```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  9 15:58:58 2020

@author: jimmy
"""
import numpy as np
import matplotlib.pyplot as plt
from MathTools import Gaussian, NonRelDoppler
from scipy.optimize import curve_fit
from astropy import units as u

class SpectralFeatures():

    # Initialize input parameters
    def __init__(self,wavelength,offset,P,mean,sigma,sampleInterval,SNR):
        # Inputs:
        #   P: peak depth of spectrum
        #   sigma: width of spectrum (in Angstroms)
        #   mean: center of spectrum (in Angstroms)
        #   sampleInterval: spacing of pixels (in Angstroms)
        #   SNR: signal-to-noise ratio you want to generate

        self.SNR = SNR
        self.wavelength = wavelength
        self.offset = offset
        self.P = P
        self.mean = mean
        self.sigma = sigma
        self.sample = sampleInterval

    def GaussianNoise(self,plot=True):
        # Returns:
        #   Plot of Gaussian spectrum with user-defined properties

        # Generate list of wavelengths to draw spectrum over
        x = self.wavelength+np.arange(self.mean-5.0,self.mean+5.0+self.sample,self.sample)

        # Calculate value of Gaussian function at each x
        function = Gaussian(x,self.offset,self.P,self.mean,self.sigma)

        # Calculate photometric precision from SNR (SNR=1/sigma_prec)
        sigma_prec = 1.0/self.SNR

        # Generate Gaussian noise depending on SNR
        noise = np.random.normal(0,sigma_prec,len(x))

        # Add Gaussian noise to Gaussian function values
        noiseData = np.add(function,noise)

        if plot == True:
            # Plot noisy data
            plt.scatter(x,noiseData)
            plt.xlabel(r'Wavelength ($\AA$)',fontsize=14)
            plt.title('Gaussian Spectrum with SNR={0}:1'.format(self.SNR),fontsize=18)
            plt.ylim(0,1.4)
            plt.tight_layout()

        return(x,noiseData)

    # Function to fit a Gaussian model to Gaussian with noise
    def GaussianModel(self,free=4,plot=False):
        # Inputs:
        #   init_val: array of initial guesses for free parameters
        #   free: number of free parameters (4=all,1=mean only free param.)
        #   plot: choose to plot the model vs. data or not
```

```python
        # Returns:
        #   best-fit parameters to model noisy Gaussian

        # Set 'free' as global parameter for later use
        self.free = free

        # Extract wavelength and Gaussian values from GaussianNoise
        x, y = self.GaussianNoise(plot=False)

        # Use curve_fit to find best parameters to fit model
        self.guesses = [0.5,0.25,0.2,0.5]

        # Fit different functions for different # of free parameters
        if free == 4:

            #  Use curve_fit to find best-fit parameters and their covariances
            best_vals, covar = curve_fit(Gaussian, x, y, p0=self.guesses)
            #print('best_vals: {}'.format(best_vals))

            # Calculate y values of model
            y_model = Gaussian(x,best_vals[0],best_vals[1],best_vals[2],best_vals[3])

            # Extract center of Gaussian from best-fit model parameters
            mean_model = best_vals[2]
            errors = np.abs(np.diag(covar))
            accuracy_model = np.sqrt(errors[2])

        elif free == 1: # fix all Gaussiain parameters but the mean
            custom_Gaussian = lambda x, mean: Gaussian(x,self.offset,self.P,mean,self.sigma)
            best_vals, covar = curve_fit(custom_Gaussian, x, y, p0=self.guesses[2])
            #print('best_vals: {}'.format(best_vals))

            # Calculate y values of model
            y_model = Gaussian(x,self.offset,self.P,best_vals[0],self.sigma)

            # Extract center of Gaussian from best-fit model parameters
            mean_model = best_vals[0]
            accuracy_model = np.sqrt(np.diag(covar))

        # Plot data vs. model
        if plot == True:
            # Plot data vs. model
            plt.scatter(x,y,label='Data',color='red')
            plt.plot(x,y_model,label='Model',color='blue')
            plt.legend()
            plt.xlabel(r'Wavelength ($\AA$)',fontsize=14)
            plt.title('Gaussian Spectrum with SNR={0}:1'.format(self.SNR),fontsize=18)
            plt.ylim(0,1.4)
            plt.tight_layout()

        # Calculate relative accuracy
        #rel_acc = 1-np.abs(mean_model)/self.wavelength*100
        return(accuracy_model)

# Function to calculate the accuracy of Gaussian model at partic. SNR
def Accuracy(free,signals,units='angs'):
    # Inputs:
    #   signals: array of SNRs
    #   free: number of free parameters (4=all,1=mean only free param.)
    # Returns:
    #   plot of accuracy of model vs. SNR

    # Empty list of accuracies and SNRs that give realistic errors
    accuracies = []
    best_signals = []

    i,j = 0,0
```

```python
# Loop over many SNR cases to find relationship between accuracy vs. SNR
for value in signals:
    j += 1
    # Create instance of class for particular SNR
    SNR_instance = SpectralFeatures(5000.0,1.0,0.5,0.0,1.0,0.5,value)

    # Calculate accuracy of model for particular SNR and add to list
    accuracy = SNR_instance.GaussianModel(free,plot=False)

    if accuracy <= 1.0:
        best_signals.append(value)
        accuracies.append(accuracy)
    else: # count number of outliers that give very high errors
        i += 1

# Tell user how many trials had excessive errors
print('{0} out of {1} trials had accuracy > 1'.format(i,j))

# Convert errors to km/s if requested
if units=='km/s':
    # Add errors to center wavelength and convert to km/s
    accuracies = [NonRelDoppler(x+5000.0) for x in accuracies]

    # Set units for y label
    unit_string = r'$\frac{km}{s}$'
else: # if units='angs'(default)
    # Set units for y label
    unit_string = r'$\AA$'

# Plot accuracy vs SNR.
plt.scatter(best_signals,accuracies,label='Median Error = {0:.2f} '.format(np.median(accurac
plt.xlabel('Signal-to-Noise Ratio',fontsize=14)
plt.ylabel(r'Error ({0})'.format(unit_string),fontsize=14)
plt.title('Accuracy of Model vs. SNR \n (Free parameters: {0})'.format(free),fontsize=18)
plt.legend()
```