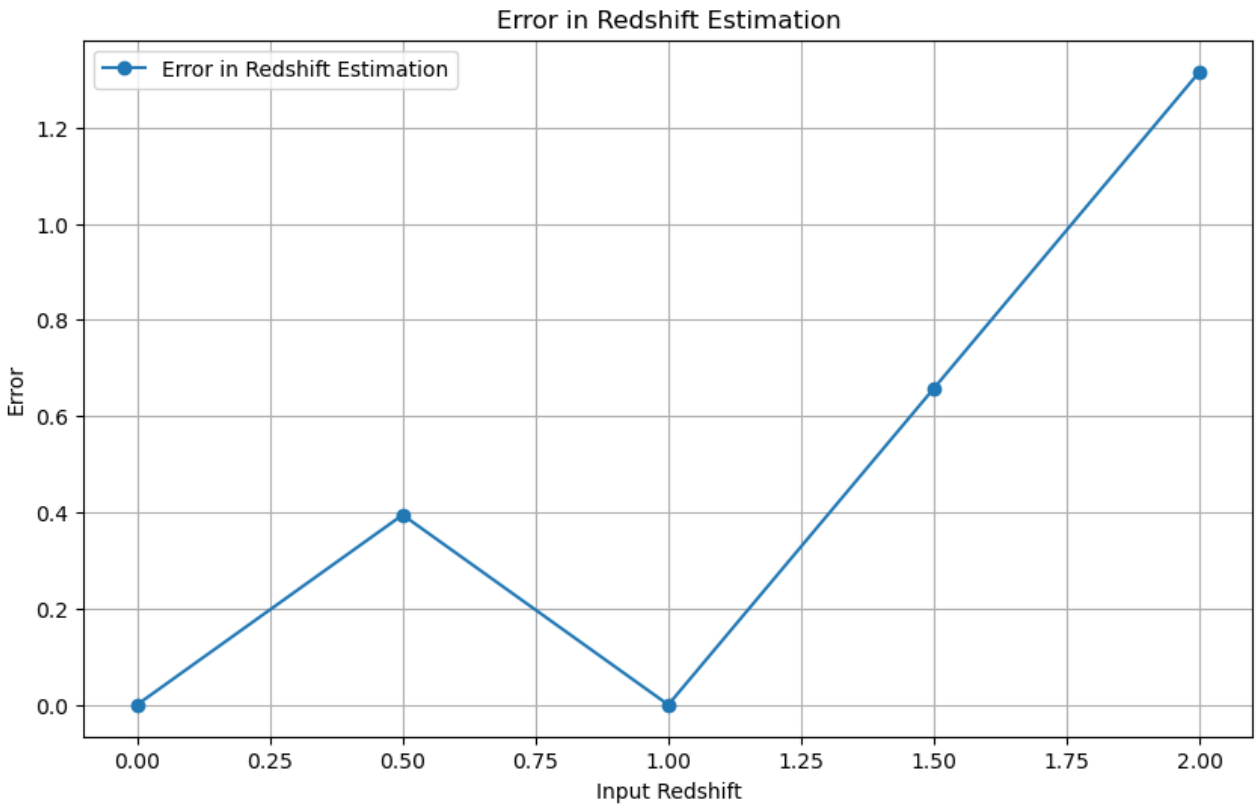
Comparison of Original and Redshifted Synthetic AGN Spectra
Original AGN Spectrum
Redshifted AGN Spectrum (z=1.0)
Flux
Wavelength (Angstrom)



Comparison of Original and Redshifted Synthetic AGN Spectra
Original AGN Spectrum
Redshifted AGN Spectrum (z=1.5)
Flux
Wavelength (Angstrom)

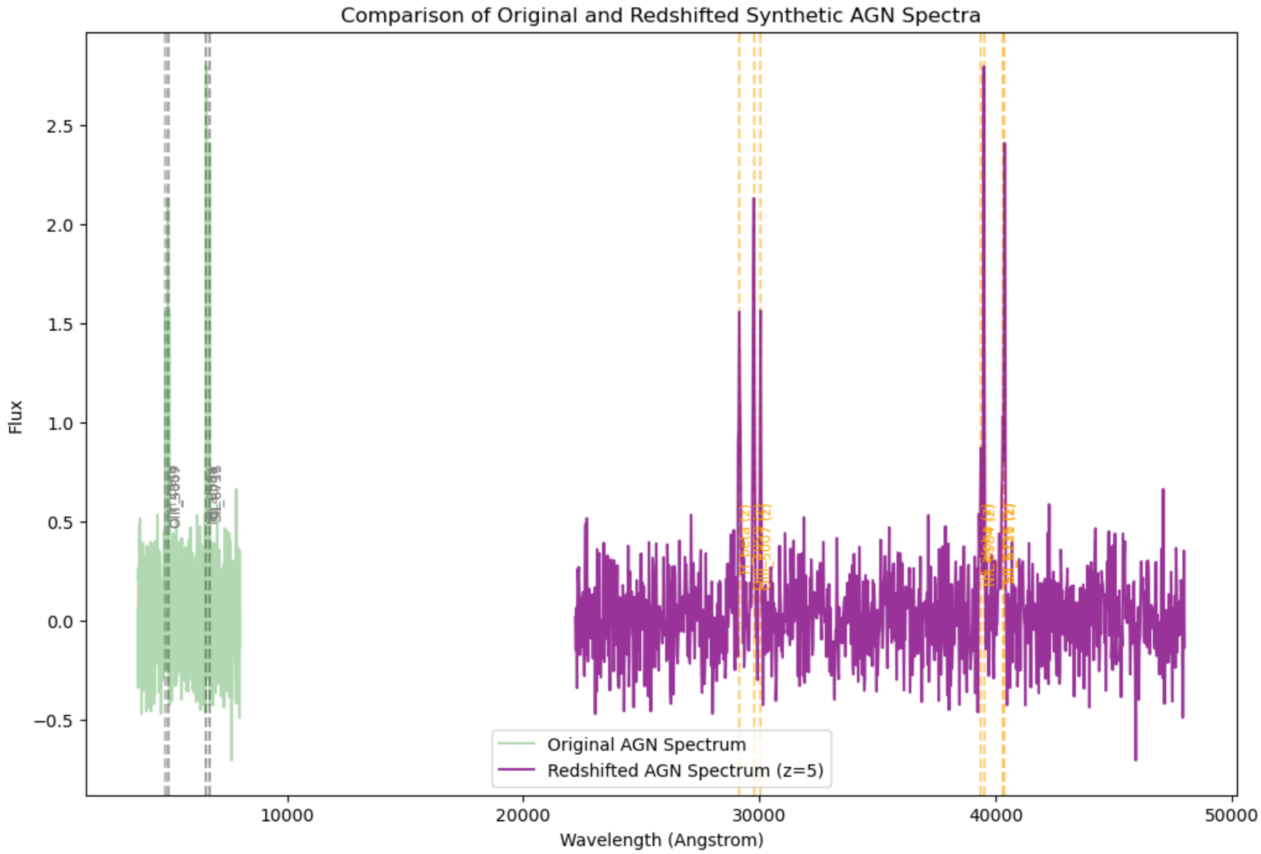

Comparison of Original and Redshifted Synthetic AGN Spectra
Original AGN Spectrum
Redshifted AGN Spectrum (z=2.0)
Flux
Wavelength (Angstrom)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
5000
7500
10000
12500
15000
17500
20000
22500
25000
OIII_5007
SII_6731
H_beta (z)
OIII_5007 (z)
SII_6731 (z)

