UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

MODELING BROAD ABSORPTION LINE QUASARS WITH SPECTRAL SYNTHESIS

PROGRAM DEVELOPMENT AND ANALYSIS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

FRANCIS MACINNIS
Norman, Oklahoma
2018

MODELING BROAD ABSORPTION LINE QUASARS WITH SPECTRAL SYNTHESIS
PROGRAM DEVELOPMENT AND ANALYSIS

A THESIS APPROVED FOR THE
HOMER L. DODGE DEPARTMENT OF PHYSICS AND ASTRONOMY

BY

Dr. Karen Leighly, Chair

Dr. Donald Terndrup

Dr. Kieran Mullen

Dr. Ferah Munshi

Dr. Nathan Kaib

Acknowledgments

I would like to thank Dr. Karen Leighly for her guidance, as well as for thinking up SimBAL in the first place. I would also like to thank Dr. Don Terndrup, Joseph Choi, Cassidy Wagner, Adam Marrs, and Collin Dabbieri for all of the snippets of code and ideas they provided, whether directly or through their own projects, posters, and presentations. I received support from NSF Astronomy and Astrophysics Grant #1518382. Finally, I would like to thank my fiancée and my cats, for providing emotional support and loud meowing, respectively.

Table of Contents

List of Figures

Abstract

Quasars with broad absorption lines (BALQSOs) are often the target of outflow studies because these features are clear markers of energetic winds. Within this population is a class known as FeLoBALs, named for the FeII broad absorption lines they have along with the low ionization lines of the LoBAL class. They are a prime target for quantifying outflow strength because their multiple broad absorption lines allow for good constraints on outflow properties.

To analyze these objects, SimBAL, a program for creating synthetic spectra from a grid of Cloudy models, is used. It matches real spectra with the synthetic to determine physical parameters of the lines, and thus the outflows, using all of the information the spectra contain. It is still in development, but it is planned to be released to the larger quasar wind community when it becomes feature complete.

To aid in SimBAL's development and improvement, I have performed various alterations and analyses to aspects of the program, from how it stores and processes the Cloudy information to how it explores parameter space. I have presented several tests and features that will improve SimBAL's efficiency, accuracy, and usability, while also exploring facets of quasar outflows along the way.

Chapter 1

Introduction

1.1 Quasar Background

At the center of almost all large galaxies lies a supermassive black hole (SMBH). These are black holes which are five to nine orders of magnitude more massive than stellar mass black holes. In some galaxies, such as our own, the galactic nucleus is quiescent. Occasionally a star or gas cloud will pass too close to the SMBH and produce some emission as it interacts, but for the most part the only indication of these $10^5 - 10^9 M_\odot$ objects is from the orbits of nearby stars. Many other galaxies, however, are host to quasars.[1]

The current consensus is that a quasar is a central SMBH surrounded by a disk of accreting material and a dusty torus (fig. 1.1). This model is the result of decades of effort to explain many distinct observational classes as a single type of object viewed from different angles. A blazar is a quasar viewed through the

[1] Quasars are sometimes also known as Active Galactic Nuclei (AGN). This comes from the fact that AGN and quasars were considered different objects before the host galaxies of quasars were observed. The current distinction is a somewhat arbitrary luminosity cutoff, where $L_{bol}(quasars) > 10^{46} \frac{erg}{s}$.

jets, the difference between Type I and Type II Seyfert galaxies[2] is whether the observers line of sight passes through the torus, and so on.

There are many properties that make quasars interesting. They are the most luminous continuous sources in the universe. They often outshine their entire host galaxies.[3] Some produce relativistic jets, and some exhibit strong variability. Some exhibit signs of winds. It is this last property that is most important to this paper.

The presence of winds is inferred from the observation of broad absorption lines (BALs) in some quasar spectra[4]. These BALs are uniformly blueshifted with respect to emission lines, and this coupled with their broad nature indicates the presence of winds coming out of the central engine. There are three broad categories of Broad Absorption Line Quasars (BALQSOs): HiBALs, LoBALs, and FeLoBALs. Most BALQSOs are HiBALs, which have only high ionization BALs such as CIV. Some objects, LoBALs, have low ionization BALs like MgII in addition to the high ionization lines. An even smaller subset, FeLoBALs, have FeII BALs along with all the others. Because they have so many BALs, these rare objects contain an incredible amount of information about their winds in their spectra. Additionally, the large number of iron BALs have many different critical densities, and so are very sensitive to the outflow properties.

