Tutorial 4: Measuring the Emission-line Kinematics from AO-assisted Keck/OSIRIS Observations of NGC 1275

Name: Felix Martinez
Partner: Emily Hay
Date: 4/22/21

1 Overview

This tutorial will walk through how to fit models of spectra, and how to measure parameter uncertainties with a Monte Carlo approach. The galaxy we will be doing this walk through for is NGC 1275, it is the brightest cluster galaxy (BCG) of the Perseus cluster and is on the closest BCGs at a distance of $\sim$ 75 Mpc. The observations of NGC 1275 were taken with the 10m Keck telescope using adaptive optics using the integral field spectrograph OSIRIS on Dec 30, 2010 using a narrow K-band filter (Kn3) which covers 2.121 - 2.229 μm.

The OSIRIS data has already been reduced before this tutorial (see Tutorial 3 for detailed steps). We will further generate the binned spectra from the data cube, and fit a function to an H_2 emission line to measure the velocity (V), velocity dispersion (σ), and flux of the gas as a function of spatial location. With the velocity information, we will be able to roughly estimate the Black Hole (BH) mass of NGC 1275 central Black Hole.

2 Results

Below we summarize our work for each section of the tutorial and address the questions posed in the manual.

2.1 Data

The reduced OSIRIS data cube is called **ngc1275_osiris.fits** located in $\sim$**/ngc1275_osiris/data/**, there is also a text file called **ngc1275_osiris_bins.txt** in this directory. We must first create a new directory called **analysis** in $\sim$**/ngc1275_osiris/** to store a location for our work. We will then write code that reads in the 0^{th} extension of the data cube, and the parameters of the text file. We will then construct 286 binned spectra, saving each bin as a fits file. Furthermore, we will construct and save a reference wavelength using the keywords: CRPIX3 (the reference pixel), CRVAL3 (the reference wavelength), and CDELT3 (the wavelength step size).

The code we used to accomplish the above task is shown below on the next page.

```
import numpy as np
import matplotlib.pyplot as plt
import os
from astropy.io import fits

# Setting up our Paths
dir_output = '/home/student/ngc1275_osiris/analysis/'
dir_input = '/home/student/ngc1275_osiris/data/'

# number of spatial bins
numbins = 286

# read in the file that gives the lenslet and corresponding bin.
# store the x spatial location as x_lens, the y spatial location as
# y_lens, and the bin as bin_num50
bin_files = np.loadtxt(dir_input+'/ngc1275_osiris_bins.txt')
x_lens = bin_files[:,0]
y_lens = bin_files[:,1]
bin_num = bin_files[:,2]


data_cube = fits.open(dir_input+'ngc1275_osiris.fits')[0].data
header = fits.open(dir_input+'ngc1275_osiris.fits')[0]
pix = header.header['CRPIX3']
delta = header.header['CDELT3']
value = header.header['CRVAL3']

# defining and saving the wavelength array to a fits file called ngc1275_wave.fits
# and store it in dir_output
wave = ((np.arange(len(data_cube))+1 - pix)*delta)+value

hdu = fits.PrimaryHDU(wave)
hdul = fits.HDUList([hdu])
hdul.writeto(dir_output+'/ngc1275_wave.fits')

# go through each of the spatial bins
for i in range(len(bin_files)):

    # set spec_tmp to be an array of zeros the same length as
    # the wavelength array. this will hold the fluxes of our
    # binned spectrum.
    spec_tmp = np.zeros(len(wave))

    # determine the indices where bin_num equals the current bin
    # we are working on.
    ind = (np.where(bin_num == i))[0]

    # multiple lenslets belong to a bin. loop through each lenslet
    # that belongs to the current bin. take the flux as a function
    # of wavelength of each lenslet and add them together.
    for j in range(len(ind)):
        x_tmp = x_lens[ind[j]]
        y_tmp = y_lens[ind[j]]
        spec_tmp = spec_tmp + data_cube[:,int(y_tmp),int(x_tmp)]



    # spec_tmp holds the total spectrum of the bin. determine the
    # average spectrum of the bin and store as spec_tmp_ave.
    spec_tmp_ave = spec_tmp/len(ind)

    # set the name of the output file
    file_out = 'ngc1275_spec_bin'+str(i)+'.fits'

    # save the average spectrum to a fits file called file_out
    # located in dir_output.
    hdu = fits.PrimaryHDU(spec_tmp_ave)
    hdul = fits.HDUList([hdu])
    hdul.writeto(dir_output+file_out)
```

Figure 1: The code we used to produce and save the spectrum for our spatial bins and constructed and save our wavelength array.

We also plotted the spectrum for bin 24 that nicely highlights H_2 and Br_γ emission lines, we also found where H_2 and Br_γ should occur using the equation:

$$z = \frac{\lambda_{obs} - \lambda_{rest}}{\lambda_{rest}} \tag{1}$$

Where we know the galaxy has a redshift of $z = 0.017264$, and that H_2 has a rest wavelength of ($\lambda_{rest} = 2.12183\mu$m), and Br_γ has a rest wavelength of ($\lambda_{rest} = 2.16612\mu$m).