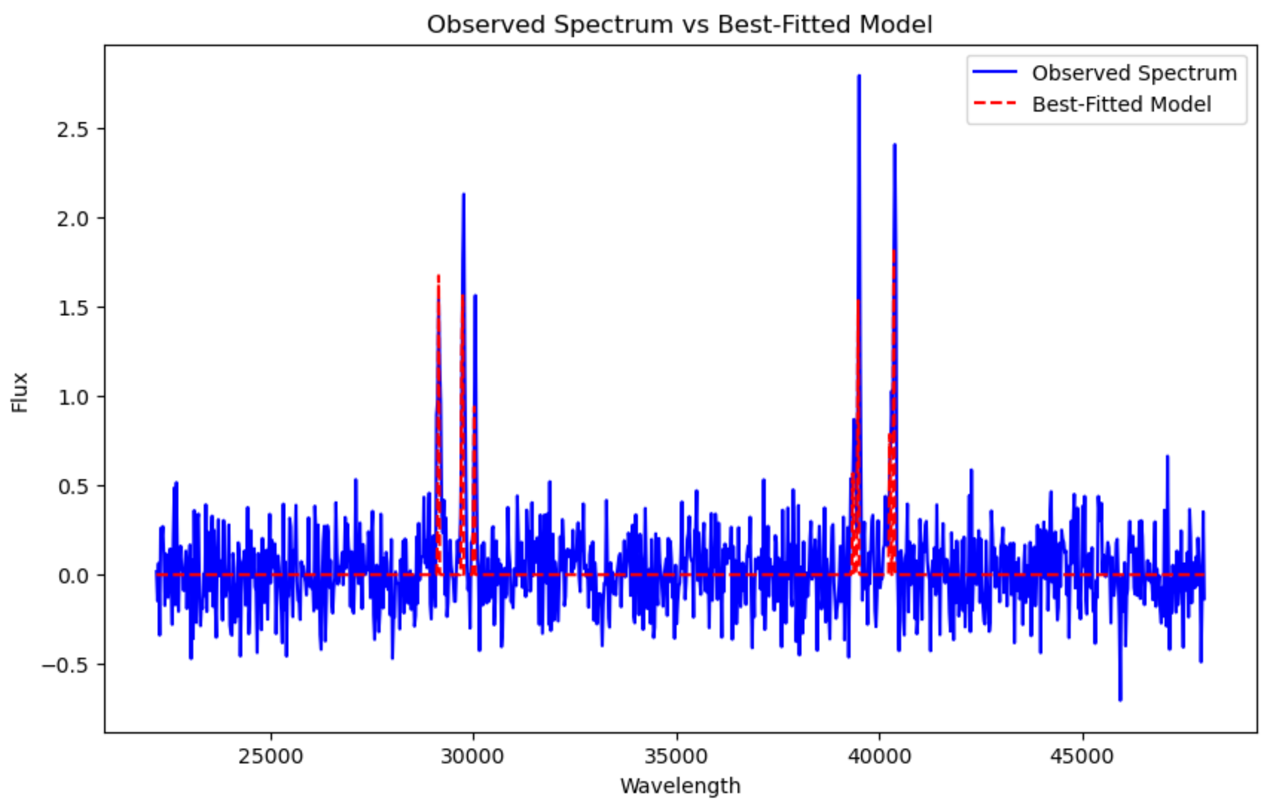

Estimated vs Actual Redshift
Estimated Redshift
Actual Redshift
Estimated Redshift
Input Redshift
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



I found that it's a mistake on the boundary condition (redshift range), since I set the estimated redshift to be 0 to 1, it only finds the best fit within this range. Also, the number of iteration is crucial as it gets more precise reading of the redshift. So I set the boundary from 0 to 10, which covers the most part of the redshift of AGN galaxy and changed the number of iteration to 1000. (This makes the code to run much longer, but gives much more accurate estimation).

In [43]:
```
# with this new setting, I was able to predict redshift with a large number
observed_wavelengths, observed_flux = generate_redshifted_spectrum(5)
emission_line_centers = [6563, 4861, 5007, 4959, 6584, 6716, 6731]
line_width = 10

def model_spectrum(wavelengths, redshift, line_intensities):
    model_flux = wavelengths ** -1.5
    for i, line_center in enumerate(emission_line_centers):
        shifted_center = line_center * (1 + redshift)
        model_flux += line_intensities[i] * norm.pdf(wavelengths, shifted_ce
    return model_flux

def chi_squared(obs_flux, model_flux):
    return np.sum((obs_flux - model_flux) ** 2)

def fit_spectrum(line_intensities, redshift):
    model_flux = model_spectrum(observed_wavelengths, redshift, line_intensi
    return chi_squared(observed_flux, model_flux)


N = len(emission_line_centers)  # number of emission lines
initial_line_intensities = [10] * N  # Initial guess for each line intensity

# Optimization Model
best_fit = None
for redshift in np.linspace(0, 10, 1000):
    result = minimize(fit_spectrum, initial_line_intensities, args=(redshift
    if best_fit is None or result.fun < best_fit[1]:
        best_fit = (redshift, result.fun, result.x)

best_fit_flux = model_spectrum(observed_wavelengths, best_fit[0], best_fit[2

plt.figure(figsize=(10, 6))
plt.plot(observed_wavelengths, observed_flux, label="Observed Spectrum", col
plt.plot(observed_wavelengths, best_fit_flux, label="Best-Fitted Model", col
plt.xlabel("Wavelength")
plt.ylabel("Flux")
plt.title("Observed Spectrum vs Best-Fitted Model")
plt.legend()
plt.show()

print(f"Best fit redshift: {best_fit[0]}")
print(f"Best fit line intensities: {best_fit[2]}")
```