Shifting gears for a moment, I would like to introduce several important observational results which will relate back to why quasar winds are significant. There are several empirically determined correlations between properties of the central SMBH and properties of the host galaxy. The most notable is the M-σ relation (Ferrarese and Merritt 2000), which is a strong linear correlation between the mass

[2]A Seyfert galaxy is now known to be a less luminous quasar.

[3]This is actually the origin of the word quasar. It is a contraction of quasi-stellar radio source, which the first of these objects to be observed were called because their host galaxies could not be detected, and it was uncertain if they were even beyond our own galaxy.

[4]Around 10-20%

of the central SMBH and the velocity dispersion of the bulge of the host galaxy, which is is turn related to the mass of the galaxy.

These empirical correlations produce a bit of a theoretical problem. The quasar itself is a fairly compact region, on the order of a parsec across, while the host galaxy is usually tens to hundreds of *kilo*parsecs across. The correlations imply that there is communication between the two, and while the quasar emits enough energy to unbind the galaxy (King and Pounds 2015), the mechanism by which the energy is transferred is not immediately apparent.

There is further evidence for some sort of feedback in the luminosity function of quasars [5]. The current most widely accepted model for the universe (hierarchical cold dark matter) predicts that the most massive objects form last. However, when looking across the age of the universe, the number of the most luminous and massive quasars peak at redshifts of 2 to 2.5 (Fabian 2012). This implies that something is quenching quasar activity.

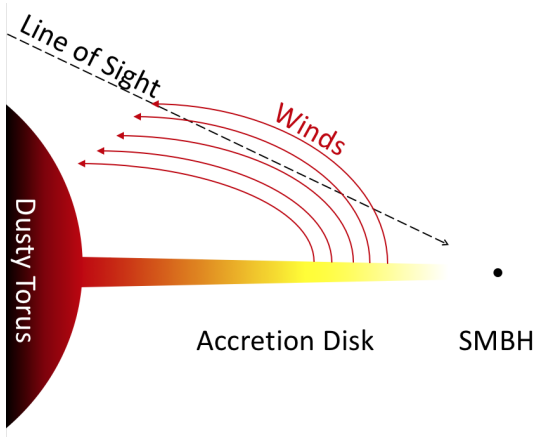

Figure 1.2: A detail of the quasar cross section, showing winds coming off of the accretion disk. The marked line of sight would result in BALs in the quasar's observed spectrum.

A promising potential explanation of feedback is quasar winds. These winds could be carrying energy from near the SMBH out to the interstellar medium (ISM), either heating up the ISM so it cannot collapse and form stars or pushing the ISM out of the galaxy entirely. The question then becomes whether or not the

[5]The quasar luminosity function is the distribution of quasar density per comoving volume as a function of intrinsic luminosity and redshift (Green 2013).

winds are energetic enough, and if they reach far enough out to interact with the ISM. Clearly the winds need to be quantified.

Quantifying the winds means measuring BALs. This is easier said than done, however, as their broad nature creates a number of hurdles. Closely grouped lines will blend into one another, making them difficult to resolve. The continuum will also be difficult to locate. In the most heavily absorbed spectra there may be almost no continuum points. This is where the SimBAL program comes in.