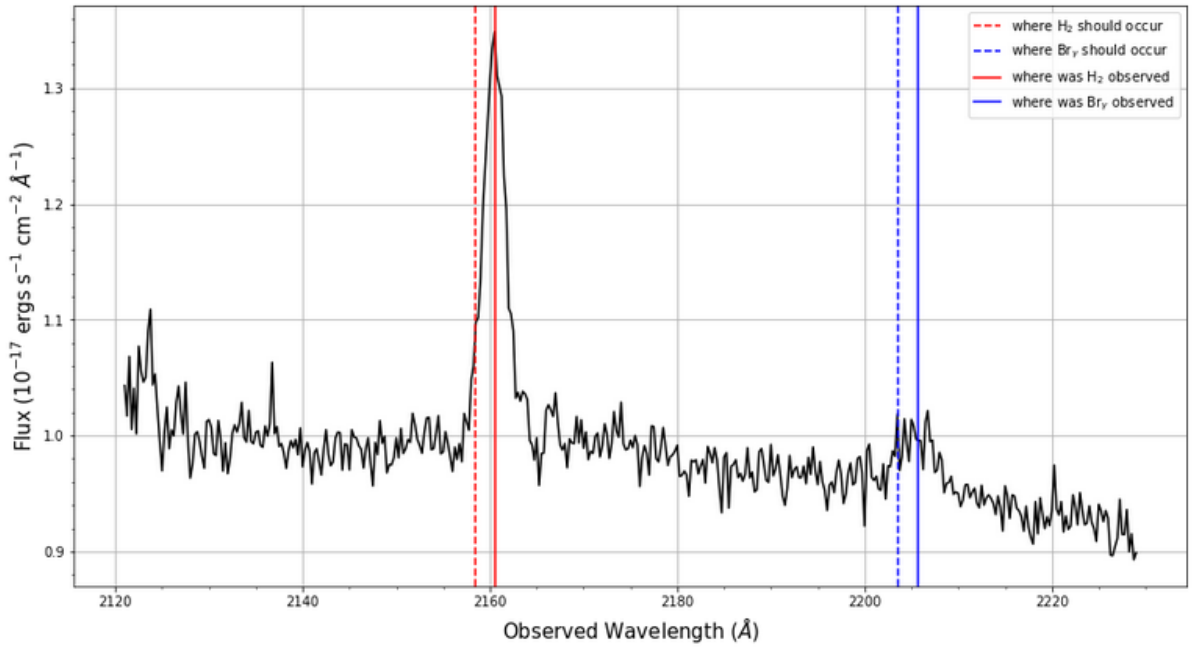

Figure 2: The spectrum of bin 24 showing our H_2 and Br_γ emission lines and where they should occur.

Strangely, our H_2 and Br_γ emission lines occur at a higher wavelength than that of where they were expected to occur. We believe this to be a result of the velocity of gas found throughout the galaxy, further shifting our measurements into the red.

2.2 Fitting the H_2 Emission Line

We performed a non-linear least squares fit to the data using **curve_fit**, which is apart of the Python SciPy package. To keep things tidy, we created a new directory in ∼**/ngc1275_osiris/analysis** called **fit_bin24**, coping over out spectrum from bin 24 and our wavelength fits file in the process. We used Python to fit a Gaussian plus a line to the H_2 spectrum trim of bin 24 (between the wavelengths of 2.140 μm and 2.180 μm). We used 5 parameters for this process, 3 for the Gaussian: Area (A), Center (C), Dispersion (σ), and 2 for the line: Slope (m), Intercept (b). We also were able to calculate the velocity in km s^{-1} of the H_2 emission line relative to the rest wavelength of the line, and the velocity dispersion in km s^{-1} by multiplying the speed of light c by the ratio of the dispersion of the Gaussian and the center of the Gaussian. Below is the function we used for our 5 parameter model.

$$M = \frac{A}{\sigma\sqrt{2\pi}} e^{-(x-C)^2/2\sigma^2} + (mx + b) \tag{2}$$

The initial starting guesses we included were: Area = 20 $\times$ 10^{-17} ergs s^{-1} cm^{-2} $Å^{-2}$, Center = 21605 Å, Dispersion = 20 Å, Slope = 0.00005 10^{-17} ergs s^{-1} cm^{-2} $Å^{-2}$, Intercept = -0.1 $\times$ 10^{-17} ergs s^{-1} cm^{-2} $Å^{-1}$. Below is a plot of our observed spectrum with the best-fit model and its residuals.

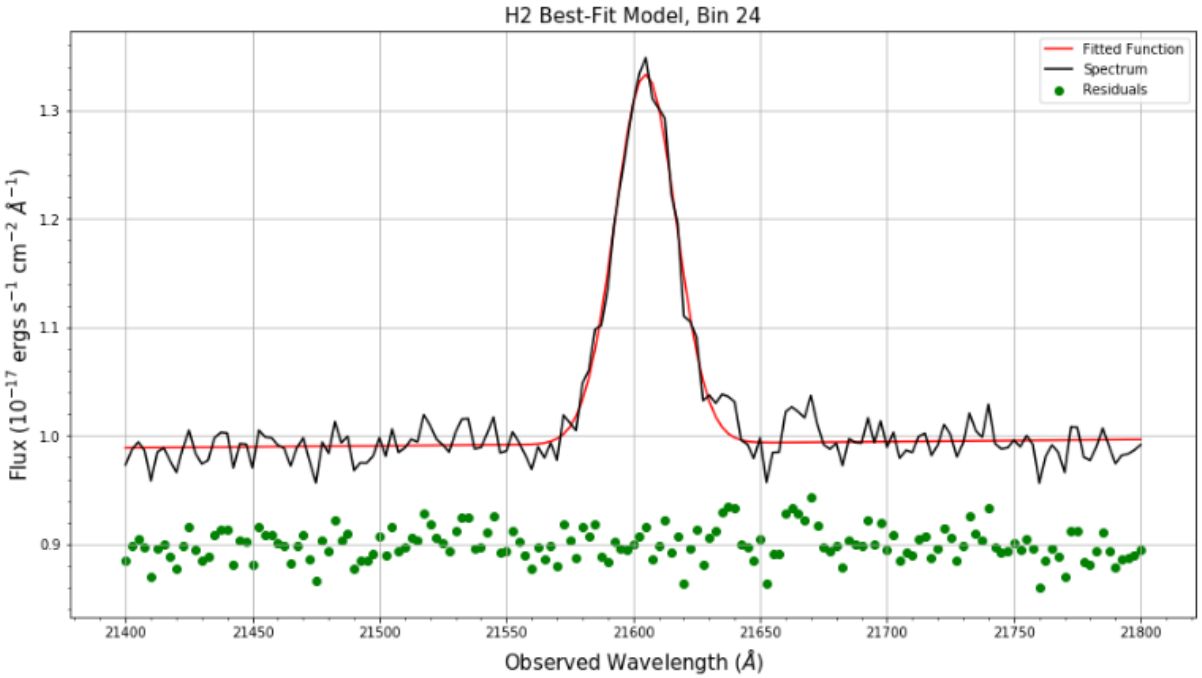

Figure 3: Our observed spectrum with the best-fit model and our residuals plotted alongside, we added a constant of 0.9 to our residuals to better display our data.

We further calculated the Standard deviation of our residuals shown below:

```
# our initial Guess
init_guess = [20, 21605, 20, 0.00005, -0.1]

# call curve_fit to fit the data using optimize.curve_fit and my_func
# you defined above.
params, params_cov = optimize.curve_fit(my_func, wave_trim, spec_trim, p0=init_guess)

# in order to determine the residuals (data - model), we need to
# reconstruct the model. use the best-fit parameter values to
# determine the model flux and store as spec_bestfit.
spec_bestfit = my_func(wave_trim, params[0],params[1],params[2],params[3],params[4])

# calculate the residuals (data - model, where model is spec_bestfit)
# and store as resid. resid will be an array of the same length as
# the data (and the same length as the model)
resid = spec_trim - spec_bestfit


# calculate the standard deviation of resid. if using np.std set
# ddof=1. print the standard deviation of the residuals to the screen.
std = np.std(resid, ddof=1)
print('Standard deviation of the residuals: '+str(std))
```

```
Standard deviation of the residuals: 0.015271093648217136
```

Figure 4: Our code showing us calculate the standard deviation of our residuals, we found them to have a standard deviation of 0.01527.