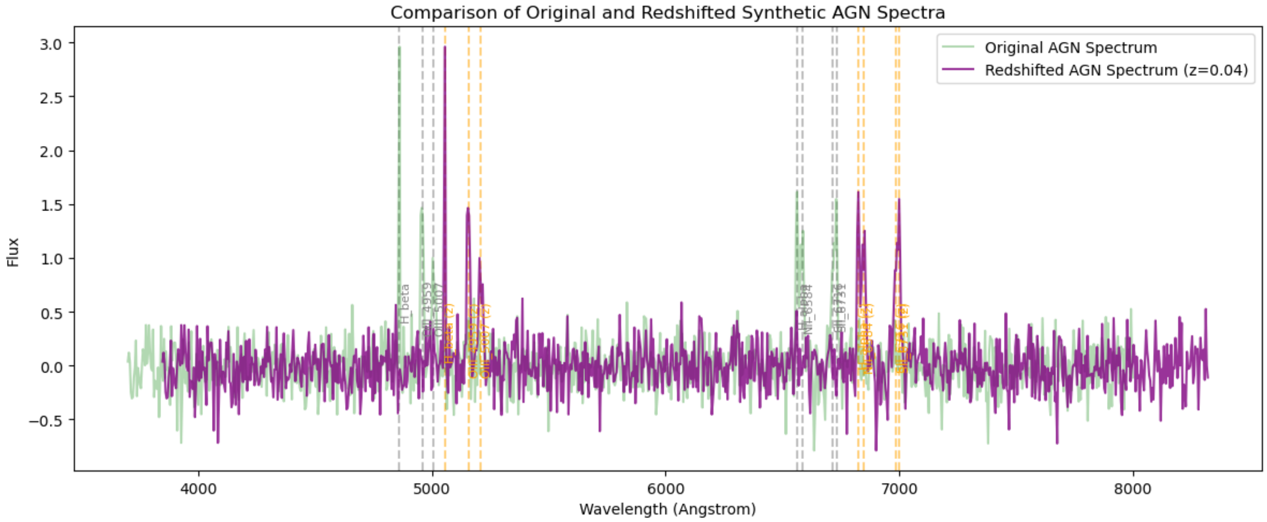


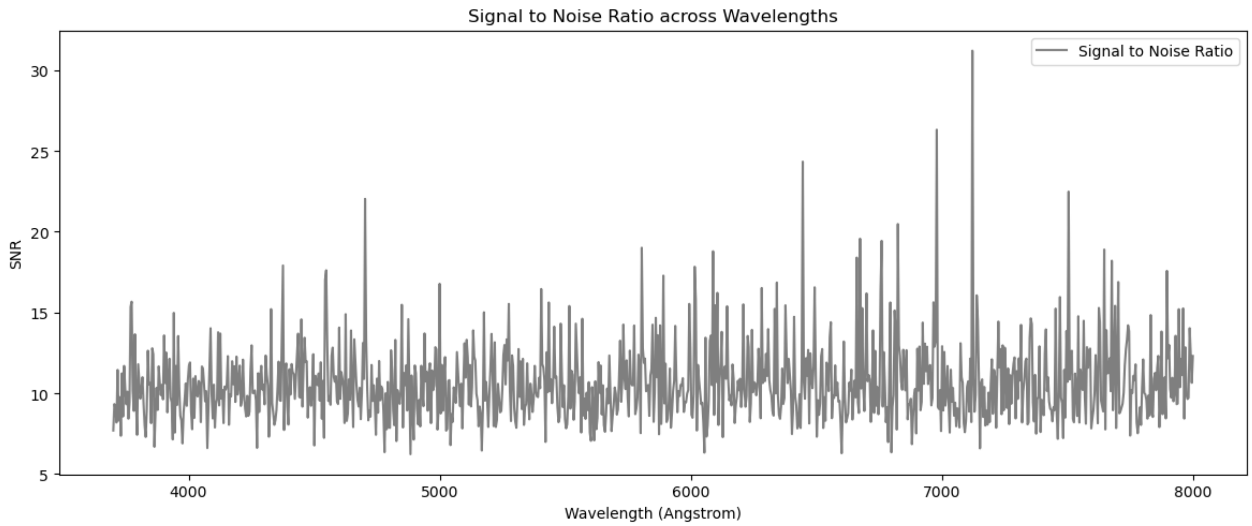

```
Best fit redshift: 4.994994994994995
Best fit line intensities: [14.94898065 48.70501321 33.17850489 80.04403446
47.2735389  32.88377089
 48.29315571]
```

This allows the predicted redshift to be as accurate as 0.01. Another important part of real spectrum is the noise, which is the error produced by the instrument or atmosphere, we can add that in our code to generate an error array.

In [47]:
```
def generate_redshifted_spectrum(redshift):
    wavelengths = np.linspace(3700, 8000, 1000)
    alpha = -15
    flux_continuum = wavelengths ** alpha

    # emission lines
    emission_lines = {
        "H_alpha": {"center": 6563, "element": "H"},
        "H_beta": {"center": 4861, "element": "H"},
        "OIII_5007": {"center": 5007, "element": "O"},
        "OIII_4959": {"center": 4959, "element": "O"},
        "NII_6584": {"center": 6584, "element": "N"},
        "SII_6716": {"center": 6716, "element": "S"},
        "SII_6731": {"center": 6731, "element": "S"}
    }

    # Function to add emission line
    def add_emission_line(wavelength_center, width, height):
        return height * norm.pdf(wavelengths, wavelength_center, width)

    # Adding emission lines
    width_range = (2, 10)
    base_height_range = (10, 20)
    for line in emission_lines.values():
        width = random.uniform(*width_range)
        height = random.uniform(*base_height_range)
        if line["element"] in ["H", "O", "N"]:
            height *= 1.5
        flux_continuum += add_emission_line(line["center"], width, height)

    # Adding noise
    noise_level = 0.2
    noise = np.random.normal(0, noise_level, wavelengths.shape)
    flux_continuum += noise

    # Error array representing uncertainty in flux measurements (by instrume
    error_array = np.random.normal(0.1, 0.02, wavelengths.shape) * flux_cont

    # Signal to Noise Ratio (SNR)
    snr = flux_continuum / error_array

    original_flux_continuum = flux_continuum.copy()
    redshifted_wavelengths = wavelengths * (1 + redshift)
    redshifted_flux_continuum = original_flux_continuum

    plt.figure(figsize=(12, 10))
    plt.subplot(2, 1, 1) # Upper plot for spectra
```

```
    for line_name, line in emission_lines.items():
        original_center = line["center"]
        redshifted_center = original_center * (1 + redshift)
        plt.axvline(x=original_center, color='gray', linestyle='--', alpha=0
        plt.axvline(x=redshifted_center, color='orange', linestyle='--', alp
        plt.text(original_center, plt.ylim()[1]*0.8, line_name, rotation=90,
        plt.text(redshifted_center, plt.ylim()[1]*0.6, f'{line_name} (z)', r

    plt.plot(wavelengths, original_flux_continuum, label='Original AGN Spect
    plt.plot(redshifted_wavelengths, redshifted_flux_continuum, label=f'Reds
    plt.xlabel('Wavelength (Angstrom)')
    plt.ylabel('Flux')
    plt.title('Comparison of Original and Redshifted Synthetic AGN Spectra')
    plt.legend()

    plt.subplot(2, 1, 2) # Lower plot for SNR
    plt.plot(wavelengths, snr, label='Signal to Noise Ratio', color='black',
    plt.xlabel('Wavelength (Angstrom)')
    plt.ylabel('SNR')
    plt.title('Signal to Noise Ratio across Wavelengths')
    plt.legend()

    plt.tight_layout()
    plt.show()

    return redshifted_wavelengths, flux_continuum, error_array
observed_wavelength, observed_flux, error = generate_redshifted_spectrum(0.0
```





In [80]:
```
# Update the galaxy class, now I modified it according to what we covered in
class Galaxy:
    '''initialize the galaxy'''
    def __init__(self, mass, position, velocity, black_hole_mass, initial_te
        self.mass = mass
        self.position = position
        self.velocity = velocity
        self.black_hole_mass = black_hole_mass
        self.initial_temperature = initial_temperature
        self.redshift = redshift

    '''black hole accretion model'''
    def calculate_accretion_rate(self, rho = np.random.uniform(1e-14, 1e-13)
        black_hole_mass = self.black_hole_mass
        # The input black hole mass should in the unit of solar mass
        G = 6.67430e-11 * u.m**3 / u.kg / u.s**2 # gravitational constant wi
        M = black_hole_mass * 1.98847e30 * u.kg  # solar masses to kg
        rho = rho * u.kg / u.m**3  # density of the accreting material in kg
        v = v * u.m / u.s  # relative velocity in m/s
        c_s = c_s * u.m / u.s # sound speed in the material in m/s
        # Bondi-Hoyle-Lyttleton accretion rate
```

```
        accretion_rate = (4 * np.pi * G**2 * M**2 * rho) / ((v**2 + c_s**2)*
        # Convert accretion rate to solar masses per year
        accretion_rate_solar_masses_per_year = accretion_rate.to(u.kg / u.s)
        return accretion_rate_solar_masses_per_year

    def calculate_radiation(self, accretion_rate, epsilon = 0.1):
        # The accretion rate is obtained from the previous function calculat
        c = 299792458 * u.m / u.s
        # calculate the luminosity (or radiation)
        radiation = epsilon * accretion_rate * 1.98847e30 * c**2  # Convert
        return radiation

    def accretion_model(self, accretion_radius, other_parameters):
        black_hole_mass = self.black_hole_mass
        accretion_rate = self.calculate_accretion_rate(black_hole_mass, accr
        radiation = self.calculate_radiation(accretion_rate, other_parameter
        jets = self.determine_jet_production(accretion_rate, radiation, othe
        return accretion_rate, radiation, jets

    '''cooling and heating process in different radius of the galaxy'''
    def heating_function(self, r, time, agn_luminosity_initial = 1e44, star_
        decay_factor = 0.05
        star_formation_rate = star_formation_initial / (1 + decay_factor * t
        agn_luminosity = agn_luminosity_initial / (1 + decay_factor * time)
        density = self.density_radius(r, rho_0, r_0, alpha)
        new_temp = 1e-35 * (star_formation_rate + agn_luminosity) * density
        return new_temp

    def cooling_function(self, temperature, r, rho_0 = 1e5, r_0 = 1, alpha =
        density = self.density_radius(r, rho_0, r_0, alpha)
        if temperature < 1e4:
            new_temp = 1e-22 * temperature * density**2
        else:
            new_temp = 1e-24 * temperature**1.5 * density**2
        return new_temp

    def cooling_rate(self, new_temp, density):
        cooling_rate = new_temp * density**2
        return cooling_rate

    def ism_temperature_evolution(self, initial_temperature, r, metallicity,
        temperature = initial_temperature
        density = self.density_radius(r, rho_0, r_0, alpha)
        for time in range(total_time):
            # cooling and heating rates
            cooling_rate = self.cooling_function(temperature, r, rho_0 = 1e5
            heating_rate = self.heating_function(star_formation_rate, agn_lu
            temperature_change = (heating_rate - cooling_rate) * timestep
            temperature += temperature_change
            if temperature < 10:  # 10 K is a reasonable lower limit for ISM
                temperature = 10
        return temperature
```

```
    def ism_temperature_evolution_over_time(self, initial_temperature, r, rh
        temperature = initial_temperature
        temperatures, cooling_rates, heating_rates = [], [], []
        for time in range(total_time):
            density = self.density_radius(r, rho_0, r_0, alpha)
            cooling_rate = self.cooling_function(temperature, r, rho_0, r_0,
            heating_rate = self.heating_function(star_formation_initial, agn
            temperature_change = (heating_rate - cooling_rate) * timestep
            temperature += temperature_change
            if temperature < 10:
                temperature = 10
            temperatures.append(temperature)
            cooling_rates.append(cooling_rate)
            heating_rates.append(heating_rate)
        return temperatures, cooling_rates, heating_rates

    '''spectra modeling (with noise and measurement error included)'''
    def generate_redshifted_spectrum(self, plot_spectrum = True):
        redshift = self.redshift
        wavelengths = np.linspace(3700, 8000, 1000)
        alpha = -15
        flux_continuum = wavelengths ** alpha

        # emission lines
        emission_lines = {
            "H_alpha": {"center": 6563, "element": "H"},
            "H_beta": {"center": 4861, "element": "H"},
            "OIII_5007": {"center": 5007, "element": "O"},
            "OIII_4959": {"center": 4959, "element": "O"},
            "NII_6584": {"center": 6584, "element": "N"},
            "SII_6716": {"center": 6716, "element": "S"},
            "SII_6731": {"center": 6731, "element": "S"}
        }

        # Function to add emission line
        def add_emission_line(wavelength_center, width, height):
            return height * norm.pdf(wavelengths, wavelength_center, width)

        # Adding emission lines
        width_range = (2, 10)
        base_height_range = (10, 20)
        for line in emission_lines.values():
            width = random.uniform(*width_range)
            height = random.uniform(*base_height_range)
            if line["element"] in ["H", "O", "N"]:
                height *= 1.5
            flux_continuum += add_emission_line(line["center"], width, heigh

        # Adding noise
        noise_level = 0.2
        noise = np.random.normal(0, noise_level, wavelengths.shape)
        flux_continuum += noise
```