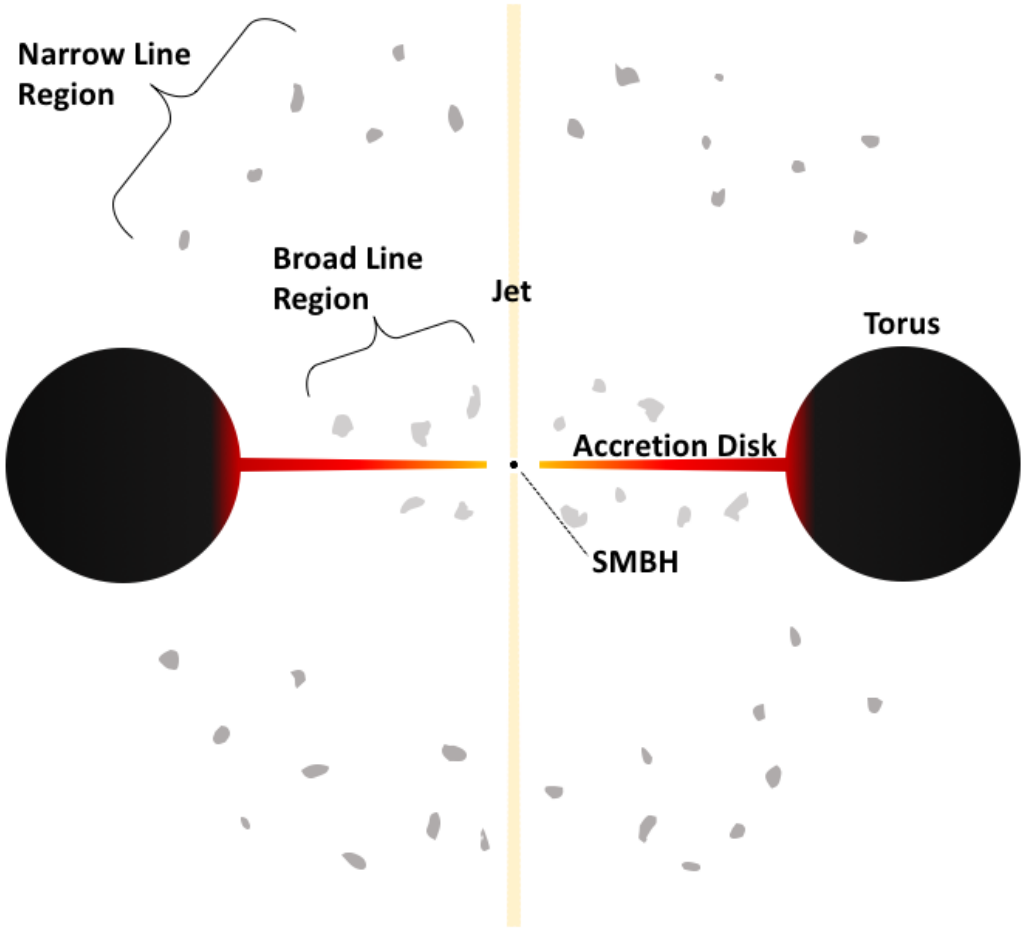

Figure 1.1: A cross section schematic of the current quasar model (not to scale). Jets are not always present. It should be noted that much of this diagram is still debated, especially as concerns the dusty torus. It is still uncertain whether or not it feeds directly into the accretion disk, if it shares the same plane as the disk, and what mechanism gives rise to it.

1.2 SimBAL Background

SimBAL is a program written in Python to model BALQSO spectra by creating synthetic spectra from a grid of Cloudy ionic column densities (Leighly et al. 2018). The original idea and spectrum making code was created by Dr. Karen Leighly[6], and it has since been modified and expanded Dr. Donald Terndrup[7], Cassidy Wagner[6], Adam Marrs[6], Hyunseop Choi[6], and Collin Dabbieri[6], as well as myself.

Cloudy is a well established photoionization code which "models the ionization, chemical, and thermal state of material that may be exposed to an external radiation field or other source of heating, and predicts observables such as emission and absorption spectra" (Ferland et al. 2013). We use it to model the absorption caused by the gas in the outflow being exposed to the ionizing radiation from the accretion disk.

The ionic column density grid is three dimensional, and is created by varying ionization parameter ($\log U$), density ($\log n$), and column density ($\log N_H$). The ionization parameter is a measure of the balance between ionization and recombination. The higher the ionization parameter, the more ionized the gas. The density is simply the number density of electrons. The column density is not actually used directly. The ionization parameter subtracted from the column density ($\log N_H - \log U$) is used instead. This effectively measures the column density relative to the hydrogen ionization front, which comes in at around 23.2 in this parameterization. This makes for a more useful diagnostic for our purposes, since when this parameter is held constant, the species of lines seen will be mostly constant across ionization parameter and density.

This grid is then combined in the SimBAL code with three additional param-

[6]The University of Oklahoma
[7]The Ohio State University

eters: velocity offset, velocity width, and log covering fraction (Figure 1.3). The velocity offset and width are fairly self-explanatory, but the covering fraction is a bit more nuanced. The covering fraction is parameterized as a power law $\tau = \tau_{max} x^a$, where τ is the integrated opacity of the line, τ_{max} is a constant times oscillator strength, wavelength, and column density (see Savage and Sembach 1991, or §5 of this paper), $x \in (0, 1)$ represents the projection of the two dimensional continuum source onto a normalized one dimension, and $\log(a)$ is the parameter to be fit. The actual covering fraction can be gotten by taking $\frac{\tau_{max}}{1+a}$. This means that when $\tau_{max} = 1$, $a = 0$ is full covering, $a = 1$ is 50% covering, $a = 2$ is 25% covering, and so on. These three kinetic parameters combine with the three gas parameters to determine the line structure of the synthetic spectrum.