When calculating the Velocity in respect to our rest wavelength, we can use this equation:

$$\frac{\Delta\lambda}{\lambda_0} = \frac{V}{c} \tag{3}$$

Where $\Delta\lambda$ is our wavelength offset ($\lambda_{\rm obs} - \lambda_{\rm rest}$), λ_0 is the rest frame wavelength for H_2 (defined above to be $\lambda_{\rm rest} = 2.12183\mu$m), and c is the speed of light measured in km s^{-1}. Below is the code we used to find out velocity and velocity dispersion.

```
#Our rest wavelength
H2_rest = 2.12183*10000

#finding the velocity
H2_v = ((params[1]-H2_rest)/H2_rest)*c

# calculate the velocity dispersion (in km/s) of the H2 emission line.
v_dis = c * params[2]/params[1]

# print the velocity and velocity dispersion in km/s to the screen.
print('The velocity we found was: '+str(H2_v)+' km/s')
print('The velocity dispersion we found was: '+str(v_dis)+' km/s')

# print the velocity and velocity dispersion in km/s to a text file
np.savetxt(dir_output+'/H2_velocity_bin'+str(binnum)+'.txt',[H2_v, v_dis])
```

```
The velocity we found was: 5461.262689792794 km/s
The velocity dispersion we found was: 167.07519947506196 km/s
```

Figure 5: Our code showing us calculate the velocity (V) and velocity dispersion (σ). We found our velocity to be V = 5461.26 km/s, and our velocity dispersion to be σ = 167.07 km/s.

2.3 Measuring Parameter Uncertainties

As explained in our Overview, we will be using a Monte Carlo approach to calculate the uncertainties on the emission-line velocity, velocity dispersion, and flux assuming the best-fit model is a reasonable description of the data. We will write a code that takes the best-fit model for bin 24 and adds random Gaussian noise to the model, setting the standard deviation of the Gaussian distribution to that of the standard deviation of the residuals calculated in the section above. We will place the output of our results in the directory **fit_bin24**.

Our code showing the Monte Carlo simulation is shown on the next page.

```
# set up and perfom the Monte Carlo simulation
#Our rest wavelength
H2_rest = 2.12183*10000

# record the start time
t = perf_counter()

# the number of Monte Carlo itterations
mc_iter = int(1e5)

# loading in the best-fit values for the 5 parameters from the
# initial fit to the observed spectrum
bf_files = np.loadtxt(dir_input+params_input)

# our bestfit spectrum
spec_bestfit = my_func(wave_trim, bf_files[0],bf_files[1],
                       bf_files[2],bf_files[3],bf_files[4])

# create an array to hold the Monte Carlo results for the three
# Gaussian parameters only
arr_mc = np.zeros(shape=(nparams-2,mc_iter))

# our inital guesses carried over from the previous section
init_guess = [20, 21605, 20, 0.00005, -0.1]

# each step is one Monte Carlo iteration
for mc in range(mc_iter):

    # add random Gaussian noise to the best-fit model spectrum.
    mockspec = spec_bestfit + np.random.normal(0, std_resid, size=len(spec_bestfit))

    # fit the mock spectrum, use the same initial guesses every time.
    params_mc, params_cov_mc = optimize.curve_fit(my_func, wave_trim, mockspec,
                                                  p0=init_guess)

    # calculate the velocity (in km/s) and store in vel_mc
    vel_mc = (c * (params_mc[1] - H2_rest))/H2_rest

    # calculate the velocity dispersion (in km/s) and store in veldis_mc
    veldis_mc = c * (params_mc[2]/params_mc[1])

    # store the solution for the flux, velocity, and velocity
    # dispersion into arr_mc.
    arr_mc[0,mc] = params_mc[0]
    arr_mc[1,mc] = vel_mc
    arr_mc[2,mc] = veldis_mc


# write the Monte Carlo results (everything in arr_mc) to a text
# file with the bin number in the name and put the file in dir_output
file_out_mcdist = dir_output+'mc'+'_'+str(binnum)+'.txt'
np.savetxt(file_out_mcdist,arr_mc)

# print the elapsed time for the Monte Carlo simulation
print('Time for the Monte Carlo: %.2f s' % (perf_counter() - t))
print('')
```

```
Time for the Monte Carlo: 148.52 s
```

Figure 6: Our code showing us perform the Monte Carlo simulation, where we save the flux, velocity, and velocity dispersion for each iteration, it took a total of 148.52 seconds to run.

Alternate simply calculating the median and standard deviation of each distribution (Method A), we can use another method to find the flux, velocity and velocity dispersion. One could calculate the cumulative distribution function (CDF) for each of the three parameters and determine where the CDF equals 0.5, corresponding to the median, and measure the 1σ and 3σ confidence limits for each parameter (corresponding to 0.158 and 0.842 for 1σ uncertainties and 0.001 and 0.999 for 3σ uncertainties, otherwise known as Method B). Below is our code in which we used to calculate the CDF with the determination of the median, 1σ upper and lower bounds, and the 3σ upper and lower bounds.

```
for k in range(nparams):
    # initialize the array that will hold the cdf for the current parameter
    cdf = np.zeros(shape=mc_iter)

    # calculate the cdf
    param_sort = np.sort(arr_mc[k,:])
    for l in range(param_sort.size):
        val = param_sort[l]
        index_tmp = np.where(param_sort < val)
        cdf[l] = index_tmp[0].size / param_sort.size

    # determine the median value (0.5)
    med_b = param_sort[get_element(cdf,0.5)]

    # determine the 1-sigma lower bound (0.158)
    lower_1sig_value_b = param_sort[get_element(cdf,0.158)]

    # determine the 1-sigma upper bound (0.842)
    upper_1sig_value_b = param_sort[get_element(cdf,0.842)]

    # determine the 3-sigma lower bound (0.001)
    lower_3sig_value_b = param_sort[get_element(cdf,0.001)]

    # determine the 3-sigma upper bound (0.999)
    upper_3sig_value_b = param_sort[get_element(cdf,0.999)]


    # determine the size of the 1-sigma lower error bar
    err_lower_1sig_b = med_b - lower_1sig_value_b

    # determine the size of the 1-sigma upper error bar
    err_upper_1sig_b = med_b - upper_1sig_value_b

    # determine the size of the 3-sigma lower error bar
    err_lower_3sig_b = med_b - lower_3sig_value_b

    # determine the size of the 3-sigma upper error bar
    err_upper_3sig_b = med_b - upper_3sig_value_b
```

Figure 7: Our code showing us calculating the CDF (Method B) and finding the resulting median values and 1σ and 3σ upper and lower uncertainties.