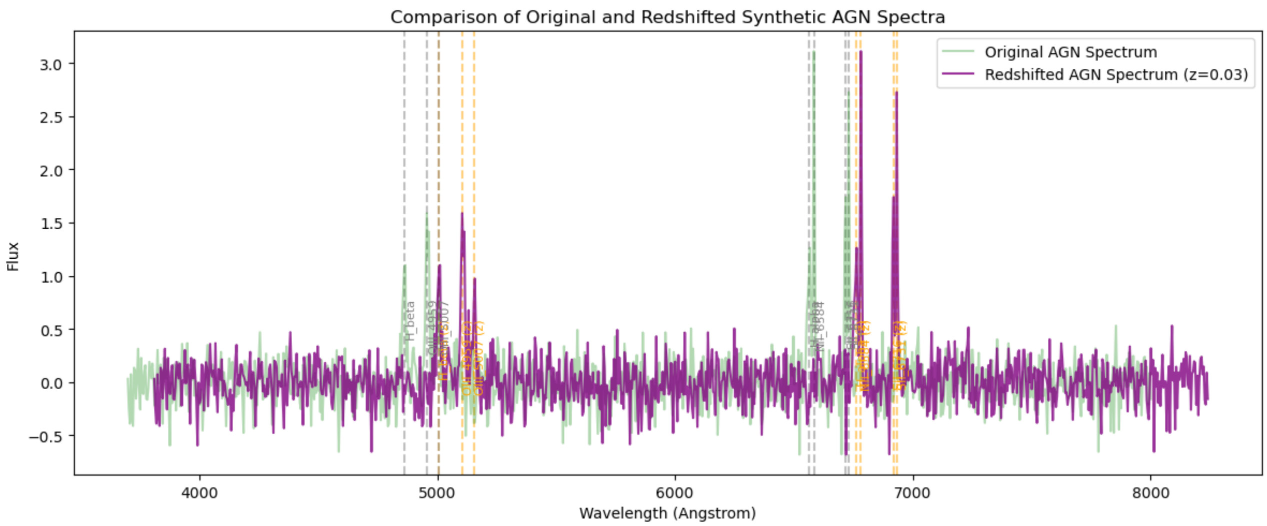
```
        # Error array representing uncertainty in flux measurements (by inst
        error_array = np.random.normal(0.1, 0.02, wavelengths.shape) * flux_

        # Signal to Noise Ratio (SNR)
        snr = flux_continuum / error_array

        original_flux_continuum = flux_continuum.copy()
        redshifted_wavelengths = wavelengths * (1 + redshift)
        redshifted_flux_continuum = original_flux_continuum
        if plot_spectrum:
            plt.figure(figsize=(12, 10))
            plt.subplot(2, 1, 1) # Upper plot for spectra
            for line_name, line in emission_lines.items():
                original_center = line["center"]
                redshifted_center = original_center * (1 + redshift)
                plt.axvline(x=original_center, color='gray', linestyle='--',
                plt.axvline(x=redshifted_center, color='orange', linestyle='
                plt.text(original_center, plt.ylim()[1]*0.8, line_name, rota
                plt.text(redshifted_center, plt.ylim()[1]*0.6, f'{line_name}

            plt.plot(wavelengths, original_flux_continuum, label='Original A
            plt.plot(redshifted_wavelengths, redshifted_flux_continuum, labe
            plt.xlabel('Wavelength (Angstrom)')
            plt.ylabel('Flux')
            plt.title('Comparison of Original and Redshifted Synthetic AGN S
            plt.legend()

            plt.subplot(2, 1, 2) # Lower plot for SNR
            plt.plot(wavelengths, snr, label='Signal to Noise Ratio', color=
            plt.xlabel('Wavelength (Angstrom)')
            plt.ylabel('SNR')
            plt.title('Signal to Noise Ratio across Wavelengths')
            plt.legend()

            plt.tight_layout()
            plt.show()

        return redshifted_wavelengths, flux_continuum, error_array
```

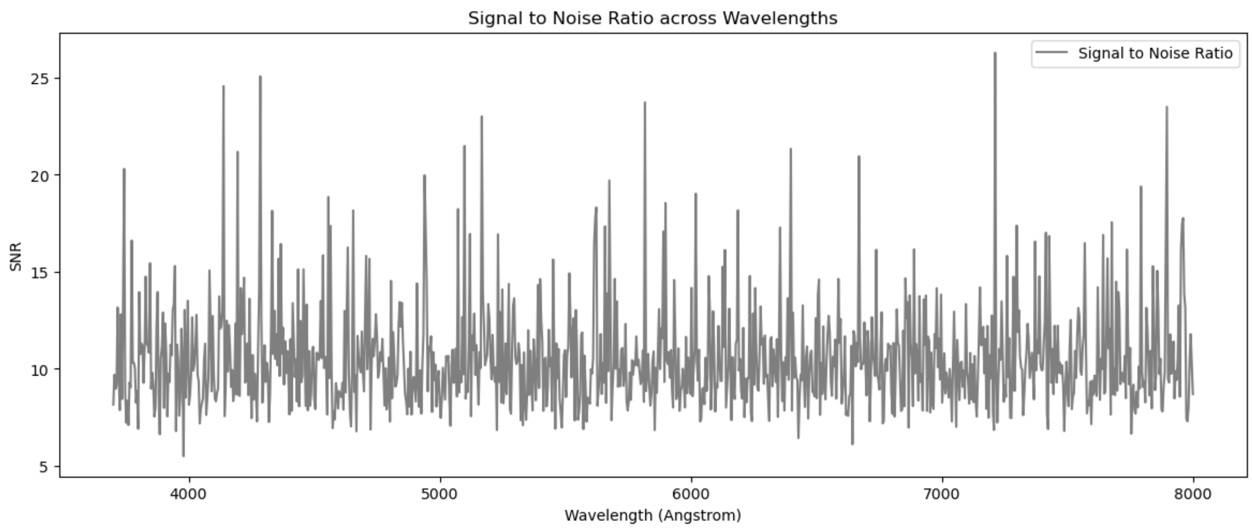
In [81]:

```
# Now, I can randomly generate a AGN galaxy using the "Galaxy" Class
mass = 1e12  # Galaxy mass in solar masses
position = np.array([1e6, 2e6, 3e6])  # Galaxy position in light years (x, y
velocity = np.array([1000, 2000, 1500])  # Galaxy velocity in km/s (velocity
black_hole_mass = 1e9
initial_temperature = 1e4
redshift = 0.03
AGN_gal1 = Galaxy(mass, position, velocity, black_hole_mass, initial_tempera
```

