# import packages to use

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.pyplot import \*

from astropy.utils.data import get\_pkg\_data\_filename

from astropy.table import Table

from astropy.io import fits

import glob

import batman

import lmfit

import corner

import matplotlib.pyplot as plt

from matplotlib import pylab

from numpy import arange,array,ones

from scipy import stats

import numpy.polynomial.polynomial as poly

from scipy.optimize import curve\_fit, minimize\_scalar

directory='/home/ian/Desktop/WebbData/visit23\_defringed/'

#directory='/Users/insertpathhere/'

number\_of\_rows=64

number\_of\_columns=1024

number\_of\_images=43

#Load images into a list

list=glob.glob(directory+"\*.fits")

#print, first image in list

print(list[0])

##Example load first fits image

hdul=fits.open(list[1])

#Get MJD mid time of exposure from Header, which has start and end MJD times

mjd\_start=hdul[0].header['EXPSTART']

mjd\_end=hdul[0].header['EXPEND']

mjd=(mjd\_end+mjd\_start)/2.

print(mjd)

#load fits file image into an array called 'data'

data = hdul[0].data

data.shape #size of image

data.dtype.name #type of image

print(np.sum(data)) #total counts in image

#close fits after loading in data needed

hdul.close()

index\_of\_images=np.arange(number\_of\_images)

index\_of\_rows=np.arange(number\_of\_rows)

index\_of\_columns=np.arange(number\_of\_columns)

total\_counts=np.zeros((number\_of\_images))

mjd=np.zeros((number\_of\_images))

row\_sum=np.zeros(number\_of\_rows) #sums pixel count for each row of image

maxRow=np.zeros(number\_of\_columns) #number of row in which max count exists in cols

col\_vals=np.zeros(number\_of\_columns)

col\_max=np.zeros(number\_of\_columns) #to contain max photon count per column in an image

col\_posn\_max=np.zeros(number\_of\_columns) #to contain position of max photon count per column in an image

col\_max\_posn=np.zeros((number\_of\_images, number\_of\_columns)) #contains all column pc among all images

col\_max\_value=np.zeros((number\_of\_images, number\_of\_columns)) #contains all position pc among all images

gauss\_max\_value=np.zeros((number\_of\_images, number\_of\_columns))

gauss\_max\_posn=np.zeros((number\_of\_images, number\_of\_columns))

col\_gauss = np.zeros(number\_of\_rows) #Create arrays for gaussian fits

col\_test= data[0:64,30]

x = np.arange(0,64)

y = col\_gauss

def Gauss(x, a, x0, sigma): return a \* np.exp(-(x - x0)\*\*2 / (2 \* sigma\*\*2))

for i in index\_of\_images:

img=list[i]

#print(img)

hdul=fits.open(img)

mjd\_start=hdul[0].header['EXPSTART']

mjd\_end=hdul[0].header['EXPEND']

mjd\_image=(mjd\_end+mjd\_start)/2.

mjd[i]=mjd\_image

data = hdul[0].data

#print(np.sum(data)) #total counts in image

for j in index\_of\_columns:

#total\_counts[i]=np.sum(data[0:64,j]) #total counts in column j of image

#col\_posn\_max[j]=np.argmax(data[0:64,j])

#col\_max[j]=max(data[0:64,j])

col\_gauss = data[0:64,j]

mean = sum(x \* col\_gauss) / sum(col\_gauss)

sigma = np.sqrt(sum(col\_gauss \* (x - mean)\*\*2) / sum(col\_gauss))

popt,pcov = curve\_fit(Gauss, x, col\_gauss, p0=[max(y), mean, sigma])

col\_posn\_max[j]=np.argmax(Gauss(x, \*popt))

col\_max[j]=np.max(Gauss(x, \*popt))

gauss\_max\_value[i][j]=popt[0]

gauss\_max\_posn[i][j]=popt[1]

#print(total\_counts)

# print(row\_vals) #prints array for each image w/ max photon count per column

#col\_max\_posn[i]=total\_counts

#col\_max\_value[i]=col\_max

#print(col\_max\_value)

#col\_max\_posn[i]=col\_posn\_max

#print(col\_max\_posn)

all\_linear\_lines = np.zeros((number\_of\_images, number\_of\_columns)) #collect all fit lines for every image

for i in index\_of\_images:

y=gauss\_max\_posn[i]

x=index\_of\_columns

slope, intercept, r\_value, p\_value, std\_err = stats.linregress(x,y)

line = np.around(slope\*x+intercept)

all\_linear\_lines[i]=line

print(all\_linear\_lines)