We could also use the **numpy.percentile** function to perform an equivalent calculation to that of Method B (known as Method C), but with a linear interpolation for the case that the desired percentile lies between two data points. Using percentiles of 50.0% for the median, and 15.8% and 84.2% for the 1σ uncertainties and 0.1% and 99.9% for the 3σ uncertainties, we can find the velocity, velocity dispersion, and flux. The code we used for Method C can be seen below.

```
# method C
for k in range(nparams):

    # determine the median value by searching for the 50th
    # percentile
    med_c = np.percentile(arr_mc[k,:],50.0)

    # determine the 1-sigma lower bound, 15.8th percentile
    lower_1sig_value_c = np.percentile(arr_mc[k,:],15.8)

    # determine the 1-sigma upoer bound, 84.2th percentile
    upper_1sig_value_c = np.percentile(arr_mc[k,:],84.2)

    # determine the 3-sigma lower bound, 0.1th percentile
    lower_3sig_value_c = np.percentile(arr_mc[k,:],0.1)

    # determine the 3-sigma upper bound, 99.9th percentile
    upper_3sig_value_c = np.percentile(arr_mc[k,:],99.9)


    # determine the size of the 1-sigma lower error bar
    err_lower_1sig_c = med_c - lower_1sig_value_c

    # determine the size of the 1-sigma upper error bar
    err_upper_1sig_c = med_c - upper_1sig_value_c

    # determine the size of the 3-sigma lower error bar
    err_lower_3sig_c = med_c - lower_3sig_value_c

    # determine the size of the 3-sigma upper error bar
    err_upper_3sig_c = med_c - upper_3sig_value_c
```

Figure 8: Our code showing us calculating the flux, velocity, and velocity dispersion (Method C) and finding the resulting median values and 1σ and 3σ upper and lower uncertainties using **numpy.percentile**.

Below are plots of the CDFs for the velocity, velocity dispersion, and flux, with vertical lines to show the median value and the 1σ and 3σ values.

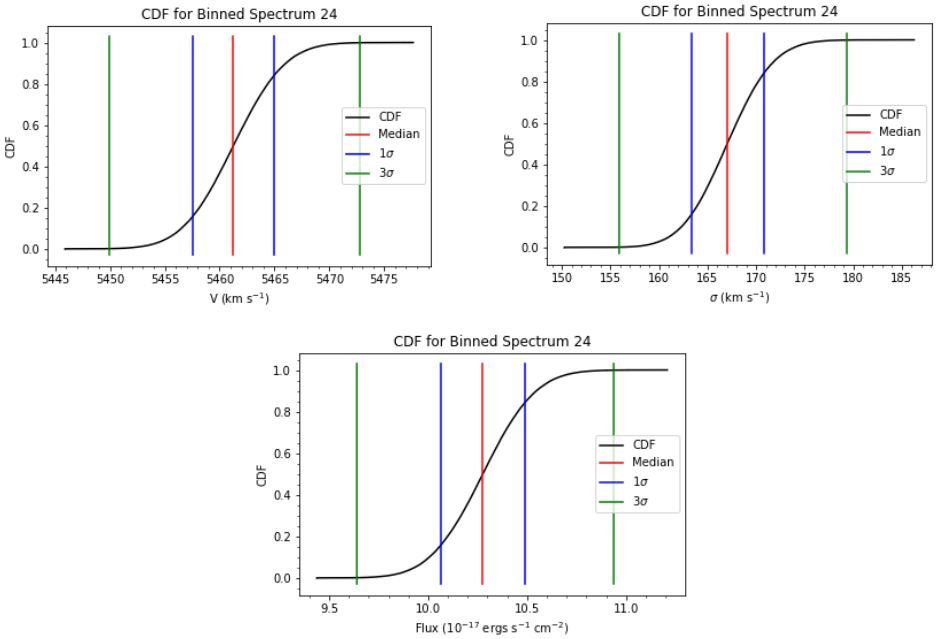

Figure 9: The top left image is our CDF graph for velocity, the top right image is our CDF graph for velocity dispersion, the bottom graph is our CDF graph for flux.

We also found histogram plots of our Monte Carlo simulation showing the velocity, velocity dispersion, and the flux, the results can be seen below.

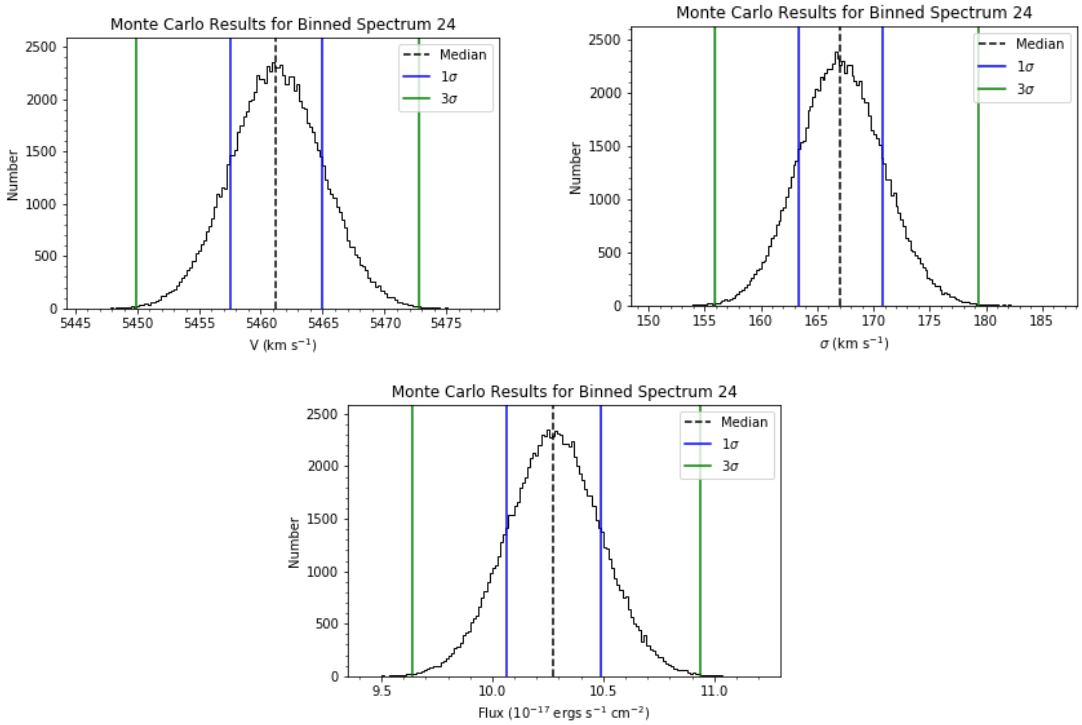

Figure 10: The top left image is our Monte Carlo histogram for velocity, the top right image is our Monte Carlo histogram for velocity dispersion, the bottom graph is our Monte Carlo histogram for flux.