In [82]:

```
# I can plot the spectra of this galaxy and find the wavelength, flux, and e
redshifted_wavelengths, flux_continuum, error_array = AGN_gal1.generate_reds
```





In [83]:
```
def calculate_line_ratio(intensity1, intensity2):
    return intensity1 / intensity2

def estimate_sfr_from_Halpha(L_Halpha):
    # Kennicutt (1998) relation: SFR = 7.9e-42 * L(H-alpha)
    return 7.9e-42 * L_Halpha
```

In [84]:
```
# the intensity of H_alpha and H_beta lines can be calculated from:
H_alpha_flux = flux_continuum[np.argmin(np.abs(redshifted_wavelengths - 6563
H_beta_flux = flux_continuum[np.argmin(np.abs(redshifted_wavelengths - 4861
# then the H_alpha/H_beta ratio is
H_alpha_H_beta_ratio = calculate_line_ratio(H_alpha_flux, H_beta_flux)
print('H_alpha/H_beta =', H_alpha_H_beta_ratio )
# and calculate the star formation rate
SFR = estimate_sfr_from_Halpha(H_alpha_flux)
print("Star Formation Rate:", SFR, "Solar Masses per Year")
```

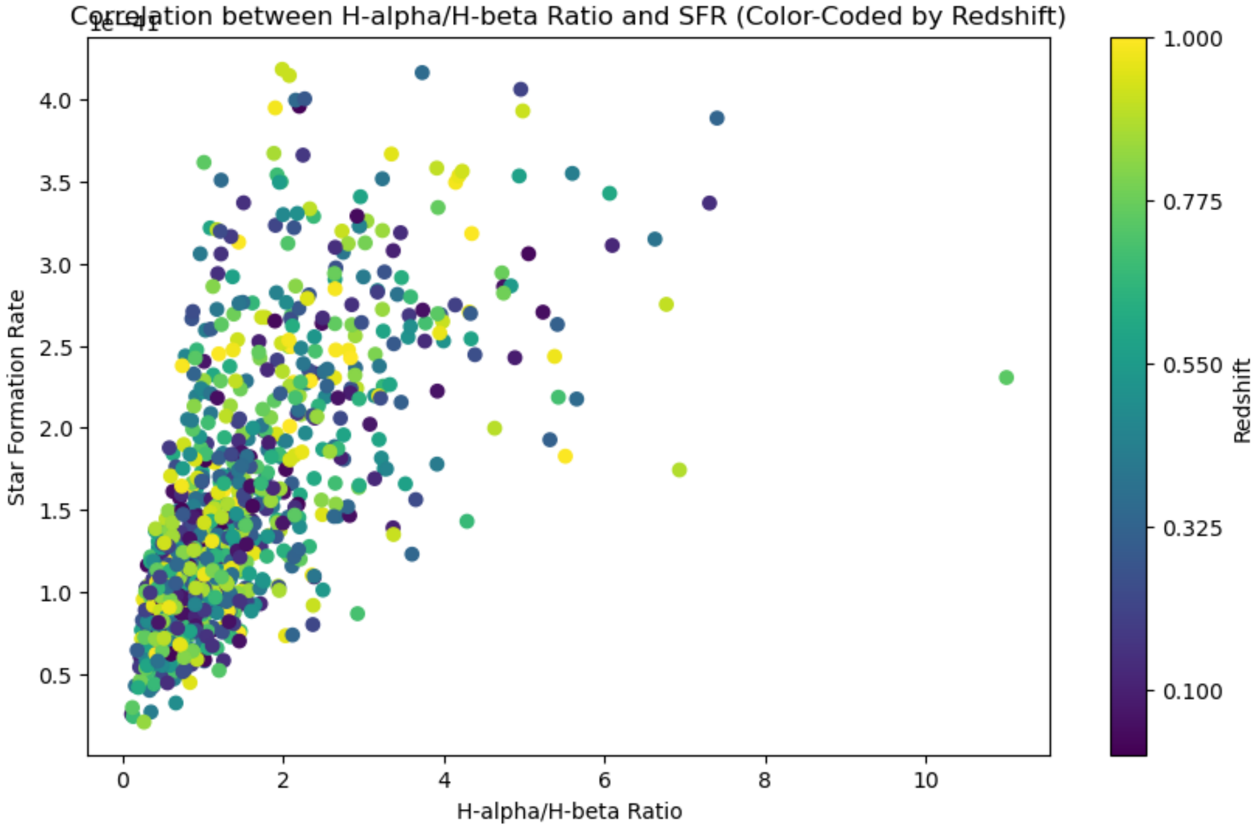
```
H_alpha/H_beta = 0.894918435893807
Star Formation Rate: 7.627137991139248e-42 Solar Masses per Year
```

In [107…
```
# To observe the correlation between H_alpha/H_beta ratio and Star formation
```

```
# I can model 100 such galaxies using the galaxy class, and plot the correla
def simulate_galaxy_and_calculate_properties():
    redshift = np.random.uniform(0.01, 1)
    # initialize galaxy
    galaxy = Galaxy(mass=np.random.uniform(1e11, 1e13),
                    position=np.random.rand(3),
                    velocity=np.random.rand(3),
                    black_hole_mass=np.random.uniform(1e7, 1e9),
                    initial_temperature=np.random.uniform(1e4, 1e5),
                    redshift=redshift)
    r = redshift

    redshifted_wavelengths, flux_continuum, _ = galaxy.generate_redshifted_s

    # H_alpha and H_beta
    H_alpha_flux = flux_continuum[np.argmin(np.abs(redshifted_wavelengths - 
    H_beta_flux = flux_continuum[np.argmin(np.abs(redshifted_wavelengths - 4

    # line ratio and SFR
    line_ratio = calculate_line_ratio(H_alpha_flux, H_beta_flux)
    sfr = estimate_sfr_from_Halpha(H_alpha_flux)

    return line_ratio, sfr, r


line_ratios = []
sfr_values = []
redshifts = []

for _ in range(1000):
    ratio, sfr, r = simulate_galaxy_and_calculate_properties()
    line_ratios.append(ratio)
    sfr_values.append(sfr)
    redshifts.append(r)

# print(redshifts)
line_ratios = np.array(line_ratios)
sfr_values = np.array(sfr_values)
redshifts = np.array(redshifts)

correlation = np.corrcoef(line_ratios, sfr_values)[0, 1]
print("Correlation between H-alpha/H-beta ratio and SFR:", correlation)
plt.figure(figsize=(10, 6))
scatter = plt.scatter(line_ratios, sfr_values, c=redshifts, cmap='viridis')
plt.xlabel('H-alpha/H-beta Ratio')
plt.ylabel('Star Formation Rate')
plt.title('Correlation between H-alpha/H-beta Ratio and SFR (Color-Coded by

# Adding a color bar
cbar = plt.colorbar(scatter)
cbar.set_label('Redshift')
cbar.set_ticks(np.linspace(0.1, 1, 5))
```