The emission lines are modeled using eigenvectors obtained by using principal component analysis (PCA) on a sample of unabsorbed spectra. PCA is a method to reduce the dimensionality of a parameter space, and is discussed in more detail in Chapter 3. The short description is that it reduces dimensionality by changing the basis to one where most of the variability can be described using the first few dimensions. This, along with a power law for the continuum, adds an additional nine parameters to the simulation: power law normalization, power law index, amplitudes for the mean spectrum and four eigenspectra from the PCA, emission line full width half maximum (FWHM), and reddening of the spectrum due to dust in the host galaxy.

All of these parameters make for a vast parameter space, full of diverse spectra. How can SimBAL sample this in a fair way without taking an unreasonable amount of time? The answer comes in the form of a class of algorithms known as Markov Chain Monte Carlo (MCMC). MCMC is a class of algorithms that uses Markov chain methods to efficiently sample a parameter space and give a representative

distribution without having to visit every point. The details of the algorithm are covered in Chapter 4, but the general overview is that SimBAL starts somewhere in parameter space, picks a new position to try to step to, creates a probability of taking the step based on a likelihood value at both the current and new positions, and then randomly takes the step or stays put based on the probability previously calculated. The likelihood we are using is a χ^2 comparison of the synthetic spectrum with the observed spectrum. The current implementation of SimBAL uses the Python package `emcee` (Foreman-Mackey et al. 2012) for its MCMC.

At the moment SimBAL is only in use by Dr. Leighly's research group. However, it is planned to eventually release to the quasar community at large. Before it is ready for general use there are still parts that need adjustment or analysis, and in this paper I address several of them. Chapter 2 describes taking the resolution of the detectors capturing real spectra into account when generating model spectra to match. In Chapter 3 I describe the results of a comparative analysis of different PCA bases on the model fits. In Chapter 4 I compare different MCMC methods to determine which are best suited for SimBAL. In Chapter 5 I demonstrate the changes implemented in how SimBAL stores the ionic column densities, as well as the necessary balance between runtime and memory usage for such a complex program. Finally, in Chapter 6 I discuss the implications of my results for SimBAL and next steps to consider in the ongoing development of this robust spectral synthesis code.

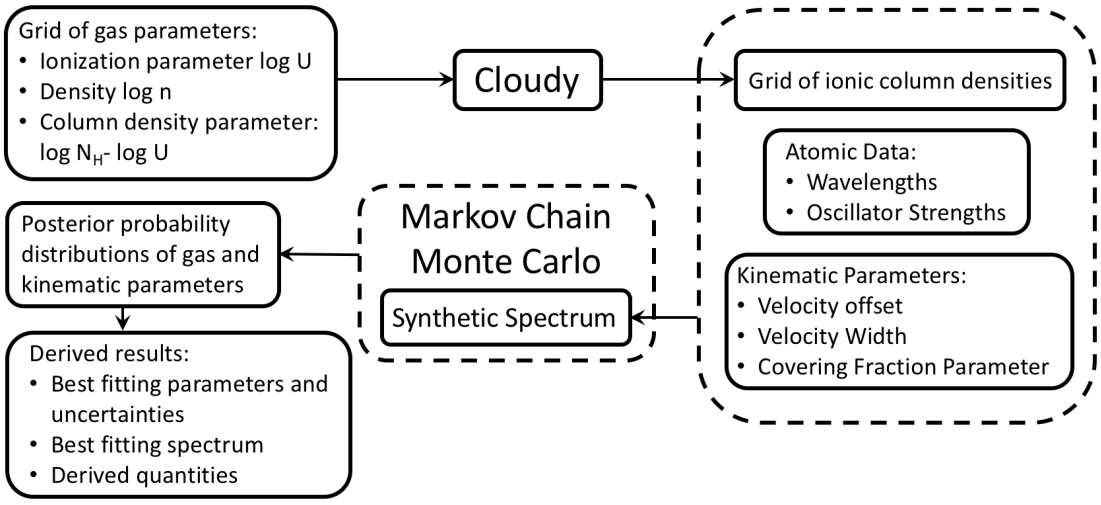

Figure 1.3: The SimBAL algorithm visualized as a flowchart. Note that this does not include the continuum modeling, which is only used to aid the absorption modeling and is not used to derive any physical quantities directly.