Table 1. Result in our Median values from Methods A, B, and C

Method	Measurement	Median	1σ Uncertainty [upper, lower]	3σ Uncertainty [upper, lower]	Units
A	Flux	10.27	σ = 0.2108	–	10^{-17} ergs s^{-1} cm^{-2}
	Velocity	5461.24	σ = 3.7068	–	km/s
	Velocity Dispersion	167.04	σ = 3.754	–	km/s
B	Flux	10.27	[0.212, -0.211]	[0.636, -0.658]	10^{-17} ergs s^{-1} cm^{-2}
	Velocity	5461.24	[3.694, -3.737]	[11.301, -11.529]	km/s
	Velocity Dispersion	167.04	[3.701,-3.813]	[11.123, -12.255]	km/s
C	Flux	10.27	[0.212, -0.211]	[0.636, -0.658]	10^{-17} ergs s^{-1} cm^{-2}
	Velocity	5461.24	[3.694, -3.738]	[11.301, -11.529]	km/s
	Velocity Dispersion	167.04	[3.700,-3.813]	[11.123, -12.245]	km/s

The table above displays our results from Methods A, B, and C. Comparing them side by side, there seems to be no large difference in the median values between all 3 methods. Furthermore, the 1σ and 3σ uncertainties between Methods B and C are very small. However, Method A does not allow us to find the Uncertainties of our parameters at different sigma. This means that when we have data with a lot of uncertainty (possibly a faint object), it would be more useful to employ Methods B or C over Method A.

3 Fitting the Remaining Spectra Using HPRC

3.0.1 Getting an Account and Requesting Time

For the remainder of the Tutorial we will be using HPRC, the Texas A&M high performance research computing to fit all of the 286 spectra. We will be using the compiled code **fit_binspec_general.py** that includes our model calculation, along with the best-fit parameters, and the Monte Carlo simulations (with Methods B and C) to fit all of the spectra.

To use HPRC, we will have to create an account that lists our name and contact information, we must also include our Principle Investigators information (Dr. Walsh as she is teaching the course this tutorial came from). Furthermore, we must request the total hours we will use, 5000 for, ADA and 5000 for Terra. Finally, we must include a brief research description about the software needed and what we will accomplish using HPRC.

3.1 Logging Into and Setting up Ada

Once we receive our account, we can begin to log into Ada using the **ssh** command. Below is the code we used to sign in:

ssh felix.martinez123@ada.tamu.edu

Once we are logged in, we must set up a virtual environment on Ada so that we can successfully run our code. Doing this in the scratch subdirectory, (**/scratch/user/felix.martinez123/**), we made a new directory called **astr420** and loaded Python 3 onto it. Now, making the Virtual environment using **virtualenv env** and activating it, we when installed in any python packages our code required (in this case we installed numpy, scipy, astropy, and matplotlib using the **pip install** command).

We then set up our **astr420** directory by making a new directories **bin** and **tutorial4** within it, then making a new directory **ngc1275** within **tutorial4**.

Now that our pathways are setup, we downloaded a pre-written python file that full fills all of the Monte Carlo testing and fitting listed above called **fit_binspec_general_ada.py** onto our virtual machine. We then transferred it over onto our Ada account by typing the command below in our Virtual Machine terminal.

scp fit_binspec_general_ada.py felix.martinez123@ada.tamu.edu:/scratch/user/felix.martinez123/astr420/bin/

This prompted a password prompt on our Virtual Machine that required two factor authentication to be terminated successfully. We also had to to the same command for the **get_element.py** script also found in our virtual machine. If we wanted to transfer a file from Ada called **text_file.txt** located in **/home/user/felix.martinez123/** to **/Data/student/ngc1275_project/** on our virtual machine, we would have execute the command listed below on our Virtual Machine.

scp /home/user/felix.martinez123/test_file.txt felix.martinez123@ada.tamu.edu:/Data/student/ngc1275_project/

3.2 Job Submission Files and Running Jobs

We now need to make a code that sets up 286 bin directories within a new directory called **for_ada** on our virtual machine, each stored with their respective bin spectrum and the wavelength file we created in the previous sections. They must also have a file called **param_in.txt** for each bins initial conditions. We must also have 286 **cm_fitbinxx** files that will help run commands for each bin, along with a single **jobs_fitbins** file that will consist of commands to be executed simultaneously on Ada, and a **launcher_fitbins** file that we can submit to Ada to run our jobs.

Below is a screenshot of our **launcher_fitbins** file. We altered lines 4 and 6 so that we set our allowed run time to 2 hours, and wanted to use 10 nodes per core.

```
1 ##NECESSARY JOB SPECIFICATIONS
2 #BSUB -J ngc1275_fitbins
3 #BSUB -L /bin/bash
4 #BSUB -W 02:00
5 #BSUB -n 286
6 #BSUB -R "span[ptile=10]"
7 #BSUB -R "rusage[mem=2560]"
8 #BSUB -M 2560
9 #BSUB -o ngc1275_fitbins.o%J
10 #BSUB -P 082770559725
11
12 module load Python/3.6.6-intel-2018b
13 cd /scratch/user/felix.martinez123/astr420/
14 source venv/bin/activate
15 cd /scratch/user/felix.martinez123/astr420/tutorial4/ngc1275/
16 tamulauncher jobs_fitbins
```

Figure 11: Our submission file to ada with a run time of 2 hours and request to use 10 nodes per core.

If we instead put **fit_binspec_general_ada.py** into the **/scratch/user/[username]/** directory, we would actually not need to change or modify any of our files due to the "**dir_tools_py**" path information line in our **fit_binspec_general_ada.py** file, this, along with the other preset pathways in our **launcher_fitbins**, **jobs_fitbins**, and **cmd_fitbinxx** files should have ada running smoothly.

