Nebula Intensity Code:

GitHub Link: <https://github.com/Amani5576/Nebula_Intensity.git>

# Intensity Spectra of Nebulae

# S. Amani Njoroge

# 4060924

# FitsExtraction

from astropy.io import fits

import numpy as np

fitFiles = ["HA.fit", "OIII.fit", "SII.fit"]

#Dictionary for usage in getting the numpy data and its type

numpyDatTyp = { 8: "numpy.uint8 (note it is UNsigned integer)",

16: "numpy.int16",

32: "numpy.int32",

64: "numpy.int64",

-32: "numpy.float32",

-64: "numpy.float64"

}

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_HDULists\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

#Below returns a HDUList of the Fit data files.

HDUs = [] #List to store HDU (Header Data Unit) for HA, OIII and SII.

for x in range(len(fitFiles)):#looping through length of array

HDUs.append(fits.open(fitFiles[x])) #Open fits file and add HDU data into HUDs array

print(HDUs[x].info()) #Outputs a Summary of the info of the HDU List.

print("")

print("\_\_\_\_\_\_\_Impotant Header information from all PrimaryHDU's\_\_\_\_\_\_\_")

print("")

print("x is : ", x)

#Getting key names from ver long Primary HDU Dictionary

head0 = fits.getheader(fitFiles[x]) # Could have used (h) or (i) but all info are equivalent

#This is only useful for looking at the key names within the Primary HDU's.

"""

print(head0) is astropy's version of head0.keys() to get key names.

In addition, this printing method also shows the assigned data to each key.

"""

#Using getval() to get specific information from Primary and Image HDU

print("Name of Object: ", fits.getval(fitFiles[x],"OBJECT"))

print("Number of data Axes for ImageHDU: ", fits.getval(fitFiles[x],"NAXIS1"), "pixels")

print("Resolution: ", fits.getval(fitFiles[x],"RESOLUTN")," ", fits.getval(fitFiles[x],"RESOUNIT"))

print("Color Spacing: ", fits.getval(fitFiles[x],"COLORSPC"))

print("Approximate right ascension in hours: (", fits.getval(fitFiles[x],"OBJCTRA"), ")")

print("Approximate declination: (", fits.getval(fitFiles[x],"OBJCTDEC"), ") degrees")

BITPIXData = fits.getval(fitFiles[x],"BITPIX") #Getting BitPixelData from Primary HDU

if BITPIXData in numpyDatTyp: #If the bitPixel number is in the defined dictionary

print("number of bits per data pixel: ", BITPIXData, " meaning of type ", numpyDatTyp[BITPIXData])

print("")

HDUDataTitles = ["\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_HA ImageHDU Data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_","\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_OIII ImageHDU Data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SII ImageHDU Data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"]

#Displaying the default data Matrix from PrimaryHDU's of HA, OIII and SII

for x in range(len(fitFiles)): #Looping through length of array fitFiles

print(HDUDataTitles[x]) #Print the relative title

print("")

print(HDUs[x][1].data) #From the image HDU, show the ImageHDU of current fitFile.

print("")

tempArray = np.array(HDUs[0][1].data) #converting image HDU file into a numpy matrix

pixelNum= np.shape(tempArray)[0] #Getting the row number of matrix

# Intensity Spectra of Nebulae

# S. Amani Njoroge

# 4060924

# DataAnalysis

from FitsExtraction import HDUs, pixelNum #importing HDU Data Sets from FIT files as well as copies of the Fit files

import numpy as np #Importing numpy for useful array manipulation.

import matplotlib.pyplot as plt #Used for graph plotting

from statistics import median, multimode, stdev

import sys

"""

Converting PrimaryHDU's into numpy arrays by firstly extracting them from

FitsEctract.py package

"""

arrs = [] #Storing numpy matrix of ImageHDU of HA, OIII and SII respetively.

plot\_Titles = ["Hydrogen Alpha", "Oxygen III", "Silicon II"]

import showSection

for x in range(len(HDUs)):

#HDUs[x][0] means the Primary HDU whilst HDUs[x][1] would have been ImageHDU

arrs.append(np.array(HDUs[x][1].data))

#Making sure to close each Fits file after accessing.