Chapter 2

Resolution Adjustment

2.1 Introduction

SimBAL creates its synthetic spectra partially based on Cloudy models (Ferland et al. 2013), which simulate the conditions of the gas to produce column densities that are then used to calculate line depths. This ends up producing a simulated spectrum based purely off of physical parameters. However, real spectra do not look like this because they are captured with real instruments which have limited resolution. This results in a smearing of the actual spectrum, with some of the flux of each pixel being spread to neighboring pixels.

This is not an unquantified process, and in fact the data files come with the resolution information. That means that the instrument resolution can be accounted for when making synthetic spectra to compare, in a process called RMF (originally meaning resolution matrix file in X-ray astronomy, I use it here to mean the convolution process detailed in the §2.2). This is important because without accounting for it, SimBAL will use physical parameters to try and explain the width of the lines, some of which is in fact due partially to the resolution and not any physical

property of the quasar.

2.2 Implementation

The data we are using comes from the Sloan Digital Sky Survey (SDSS), an imaging and spectroscopic survey using a 2.5m optical telescope at the Apache Point Observatory in New Mexico. This survey has produced hundreds of thousands of quasar spectra to date. The data files, along with the standard measurements of wavelength, flux, inverse variance, and the like there is also a measurement of the wavelength dispersion in pixels. This is recorded as the sigma of a fitted Gaussian centered at each pixel. The Gaussian describes how the flux density incident upon each pixel is smeared across the neighboring pixels (figure 2.1). Integrating the Gaussian across the width of each pixel gives the fraction of the flux that will end up in that pixel.

Applying this to a spectrum is actually fairly simple. The resolution matrix is an almost diagonal matrix, with a width and height equal to the number of pixels. This is because it is mapping the flux intended for each pixel back onto the pixels of the measurement. If the instrument had perfect resolution, the matrix would be an identity matrix because the flux at each pixel would cleanly map onto only that pixel. Of course this is not the case, and that introduces off-diagonal elements to the matrix. The smearing profile is a Gaussian, and since Gaussians have infinite tails this means that the matrix could be completely full, with even the most distant pixels smearing slightly into each other. In practice this is unnecessary, as the amount of flux being contributed becomes negligible after a handful of steps away from the central pixel. For my implementation eight neighboring pixels are considered (four on each side). This is accurate to around 0.001% of the flux for

the range of dispersion values we encounter.