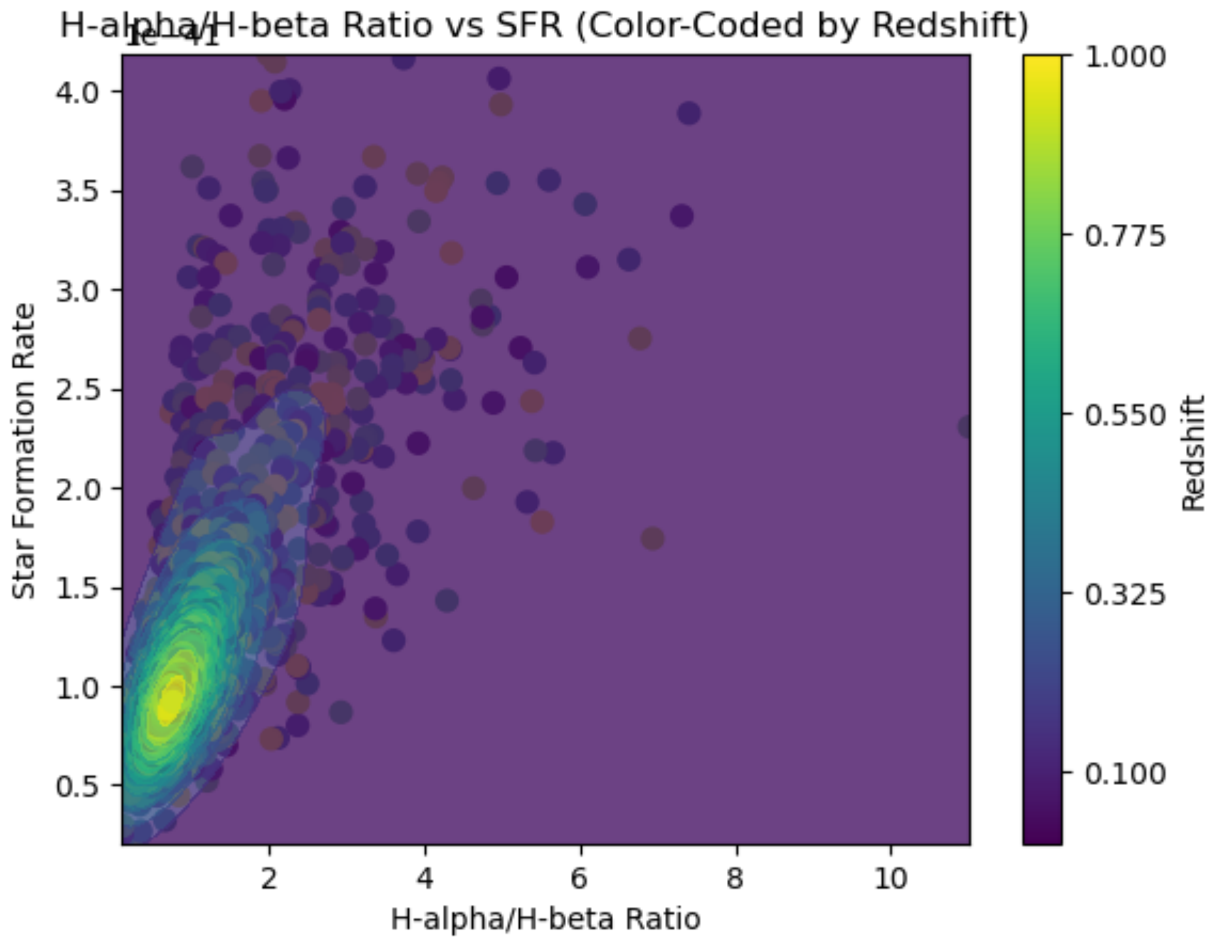
```
plt.show()
```

```
Correlation between H-alpha/H-beta ratio and SFR: 0.6704983723809521
```



In [112…

```
# I can use the KDE plot to present the density distribution of the H_alpha/
# 2D Gaussian KDE
xy = np.vstack([line_ratios, sfr_values])
z = gaussian_kde(xy)(xy)

# sort the points by density for the scatter plot
idx = z.argsort()
line_ratios, sfr_values, z, redshifts = line_ratios[idx], sfr_values[idx], z

xmin, xmax = line_ratios.min(), line_ratios.max()
ymin, ymax = sfr_values.min(), sfr_values.max()
xx, yy = np.meshgrid(np.linspace(xmin, xmax, 100), np.linspace(ymin, ymax, 1

# evaluate the KDE on the grid
zz = gaussian_kde(xy)(np.vstack([xx.ravel(), yy.ravel()])).reshape(xx.shape)

fig, ax = plt.subplots()
scatter = ax.scatter(line_ratios, sfr_values, c=redshifts, s=50, cmap='virid
ax.contourf(xx, yy, zz, levels=10, alpha=0.8, cmap='viridis')  # Adjust leve

ax.set_xlabel('H-alpha/H-beta Ratio')
ax.set_ylabel('Star Formation Rate')
plt.title('H-alpha/H-beta Ratio vs SFR (Color-Coded by Redshift)')

cbar = plt.colorbar(scatter)
cbar.set_label('Redshift')
cbar.set_ticks(np.linspace(0.1, 1, 5))
plt.show()
```



This correlation plot and the calculated correlation above shows that the H_alpha/H_beta ratio and the star formation rate of the AGN galaxies are closely correlated (correlation 0.6 in astrophysics is already considered a good correlation).

Even though the redshift is randomly generated, we can see the points are scattered at low H_alpha/H_beta ratio, and low star formation rate. Also, star formation rate increases as the H_alpha/H_beta ratio increases, and the points distribute more scattered at high H_alpha/H_beta ratio and high star formation rate.

Implications

There are much more to consider about the complete model of AGN galaxy, including things hard for me to analyze such as the fluid dynamics or each element's properties. Thus I can only model the most simple and straightforward spectra. However, other people have developed the computation codes that models the spectra very well over decades.

With the observed spectra, we are able to model the galaxy spectra, calculate its redshift, and find the correlation between parameters of a galaxy.

For example, ALF is a code developed by Conroy (Conroy 2014) that models a galaxy's spectrum with parameters single age, [Z/H], velocity, and velocity dispersion, and the abundances of the elements C, N, O, Mg, Si, Ca, Ti, and Na.

After putting in the observed spectrum and running the simulation on a HPC for Nburn = 20000 and Nwalker = 512, I got the resulting .bestspec, .mcmc, and .sum file.

The .bestspec contains the best fitted model, .mcmc is the monte carlo simulation result, and .sum contains the statistical result of the modeling.

I here define some functions to read and interpreate the data.