#Must come after converting the data to numpy array first

HDUs[x].close()

matrix\_title = ["\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_HA array/matrix\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_OIII array/matrix\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_", "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SII array/matrix\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"]

for x in range(len(arrs)): #Looping through length of arrs (starting from 0 to length-1)

print(matrix\_title[x]) #Print the title matrix before printing the matrix data

print("")

print(arrs[x]) #Print the matrix

print("")

print("Please note that 2D arrays are already matrices")

print("")

minToMax\_arr = [] #Matrix needed to store 1D data for checking Max and Min

#photons in particular array

#Array storing string elements for ouptutting stat titles

stat\_titles = ["\_\_\_\_\_HA Stats\_\_\_\_\_\_\_\_", "\_\_\_\_\_OIII Stats\_\_\_\_\_\_\_\_", "\_\_\_\_\_SII Stats\_\_\_\_\_\_\_\_" ]

max\_vals = [] #storing Maximum value of HA, OIII and SII into array, RESPECTIVELY

min\_vals = [] #storing Minimum value of HA, OIII and SII into array, RESPECTIVELY

median\_arr = [] #storing Median value of HA, OIII and SII into array, RESPECTIVELY

modes\_arr = [] #storing Modal(s) value of HA, OIII and SII into array, RESPECTIVELY

stDev\_arr = [] #storing standard Deviation value of HA, OIII and SII into array, RESPECTIVELY

for x in range(len(stat\_titles)):

print(stat\_titles[x]) #print the current stat title

print("")

for i in arrs[x]: #Looping over each individual sub-araay list

for j in i: #Looping through each element in specific sub-array list

minToMax\_arr.append(j) #Adding each element into 1D array

minToMax\_arr.sort() #Sorts the array in ascending order

maxim, minim = minToMax\_arr[-1], minToMax\_arr[0]

print("Maximum photons in a pixel = ", maxim) #Max pixel value

print("Minimum photons in a pixel = ", minim) #Min Pixel value

med = median(minToMax\_arr)

print("Median = ", med) #Getting median value

median\_arr.append(med)

mode = multimode(minToMax\_arr)

print("Mode(s) = ", mode) #Getting modal value. Using multimode in case of two modes or more

modes\_arr.append(mode)

mean = sum(minToMax\_arr)/len(minToMax\_arr) #Getting Mean value of data

std = stdev(minToMax\_arr, xbar = mean)

print("StDev = ", std) #Standard Deviation

stDev\_arr.append(std)

max\_vals.append(maxim)

min\_vals.append(minim)

minToMax\_arr.clear() #Clearing array of all content for OIII and SII data storing

print("")

###############################################################################

"""

Below is the part where I Look at levels of high to relatively low intensity

The levels of intensity will be rated by the maximum and minimum values that

were previously collected.

All other lower levels of intensity are counted as negligible if user chooses

a number of levels.

By increasing the scaling factor, the data is categorized in more levels.

By inputting the number of levels (starting from the highest level; Level 1),

the data recorded will be categorized upto that level.

"""

###############################################################################

#Creating user-input based Scale-Levels for relative intensity

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

scF = int(input("""

What Integer Scaling Factor would you like to make in order to create Scale-Levels for relative intensity?

Its advisable to choose a high number such as 40 or %d.

Highest value to be chosen is %d due to a limitation of pixels.

""" % (int(pixelNum/4),pixelNum)))

Levels = int(input("""

Levels have been successfully constructed.

Input the number of level intensities you desire (from highest intesity as the first level)

If you desire to see all levels of intensity, then type %d again.

(Be Warned, it might take some time with regards to the processessing power of your machine)

Recommended to choose upto the top quarter tier such as %d or lower.

So what will it be?

""" % (scF,int(scF/4))))

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print("")

if scF > pixelNum: #If the chosen number of levels are bigger than the Scaling Factor:

print("""

You have split the data into %d levels but the limitation (based on pixels) is %d.

""" % (scF, pixelNum))

sys.exit("Please rerun the code an choose wisely.") #exiting the code with a message if the preceeding if statement is met

if Levels > scF: #If the chosen number of levels are bigger than the Scaling Factor:

print("""You have split the data into %d levels but have chosen your levels of interests to level %d.