Once all of the directories and command files are created, we can move every file onto ada by compressing them down on a single tar file called **ngc1275_ada.tgz** and then using the scp command into the **ngc1275**

directory that we just created on Ada. Untaring our newly moved file onto Ada, we can finally submit our job by using the command below in the **ngc1275** directory on Ada.

bsub < launcher_fitbins

Unfortunately, when we ran our job, only half of our bins ran successfully. Strangely, the successful bins seemed to be random, with bins 285, 99, and 9, failing, but bins 218, and 35 running. As this problem was not unique to our group, we received a filed called **ada_results.tgz** from our professor that included a successful run of all 286 bins.

3.3 Transfer and Check the Results

Now that we have a file with a successful run, we can untar the file in our virtual machine and check if the run was truly successful. We will do this by writing a code that opens each of the 286 spatial bins and displays the **specfitxx.png** file onto our screen so we can check it by eye. Using the code below for our sanity check:

```
#!/bin/bash

for i in {0..285}
do
  cd /home/student/ngc1275_osiris/analysis/ada_results/bin$i
  eog specfit$i.png
done
```

Figure 12: A Bash script that opens each of our spec fit images for each bin.

We were able to get the results for bin 7:

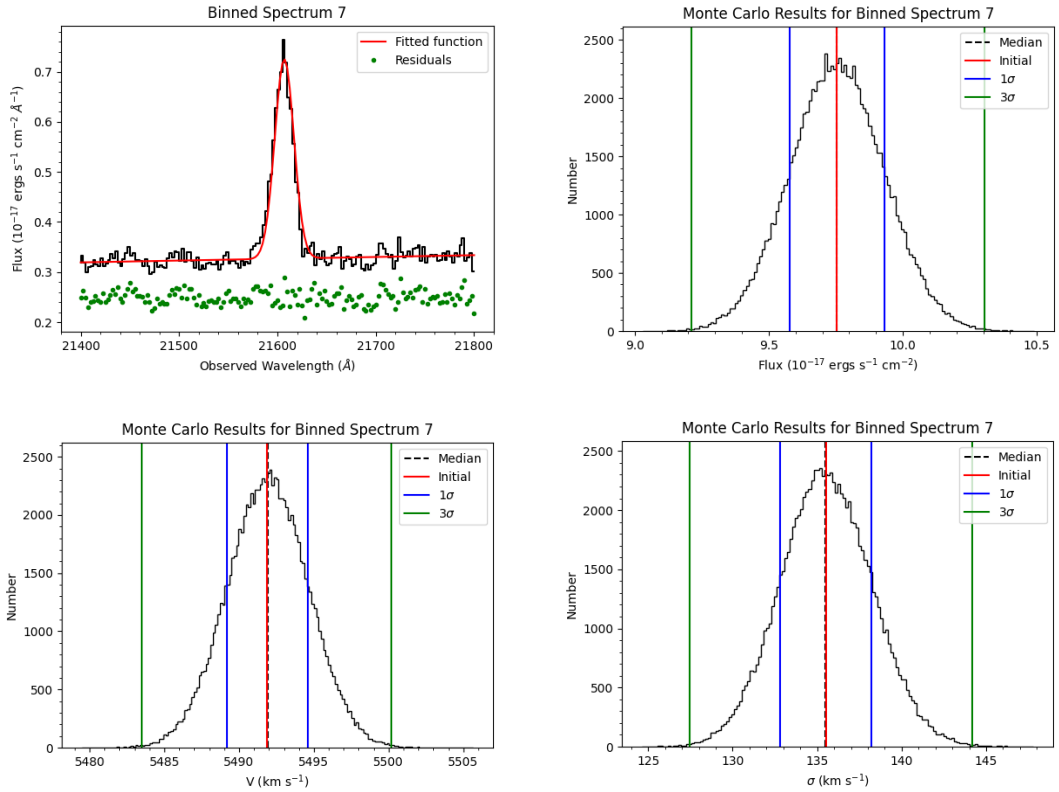

Figure 13: Where the top left image is our spectral fit for bin 7, the top right is our Monte Carlo Flux test for bin 7, the bottom left is our Monte Carlo Velocity test for bin 7, and the bottom right is our Monte Carlo Velocity Dispersion test for bin 7.

Now that that looks like it ran and fit a random bin successfully, we will compare the results from ada to the results we did manually for bin 24.

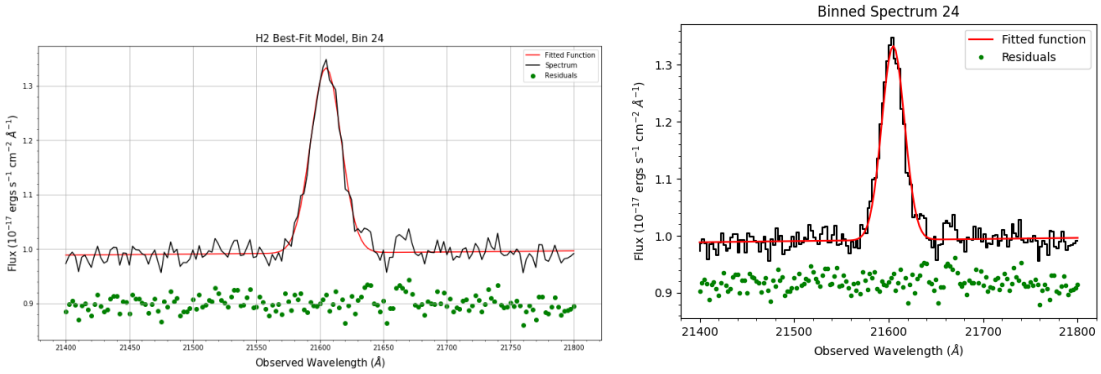

Figure 14: Comparing our manual spectral fit for bin 24 (left) to our ada spectral fit for bin 24 (right).

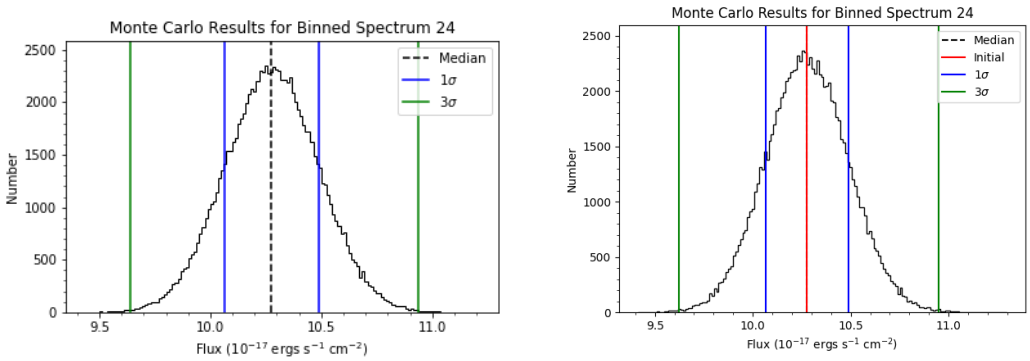

Figure 15: Comparing our manual Monte Carlo Flux for bin 24 (left) to our ada Monte Carlo Flux for bin 24 (right).