In [136…

```
# used the code I wrote during the summer
import sys
import warnings
from copy import deepcopy
import numpy as np
from numpy.polynomial.chebyshev import chebfit, chebval
from scipy import constants, interpolate
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from astropy.io import ascii
from astropy.table import Table, Column, hstack
import os
import matplotlib.pyplot as plt
import numpy as np
from astropy.table import Table
from astropy.io import ascii
class analyze_spectra(object):
    def __init__(self, outfiles, read_mcmc=True, info=None, index=False):
        self.outfiles = outfiles
        self.nsample = None
        self.spectra = None
        if read_mcmc:
```

```
            self.mcmc = np.loadtxt('{0}.mcmc'.format(self.outfiles))
        results = ascii.read('{0}.sum'.format(self.outfiles))
        self.labels = np.array([
                    'chi2', 'velz', 'sigma', 'logage', 'zH', 'FeH', 'a',
                    'C', 'N', 'Na', 'Mg', 'Si', 'K', 'Ca', 'Ti', 'V', 'Cr',
                    'Mn', 'Co', 'Ni', 'Cu', 'Sr', 'Ba', 'Eu', 'Teff',
                    'IMF1', 'IMF2', 'logfy', 'sigma2', 'velz2', 'logm7g',
                    'hotteff', 'loghot', 'fy_logage', 'logemline_h',
                    'logemline_oii', 'logemline_oiii', 'logemline_sii',
                    'logemline_ni', 'logemline_nii', 'logtrans', 'jitter',
                    'logsky', 'IMF3', 'IMF4', 'h3', 'h4',
                    'ML_v','ML_i','ML_k','MW_v', 'MW_i','MW_k'
                    ])
        results = Table(results, names=self.labels)
        types = Column(['mean', 'chi2', 'error',
                          'cl25', 'cl16', 'cl50',
                          'cl84', 'cl98', 'lo_prior',
                          'hi_prior'],
                          name='Type')
        results.add_column(types, index=0)
        self.results = results['Type', 'chi2', 'velz', 'sigma', 'logage', 'z
                    'C', 'N', 'Na', 'Mg', 'Si', 'K', 'Ca', 'Ti', 'V', 'Cr',
                    'Mn', 'Co', 'Ni', 'Cu', 'Sr', 'Ba', 'Eu', 'Teff',
                    'IMF1', 'IMF2', 'logfy', 'sigma2', 'velz2', 'logm7g',
                    'hotteff', 'loghot', 'fy_logage', 'logemline_h',
                    'logemline_oii', 'logemline_oiii', 'logemline_sii',
                    'logemline_ni', 'logemline_nii', 'logtrans', 'jitter',
                    'logsky', 'IMF3', 'IMF4', 'h3', 'h4',
                    'ML_v','ML_i','ML_k','MW_v', 'MW_i','MW_k' ]
        my_df = Table(self.mcmc, names = self.labels)
        self.my_df = my_df['chi2', 'velz', 'sigma', 'logage', 'zH', 'FeH', '
                    'C', 'N', 'Na', 'Mg', 'Si', 'K', 'Ca', 'Ti', 'V', 'Cr',
                    'Mn', 'Co', 'Ni', 'Cu', 'Sr', 'Ba', 'Eu', 'Teff',
                    'IMF1', 'IMF2', 'logfy', 'sigma2', 'velz2', 'logm7g',
                    'hotteff', 'loghot', 'fy_logage', 'logemline_h',
                    'logemline_oii', 'logemline_oiii', 'logemline_sii',
                    'logemline_ni', 'logemline_nii', 'logtrans', 'jitter',
                    'logsky', 'IMF3', 'IMF4', 'h3', 'h4',
                    'ML_v','ML_i','ML_k','MW_v', 'MW_i','MW_k' ]
        with open('{0}.sum'.format(self.outfiles)) as f:
            for line in f:
                if line[0] == '#':
                    if 'Nwalkers' in line:
                        self.nwalkers = float(line.split('=')[1].strip())
                    elif 'Nchain' in line:
                        self.nchain = float(line.split('=')[1].strip())
                    elif 'Nsample' in line:
                        self.nsample = float(line.split('=')[1].strip())
        MCMC = ascii.read('{0}.mcmc'.format(self.outfiles))
        m = np.loadtxt('{0}.bestspec'.format(self.outfiles))
        data = {}
        data['wave'] = m[:,0]/(1.+self.results['velz'][5]*1e3/constants.c)
        data['m_flux'] = m[:,1] # Model spectrum, normalization applied
```

```
        data['d_flux'] = m[:,2] # Data spectrum
        data['snr'] = m[:,3]  # Including jitter and inflated errors
        data['unc'] = 1/m[:,3]
        if not index:
            data['poly'] = m[:,4] # Polynomial used to create m_flux
        data['residual'] = (m[:,1] - m[:,2])/m[:,1] * 1e2
        self.spectra = data

    def normalize_spectra(self):
        """
        Normalize the data and model spectra
        """
        self.spectra['m_flux_norm'] = deepcopy(self.spectra['m_flux'])
        self.spectra['d_flux_norm'] = deepcopy(self.spectra['d_flux'])
        self.spectra['unc_norm']    = deepcopy(self.spectra['unc'])

        chunks = 1000
        min_ = min(self.spectra['wave'])
        max_ = max(self.spectra['wave'])
        num  = int((max_ - min_)/chunks) + 1

        for i in range(num):
            k = ((self.spectra['wave'] >= min_ + chunks*i) &
                 (self.spectra['wave'] <= min_ + chunks*(i+1)))

            if len(self.spectra['d_flux_norm'][k]) < 10:
                continue

            coeffs = chebfit(self.spectra['wave'][k],
                             self.spectra['d_flux_norm'][k], 2)
            poly = chebval(self.spectra['wave'][k], coeffs)
            self.spectra['d_flux_norm'][k] = self.spectra['d_flux_norm'][k]/
            self.spectra['unc_norm'][k] = self.spectra['unc_norm'][k]/poly

            coeffs = chebfit(self.spectra['wave'][k],
                             self.spectra['m_flux_norm'][k], 2)
            poly = chebval(self.spectra['wave'][k], coeffs)
            self.spectra['m_flux_norm'][k] = self.spectra['m_flux_norm'][k]/

    def plot_model(self, fname):
        chunks = 5000
        min_ = min(self.spectra['wave'])
        max_ = max(self.spectra['wave'])
        num = int((max_ - min_)/chunks) + 1

        with PdfPages(fname) as pdf:
            for i in range(num):
                fig = plt.figure(figsize=(14,9), facecolor='white')
                ax1 = plt.subplot2grid((3,2), (0,0), rowspan=2, colspan=2)
                ax2 = plt.subplot2grid((3,2), (2,0), rowspan=1, colspan=2)

                j = ((self.spectra['wave'] >= min_ + chunks*i) &
                     (self.spectra['wave'] <= min_ + chunks*(i+1)))
```

```
            ax1.plot(self.spectra['wave'][j],
                     self.spectra['d_flux_norm'][j],
                     color='gray', lw=2, alpha = 0.7, label='Data')

            ax1.plot(self.spectra['wave'][j],
                     self.spectra['m_flux_norm'][j],
                     color='red', lw=2, alpha = 0.8, label='Model')
            ax1.legend(frameon=False)

            ax2.plot(self.spectra['wave'][j],
                     self.spectra['residual'][j],
                     color='#7156A5', lw=2, alpha=0.7)

            ax2.fill_between(self.spectra['wave'][j],
                    -(self.spectra['unc'][j])*1e2,
                    +(self.spectra['unc'][j])*1e2,
                    color='#CCCCCC')

            ax2.set_ylim(-15.9, 15.9)
            #ax2.set_ylim(-4.9, 4.9)

            ax1.set_ylabel(r'Flux (arbitrary units)',
                           fontsize=22)
            ax2.set_ylabel(r'Residual $\rm \%$',
                           fontsize=22)

            ax2.set_xlabel(r'Wavelength $(\AA)$',
                           fontsize=22, labelpad=10)

            ax1.set_ylim(0.5, 1.5)

            plt.savefig(fname + "_" + str(i) + ".jpg", format='jpg')

            plt.show()


    def plot_corner_new(self, outname, save=True, xranges=None, plot_datapoi
        import corner
        import numpy as np
        import matplotlib.pyplot as plt

        # Assuming you have a DataFrame named my_df
        data = np.array([self.my_df['sigma'], self.my_df['zH'], self.my_df['

        # Create the corner plot without filled contours
        figure = corner.corner(data,
                               labels=['sigma', 'zH', 'logage', '[Mg/Fe]'],
                               show_titles=False,
                               color=plot_color,
                               fill_contours=True,
                               plot_datapoints=plot_datapoints
                              )
```