ApertureCountsByRanges = np.zeros((43, 10, 10), dtype=object)

#ApertureCountsPerImage = np.zeros(43)

ApertureTotalCounts = 0

ApertureRangeError = np.zeros((43, 10, 10), dtype=object)

g = 0

h = 0

while (g < 10):

g=g+1

h=0

while (h < 10):

h=h+1

for i in index\_of\_images:

img=list[i]

hdul=fits.open(img)

data = hdul[0].data

for j in index\_of\_columns:

#print(np.sum(data)) #total counts in image

Col\_sum=np.sum(data[int(all\_linear\_lines[i][j]-[h]):int(all\_linear\_lines[i][j]+[g]),j]) #total counts in 2 pix wide aperture

ApertureTotalCounts = ApertureTotalCounts+Col\_sum

#ApertureCountsPerImage[i] = ApertureTotalCounts

#ApertureCountsByRanges[i][g-1][h-1] = ApertureCountsPerImage

ApertureCountsByRanges[i][g-1][h-1] = ApertureTotalCounts

ApertureTotalCounts = 0

ApertureRangeError[i][g-1][h-1]=np.sqrt(np.sum(ApertureCountsByRanges))

g = 0

h = 0

ppmByRange=np.zeros((43,10,10))

for i in index\_of\_images:

while (g < 10):

h=0

while (h < 10):

#TRANSIT model batman package https://astro.uchicago.edu/~kreidberg/batman/

#Setup inital parameters (can get from http://exoplanets.org/detail/WASP-39\_b)

params = batman.TransitParams() #object to store transit parameters

params.t0 = 55342.46880 #mjd time of inferior conjunction 56368.454950

params.per = 4.0552590 #orbital period

params.rp = 0.14491 #planet radius (in units of stellar radii)

params.a = 11.70 #semi-major axis (in units of stellar radii)

params.inc = 87.83 #orbital inclination (in degrees)

params.ecc = 0. #eccentricity

params.w = 90. #longitude of periastron (in degrees)

params.limb\_dark = "nonlinear" #limb darkening model

params.u = [0.6642, -0.4701, 1.0466,-0.5097] #limb darkening coefficients

#>>> dir(params) #View attributes

#['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_doc\_\_', '\_\_format\_\_', '\_\_getattribute\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_module\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'a', 'ecc', 'fp', 'inc', 'limb\_dark', 'per', 'rp', 't0', 't\_secondary', 'u', 'w']

#print(params.rp)

#0.1453

# create a set of Parameters

p = lmfit.Parameters() #object to store L-M fit Parameters

p.add('rprs', value=params.rp) # HST phase^4 parameter

p.add('t0', value=params.t0) # HST phase^4 parameter

p.add('f0', value=10736338.8, min=0 , max=ApertureCountsByRanges[i][g][1]\*1.2) #Baseline Flux

p.add('p1', value=0.0) # Orbital phase parameter

p.add('p2', value=0.0) # HST phase^1 parameter

p.add('p3', value=0.00) # HST phase^4 parameter

p.add('p4', value=0.0) # HST phase^3 parameter

p.add('p5', value=0.0) # HST phase^4 parameter

#Batman Transit model

t\_fine = np.linspace(np.min(mjd), np.max(mjd), 1000) #times at which to calculate light curve

m = batman.TransitModel(params, t\_fine) #initializes model

transit\_fine = m.light\_curve(params) #Calculates Fine-Grid Tranist model

m = batman.TransitModel(params, mjd) #initializes model

transit\_mjd = m.light\_curve(params) #Calculates Fine-Grid Tranist model

#Calculate phase of HST telescope, 96.36 minutes, used for detrending

#phase\_hst array values should be between -0.5 and 0.5

phase\_hst=(mjd-np.min(mjd))/(6.691666E-2) #phase in days relative to first exposure

phase\_hst\_fix=np.fix(phase\_hst) #rounds to nearest zero

phase\_hst=phase\_hst-phase\_hst\_fix #all between 0 to 1.0

phase\_hst[np.where(phase\_hst > 0.5)]=phase\_hst[np.where(phase\_hst > 0.5)]-1.0 #all between -0.5 and 0.5