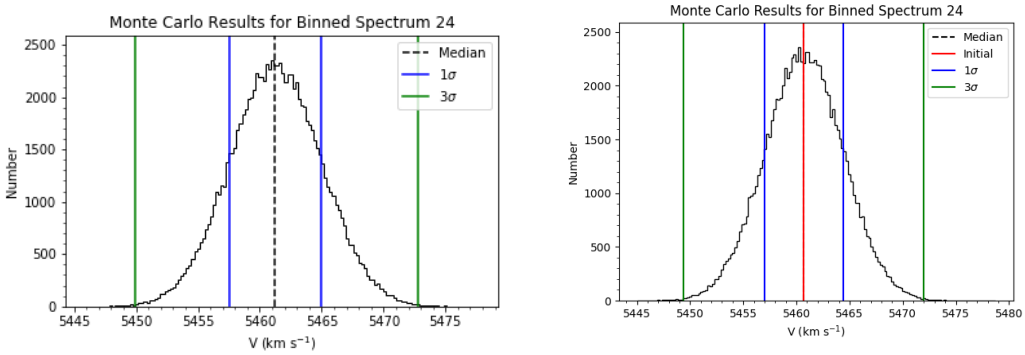

Figure 16: Comparing our manual Monte Carlo Velocity for bin 24 (left) to our ada Monte Carlo Velocity for bin 24 (right).

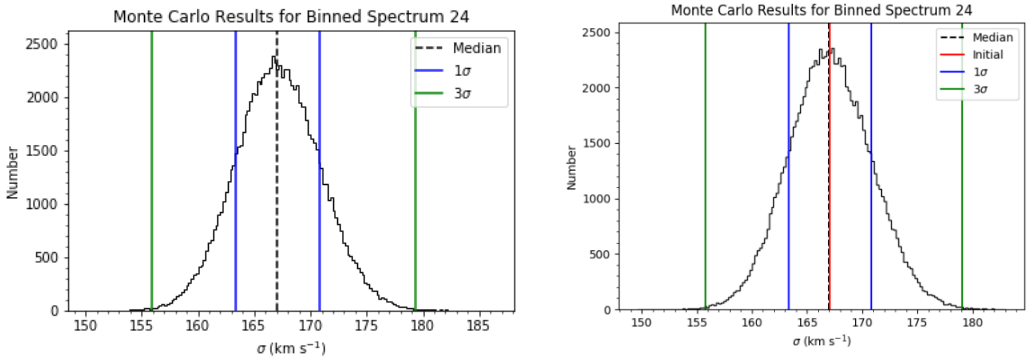

Figure 17: Comparing our manual Monte Carlo Velocity Dispersion for bin 24 (left) to our ada Monte Carlo Velocity Dispersion for bin 24 (right).

Table 2. Comparing our results from Methods C to that of Ada.

Method	Measurement	Median	1σ Uncertainty [upper, lower]	3σ Uncertainty [upper, lower]	Units
Ada	Flux	10.27	[0.212, -0.211]	[0.636, -0.658]	10^{-17} ergs s^{-1} cm^{-2}
	Velocity	5461.24	[3.694, -3.738]	[11.301, -11.529]	km/s
	Velocity Dispersion	167.04	[3.700,-3.813]	[11.123, -12.245]	km/s
Method C	Flux	10.27	[0.212, -0.211]	[0.636, -0.658]	10^{-17} ergs s^{-1} cm^{-2}
	Velocity	5461.24	[3.694, -3.738]	[11.301, -11.529]	km/s
	Velocity Dispersion	167.04	[3.700,-3.813]	[11.123, -12.245]	km/s

Comparing the above results to what we received when computing Method C manually, we can see that they do, in fact, match up correctly.

Now that the test is passed and completed, we must compile each spatial bin's **kin_xx.txt** file and create a single text file called **ngc1275_kinematic.txt** using the code below.

```
# Making a kinematics texfile with: Bin Number, Flux (10^{-17} ergs/s/cm^2),
# Flux 1-sig err down, Flux 1-sig err up, V (km/s), V 1-sig err down, V 1-sig err up,
# Sigma (km/s), Sigma 1-sig err down, Sigma 1-sig err up

# Importing modules
import numpy as np

# Naming pathways
dir_kin = '/home/student/ngc1275_osiris/analysis/ada_results/'
kinfile = 'ngc1275_kinematics.txt'

# Making the kinematics array
kinematics = []
for i in range(286):
    kin = np.loadtxt(dir_kin+'bin{}/kin_{}.txt'.format(i,i))
    array = np.array([i+1,kin[0],kin[1],kin[2],kin[5],kin[6],kin[7],kin[10],kin[11],kin[12]])
    kinematics.append(array)

# Saving the kinematics array
np.savetxt(dir_kin+kinfile,kinematics)
```

Figure 18: Our code for making a master kinematics text file.

4 Plotting the Emission-Line Kinematics

Now that we have a master kinematics file, we can use this combined with the spatial bin information in **ngc1275_osiris_bins.txt** to construct a set of maps to display information about the center of the galaxy. We will only be using 1 sigma uncertainties for this, and as we have asymmetric (but similar) error bars, we will simply average the two to produce a single uncertainty value for each spatial bin. Below are our results for mapping the emission-line velocity, velocity dispersion, and flux, as well as the velocity uncertainty, velocity dispersion uncertainty, and flux uncertainty of ngc 1275.