""" % (scF, Levels))

sys.exit("Please rerun the code an choose wisely.") #exiting the code with a message if the preceeding if statement is met

print("NOTE: data output will be given as: ( <x-coord> , <y-coord> , <Intensity level> )")

print("")

print("If <Intensity level> = 0, then this is the only pixel in the matrix that has the largest number of photons")

print("")

scales = [] #Array holding constructed scales for HA OIII and SII

for x in range(len(max\_vals)):

scales.append((max\_vals[x] - min\_vals[x])/scF)

XYm\_gList = [] #List that holds tuples of (x,y,intensity level) for g\_arr

XYm\_hList = [] #List that holds tuples of (x,y,intensity level) for h\_arr

XYm\_iList = [] #List that holds tuples of (x,y,intensity level) for i\_arr

#array containing all XY\_()List

XYm\_Lists = [XYm\_gList, XYm\_hList, XYm\_iList]

for q in range(len(XYm\_Lists)):

i\_idxNum = 0 #Index number

for i in arrs[q]: #Looping through array of HA, then OIII then SII

y = i\_idxNum #row number or y coordinate of element

j\_idxNum = 0 #column number

for j in i: #for every element in the list "i"

mltp = 0 #creating multiplier variable

x = j\_idxNum #column number or x coordinate of element

while mltp <= scF and mltp<=Levels+1: #If the multiplier vairable smaller than or equal to the scaling factor

#Also making sure to dump the rest of the other data inside an extra lower level

if j >= max\_vals[q] - mltp\*scales[q]:

"""Checking condition if data is in specific intensity level

in terms of the pixel's photon number First level is from maximum (inclusive)

to lover region of that first level (aslo inclusive)"""

if mltp == Levels+1 or (x,y,mltp-1) in XYm\_Lists[q]: #If data lands in extra level or there already exists a coordinate

break #Get out of while loop rather than recording it

else:

temp\_tup = (x,y,mltp) #Trapping tuple of coords and their corresponding multiplier

#tuples t in (<x>,<y>, t) where t = 0 are just the pixel coordinates that have maximum number of photons.

XYm\_Lists[q].append(temp\_tup) #Tuple will be having cooridnate elements x, y as well as the reverse multiplier prescribed to that coordinate.

mltp += 1 #Increasing multiplier to find the next relative intensity

j\_idxNum += 1 #Incrementing the x-cooordinate

i\_idxNum += 1 #Incrementing the y-coordinate

#Sorting all tuple elements in the List array

XYm\_Lists[q].sort() #Sorting list in ascending order

#Note that input() automatically converts user intput into a string

if Levels !=1:

choice = input("""

Would you like to see all levels from highest to chosen level ? If so type "Yes" or "all"

(make sure to use quotation marks)

Or

Would you like to see only one level ? If so type the integer level number:

Or

Would you rather see specific levels?

e.g: Only want pixel coordinate with the highest value: type -> 0

Only level 2 and level 3: then type -> 2,3

Only Level 6 and level 31 and level 4: then type -> 6, 31 , 4

(spacing doesnt matter, but make sure to separate using commas)

""")

print("---------------------------------------------------------")

print("")

#NOTE: choice is only a string list and needs to be converted for proper usage:

numbers = ["1","2","3","4","5","6","7","8","9"]

if "," in choice: #If multitude of specific levels are chosen

#Convert choice string input into list of those specific levels

f = choice.split(",") #Automatically splits every string element based on the specified arguement splitter

#f is an array of strings with each string being a level number

d = [] #Element to store integer values

for i in f: #for every string element

d.append(int(i)) #adding integer version of each number string into new list

elif "," not in choice and choice[0] in numbers: #if the input was only of one level

d = int(choice) #Convert level from string to integer

else:

d = choice #Remains as string

pass #Take it as a String value cause it has no numbers.