#matplotlib.pyplot.scatter(mjd,phase\_hst) #View

#Calculate Orbital Phase

phase=(mjd-params.t0)/(params.per) #phase in days relative to T0 ephemeris

phase=phase-np.fix(phase[number\_of\_images-1]) # Have current phase occur at value 0.0

x=mjd

y=ApertureCountsByRanges[i][g][h]

err=ApertureRangeError[i][g][h]

def residual(p):

params.rp = p['rprs'].value # Set Batman rprs to new fit rprs

params.t0 = p['t0'].value # Set Batman rprs to new fit rprs

m = batman.TransitModel(params, mjd) #initializes model

transit\_mjd = m.light\_curve(params) #Calculates Fine-Grid Tranist model

model=transit\_mjd\*p['f0'] \* (p['p1']\*phase + p['p2']\*phase\_hst + p['p3']\*phase\_hst\*\*2. + p['p4']\*phase\_hst\*\*3. + p['p5']\*phase\_hst\*\*4. + 1.0) #Simple transit model is baseline flux X transit model

chi2now=np.sum((y-model)\*\*2/err\*\*2)

#print("current chi^2=",chi2now)

return (y-model)/err

def model(p):

params.rp = p['rprs'].value # Set Batman rprs to new fit rprs

params.t0 = p['t0'].value # Set Batman rprs to new fit rprs

m = batman.TransitModel(params, mjd) #initializes model

transit\_mjd = m.light\_curve(params) #Calculates Fine-Grid Tranist model

model=transit\_mjd\*p['f0'] \* (p['p1']\*phase + p['p2']\*phase\_hst + p['p3']\*phase\_hst\*\*2. + p['p4']\*phase\_hst\*\*3. + p['p5']\*phase\_hst\*\*4. + 1.0) #Simple transit model is baseline flux X transit model

return model

def model\_fine(p):

params.rp = p['rprs'].value # Set Batman rprs to new fit rprs

params.t0 = p['t0'].value # Set Batman rprs to new fit rprs

m = batman.TransitModel(params, t\_fine) #initializes model

transit\_mjd = m.light\_curve(params) #Calculates Fine-Grid Tranist model

model\_fine=transit\_mjd\*p['f0']

return model\_fine

# create Minimizer

mini = lmfit.Minimizer(residual, p, nan\_policy='omit')

# first solve with Nelder-Mead

#out1 = mini.minimize(method='Nelder')

# then solve with Levenberg-Marquardt using the

# Nelder-Mead solution as a starting point

# https://lmfit.github.io/lmfit-py/fitting.html

result = mini.minimize(method='leastsq')

#result = mini.minimize(method='leastsq', params=out1.params)

#print(dir(result)) # To View All Atributes of the

#print("redchi",result.redchi)

#print("chi2",result.chisqr)

#print("nfree",result.nfree)

#print("bic",result.bic)

#print("aic",result.aic)

#print("L-M FIT Variable")

#print(lmfit.fit\_report(result.params))

#print(p['f0'].value)

# file-output.py

#Update with best-fit parameters

p['rprs'].value=result.params['rprs'].value

p['t0'].value=result.params['t0'].value

p['f0'].value=result.params['f0'].value

p['p1'].value=result.params['p1'].value

p['p2'].value=result.params['p2'].value

p['p3'].value=result.params['p3'].value

p['p4'].value=result.params['p4'].value

p['p5'].value=result.params['p5'].value

# Re-calculate Bestfit Model

final\_model=model(p)

final\_model\_fine=model\_fine(p)

#print("residual standard deviation",np.std((y-final\_model)/p['f0'].value))

#print("residual standard deviation (ppm)",1E6\*np.std((y-final\_model)/p['f0'].value))

ppmByRange[i][g][h]=1E6\*np.std((y-final\_model)/p['f0'].value)

#ppmByRange[g][h]=currentppm

#Plot data models

#matplotlib.pyplot.scatter(mjd,total\_counts, linewidth=1.5)

#plt.errorbar(mjd,total\_counts,yerr=ApertureRangeError[g-1][h-1], fmt='o')

#xlabel('MJD')

#ylabel('Counts')

#overplot models

#plot(t\_fine,final\_model\_fine, linewidth=1.5) #overplot fine-grid Transit model

#matplotlib.pyplot.scatter(mjd,final\_model, linewidth=2) #overplot Transit model at data

#show()

#Plot Residuals

#matplotlib.pyplot.scatter(mjd,(y-final\_model)/p['f0'].value, linewidth=2) #overplot Transit model at data

#show()

h=h+1

print(ppmByRange)

g=g+1