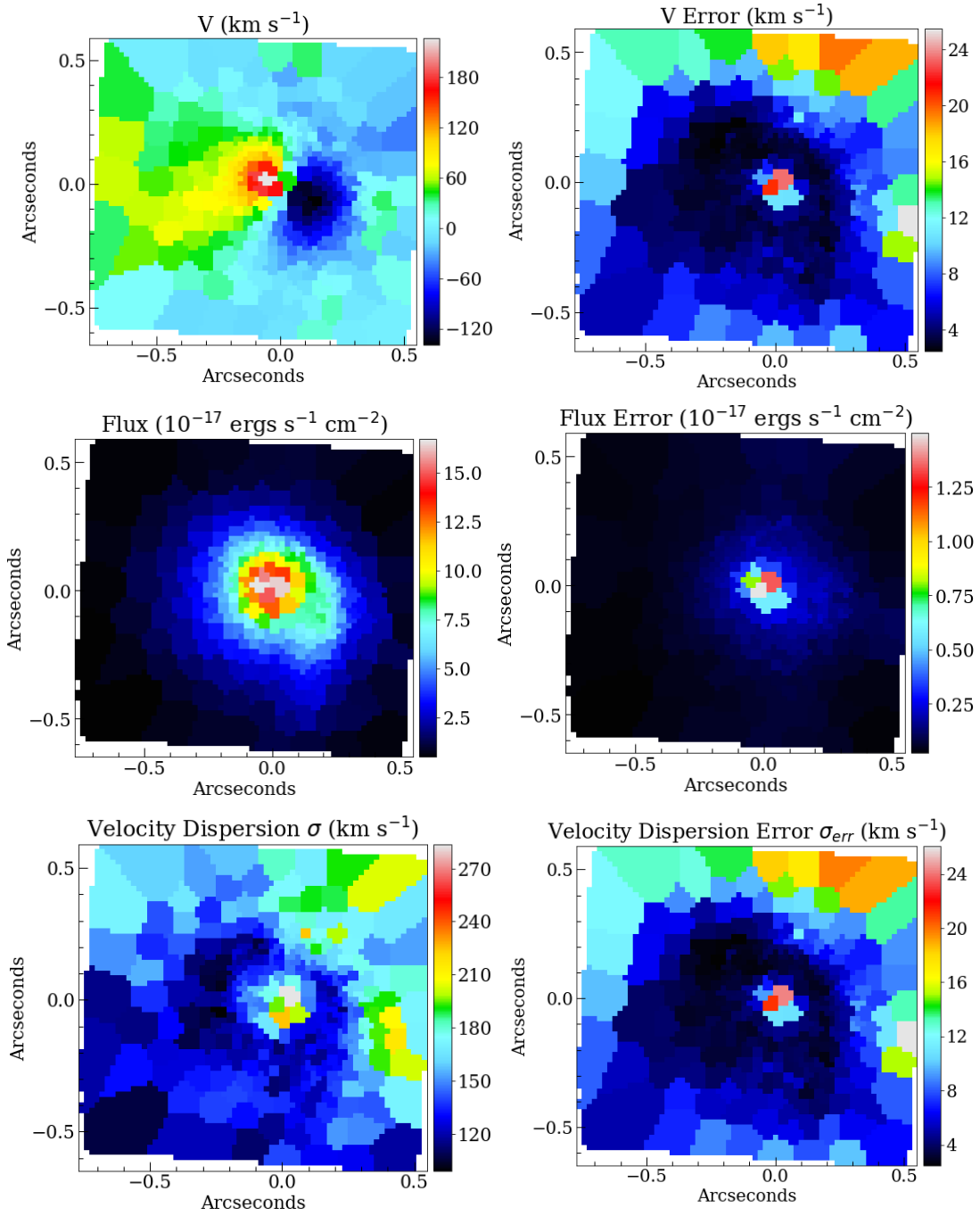

Figure 19: Where the left column consists of our mapping results, starting with velocity, then flux, then the velocity dispersion, the right column consist of our uncertainty, held in the same order of velocity, then flux, then the velocity dispersion.

When computing these maps, we had to find the galaxies systematic (median) velocity v_{sys}, we found this value to be:

$$v_{sys} = 5383.937175 \text{ km/s}$$

Something that is really cool about the maps in figure 18 is that one can see the rotation of the gas around a central mass in the velocity map. The left side of the image is coming towards us, while the right side

is shooting away, where the center is not moving towards us at all, this has a suggestion that there is gas rotating around a central mass very quickly. Furthermore, we can see when examining the flux mapping that where this center is, it is emitting a lot of radiation, it is completely drowning out any other outside sources and peaks at the center, where the gas is rotating. Finally, when looking at the Velocity Dispersion, it is fairly constant around the rotating gas, with it only changing at the center of the image, and the upper right section of the image.

5 A Rough Estimate of the Black Hole Mass

We can use Newtons Version of Kepler's 3rd law to find the mass of the Black Hole in the center of ngc 1275. This equation is:

$$p^2 = \frac{4\pi^2}{G(m_1+m_2)}a^3 \tag{4}$$

Where p is the period, a is the average orbital distance between the centers of the two bodies, m_1 is the mass of object 1, m_2 is the mass of object 2, and G is the gravitational constant. If we assume that the gas mass (m_2) is negligible and moves in a circular orbit, then we can rearrange equation (4) so that:

$$p^2 = \frac{4\pi^2}{G(m_1+m_2)}a^3 \longrightarrow m_1 = \frac{4\pi^2}{Gp^2}a^3$$

where,

$$v_c = \frac{2\pi}{p}a$$

so,

$$m_1 = \frac{v_c^2 a}{G} \tag{5}$$

We can find a value for v_c by:

$$v_{LOS} = v_{10} - v_{sys}$$

Where v_{10} is the velocity at bin 10 (v_{10} = 5485.831 km/s), and v_{sys} is the velocity we found in the section above. To find,

$$v_c = \frac{v_{LOS}}{sin(i)}, \qquad v_{LOS} = 101.894 \text{ km/s}$$

Where $i = 36°$, i is also known as the inclination angle.

So, as Ngc 1275 is 75 Mpc away, and a is $0''.0533$ at bin 10, then using the values:

$$a = 19.1217 \text{ pc} \quad \text{and,} \quad G = \frac{1}{233}\frac{\text{pc}}{M_\odot}\left(\frac{km}{s}\right)^2 \quad \text{and,} \quad v_c = 173.352 \text{ km/s}$$

The mass of the Black Hole is then:

$$m_1 = 134,557,508.467\ M_\odot$$

The black hole mass is dependent on the adopted distance to the galaxy due to our conversion rate from a in arcseconds, to a in parsecs. The further the galaxy is away from us, the larger the conversion rate would be per arcseconds. Furthermore, when estimating the black hole mass, we selected a location close

to the central black hole due to it likely being dominated by the black hole mass. However, as we choose a point further out, another mass component becomes non-negligible that is still not the gas. This is likely another stellar object (probably a star) orbiting around the Super Massive Black Hole (SMBH) at the center of its galaxy. As seen when examining our own SMBH in the Milky Way, we have observed similar characteristics of stars being flung around the central black hole.