#function that filters lists by removing unwanted tuples based on chosen level from user input

def levelFilter(tuplist): #creation of tuple-filtering function

templist= []

#If the level given is only an integer value of a specific level

if type(d) == int:

for tup in tuplist: #For each tuple within the list

if tup[2] == d: #if the intesity level within tuple is equal to desired user input

templist.append(tup) #Add tuple element into temporary list

#If the level given is an array of specifically chosen levels

elif type(d) == list:

for tup in tuplist: #For each tuple within the list

for h in d: #For each level of interest given by user

if tup[2] == h: #if the intesity level within tuple is equal to desired user input

templist.append(tup) #Add tuple element into temporary list

return templist

for x in range(len(XYm\_Lists)): #looping through length of array XYm\_Lists

#if the input is "yes" or "all"

if type(d) == str:#Making sure its not case sensitive by just checking if its a string

if XYm\_Lists[x] == []: #If the list is empty

print("Coordinates of %s from intensity Level 1 -> %d:" % (plot\_Titles[x],Levels))

print("")

print("There are no intenisties within Level 1 -> %d for %s" %(Levels, plot\_Titles[x]))

print("")

else:

if Levels > 1: #If levels desired are coninuous from highest intensity till a desired limit

print("Coordinates of %s from intensity Level 1 -> %d:" % (plot\_Titles[x],Levels))

print("")

print(XYm\_Lists[x]) #Print the list of tuples

print("")

else:

print("Coordinates of %s from intensity Level 1:" % (plot\_Titles[x]))

print("")

print(XYm\_Lists[x]) #Print the list of tuples

print("")

else: #Otherwise print out the filtered tuples

filt = levelFilter(XYm\_Lists[x])

if filt == []: #If the list is empty

if type(d) == list: #If user had inputted more than one specific intenisty level

print("Coordinates of %s in levels %s data:" % (plot\_Titles[x],choice))

print("")

print("There are no intenisties within the levels ",d,", based on your Scaling Factor; ", scF)

print("")

else: #If it was not a list of levels but instead just one level

print("There is no intensity value belonging within that level %d, based on your Scaling Factor; %d" % (d,scF))

print("")

else:

print("Coordinates of %s in levels %s data:" % (plot\_Titles[x],choice))

print("")

print(filt) #printing out filtered tuples

print("")

else:

for x in range(len(XYm\_Lists)): #looping through length of array XYm\_Lists

print("Coordinates of %s from intensity Level 1:" % (plot\_Titles[x]))

print("")

print(XYm\_Lists[x]) #Print the list of tuples

print("")

# Intensity Spectra of Nebulae

# S. Amani Njoroge

# 4060924

# ShowSection

from DataAnalysis import arrs #Importing arrays

import matplotlib.pyplot as plt

colors = plt.cm.gray #Making Background black and data white.

def show(filename):

if filename.endswith(".fit") == True:

if filename.startswith("OIII") or filename.startswith("SII") or filename.startswith("HA"):

if filename == "HA.fit": #Checking whether input data is same as output

N1 = plt.Normalize(arrs[0].min(), arrs[0].max())

#Creation of a normalizer based on min value to max value

"""Normalizing colour band makes sure

entire range of black to white is

used for particular data set"""

arrsNorm0 = colors(N1(arrs[0])) #Normalizing the matrix values

'''new matrix will have assigned colours for each pixel

based on pixel value. The higher the value the brighter the grey

(towards white) and the lower the value the darker the grey (towards black)

'''

plt.axis("off") #No axis on the image

plt.title("Hydrogen Alpha") #Title of the image

plt.imshow(arrsNorm0) #pyplot DAature tospit out the array in an image

elif filename == "OIII.fit":

N2 = plt.Normalize(arrs[1].min(), arrs[1].max())

arrsNorm1 = colors(N2(arrs[1]))

plt.axis("off")

plt.title("Oxygen III")

plt.imshow(arrsNorm1)

elif filename == "SII.fit":

N3 = plt.Normalize(arrs[2].min(), arrs[2].max())

arrsNorm2 = colors(N3(arrs[2]))

plt.axis("off")

plt.title("Silicon II")

plt.imshow(arrsNorm2)

else:

print("You typed the name of file wrongly: HA, OIII and SII are the only existing ones")

else:

print("You have not inserted the name properly: <name.fit>")

plt.show()