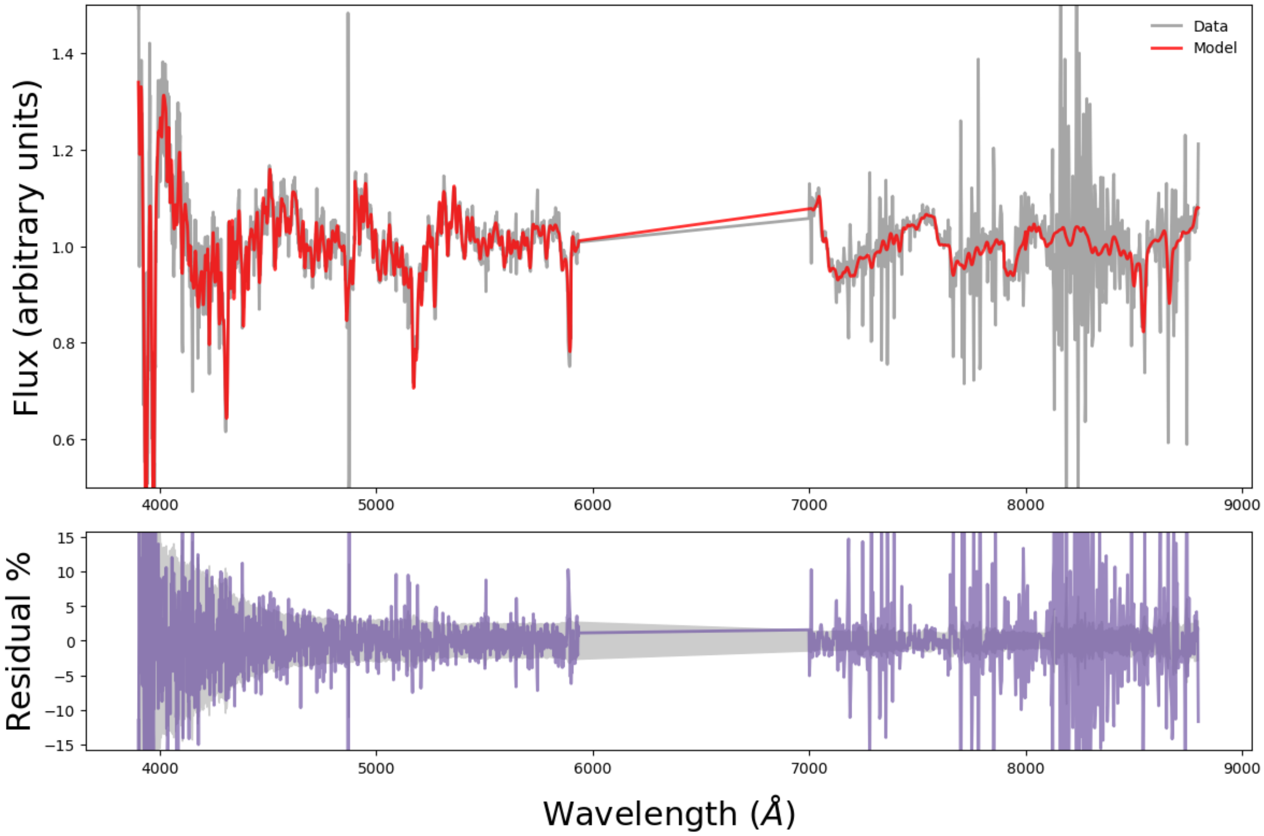
```
        # Access the axes
        axes = np.array(figure.axes).reshape((4, 4))

        # Loop over the diagonal
        for i in range(4):
            ax = axes[i, i]
            ax.axvline(np.median(data[:, i]), color="r")

        # Loop over the histograms
        for yi in range(4):
            for xi in range(yi):
                ax = axes[yi, xi]

                # Calculate medians
                median_x = np.median(data[:, xi])
                median_y = np.median(data[:, yi])

                # Calculate 1-sigma errors
                err_x = np.percentile(data[:, xi], [16, 84])
                err_y = np.percentile(data[:, yi], [16, 84])

                # Calculate differences for error bars
                xerr = [[median_x - err_x[0]], [err_x[1] - median_x]]
                yerr = [[median_y - err_y[0]], [err_y[1] - median_y]]

                # Plot error bars
                ax.errorbar(median_x, median_y, xerr=xerr, yerr=yerr, color=

                plt.tight_layout()
                if save:
                    plt.savefig(outname + '_corner')
```

In [137…

```
my_spec = analyze_spectra('shifted_combined_09000204_ob10_512_20k')
outfile = 'shifted_combined_09000204_ob10_512_20k'
```

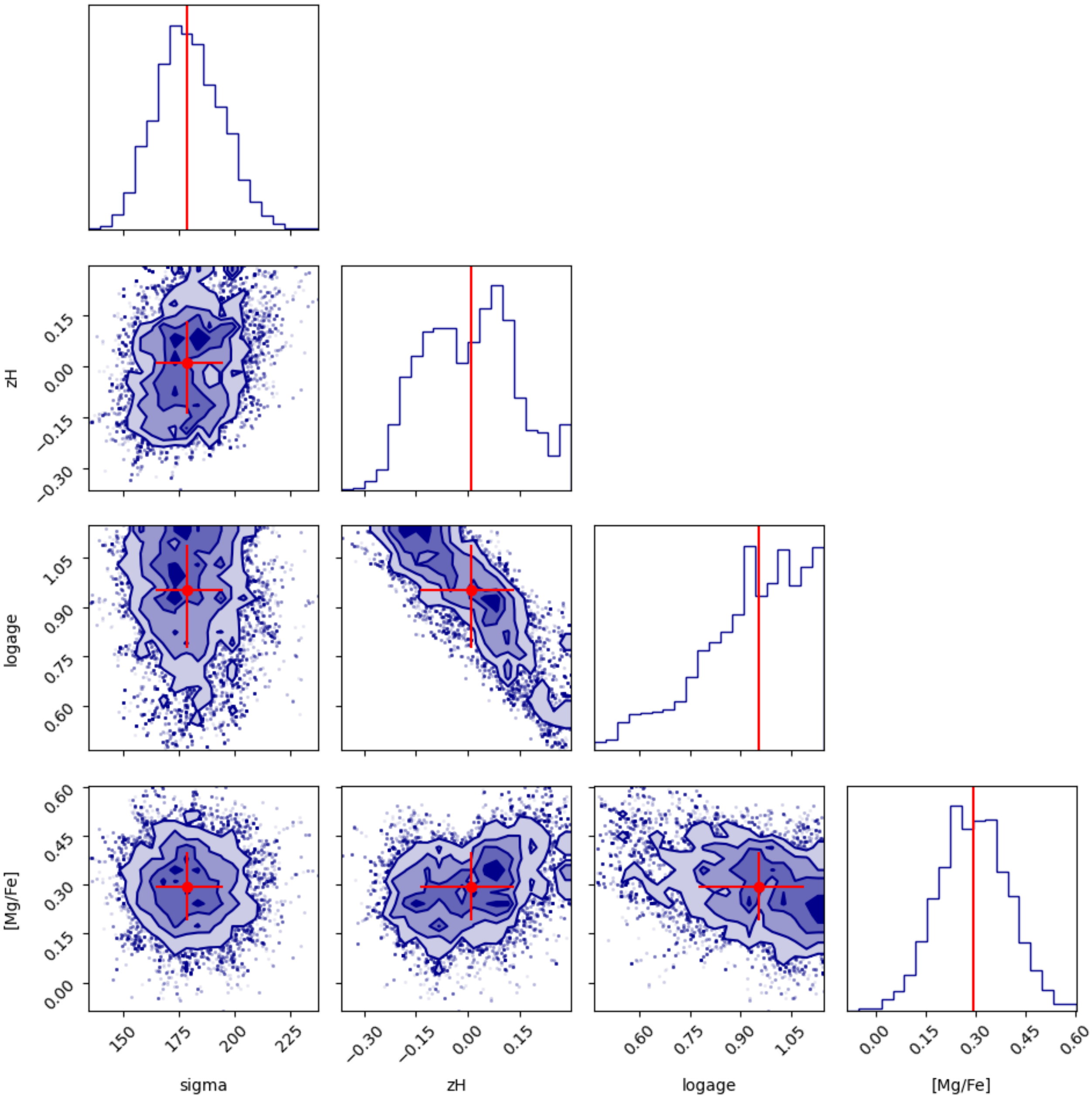
In [138…

```
my_spec.normalize_spectra()
my_spec.plot_model(outfile)
```



In [139...

```
my_spec.plot_corner_new(outfile)
```



Here we can find the correlation between these stellar populations

```
In [ ]:
```