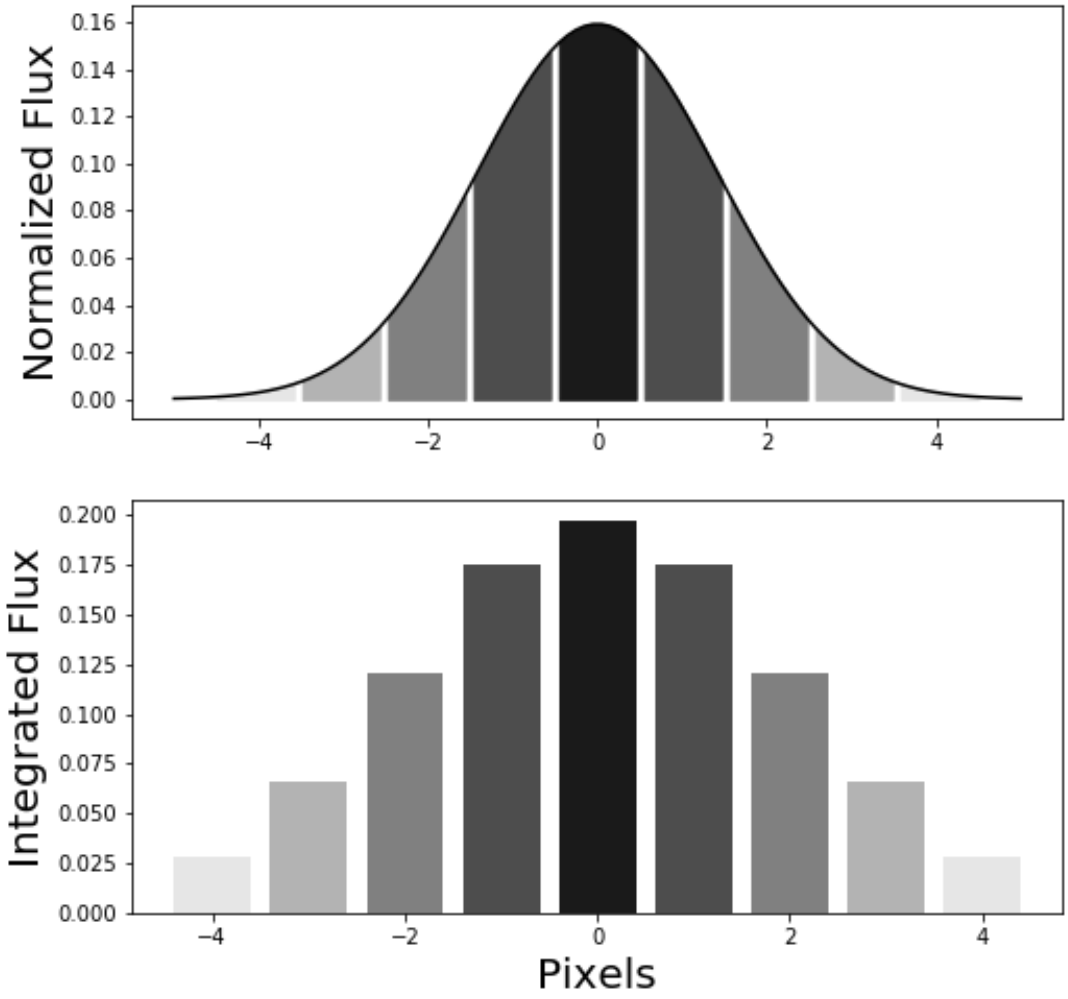

Figure 2.1: The way one unit of flux is distributed across neighboring pixels. The top plot shows an example of a pixel with a wavelength dispersion of 2. Each bar under the curve shows how much of the unit of flux is ending up at each pixel. The bottom plot shows the integrated level of flux at each pixel. The height of each bar is the area of the similarly colored bar above.

Once the resolution matrix is made, it is a simple matter of matrix multiplication by the unsmeared model spectrum to obtain the smeared model spectrum. While simple, this is a rather large operation, and it needs to be done every MCMC step. Assuming a spectrum length of 3500 pixels, this comes out to 85,737,750,000

basic operations per step. When implemented this led to an eight-fold increase in runtime, making it of questionable value. There is a workaround, however. While the matrix is huge, it is very sparse. Furthermore, it has a known shape, with the same elements nonzero for every spectrum. This means that the significant operations can be coded in directly, cutting out the billions of unnecessary operations involving only zeros. In fact, the true matrix never needs to be made. The nonzero elements of each row can just be stacked into a much smaller array. This implementation increases runtime by about 27%, a significant improvement over the previous method.

A final detail to be aware of is what happens at the edge of the spectrum. If the ends are not accounted for, flux can be smeared right off the spectrum and be lost. This results in a small but noticeable dip in the spectrum, which can be problematic if there is real absorption of interest near the fringes. All that needs to be done to avoid this is to take the flux that would be smeared onto the nonexistent pixels and move it to the nearest actual pixel. It is possible a deeper understanding of what physically happens at the edge of a spectrograph would suggest a more realistic way to account for this, but the precision that might be gained is not currently worth the time investment.

2.3 Analysis

Two FeLoBAsL from the Farrah et al. (2012) sample: SDSS J091854.48 + 583339.6 and SDSS J101927.36 + 022521.4, hereafter referred to as J0918 and J1019, respectively. Both were modeled with resolution smearing and without. The results are presented here.

I start by analyzing J0918. Comparing two modeling runs of the same object,

one with resolution smearing implemented and one without, demonstrates the importance of taking resolution into account. Most of the parameters are unchanged (figure 2.2). If this was not the case, it would imply that the resolution implementation was interfering with the modeling in some undesirable fashon. The velocity width, on the other hand, shows a dramatic shift (Figure 2.3). This is expected, as the line width is what smearing primarily affects. However, the size of the offset highlights the importance of accounting for detector resolution. The posterior is slightly tighter as well, although it is a small enough difference that I am hesitant to claim it as an improvement.

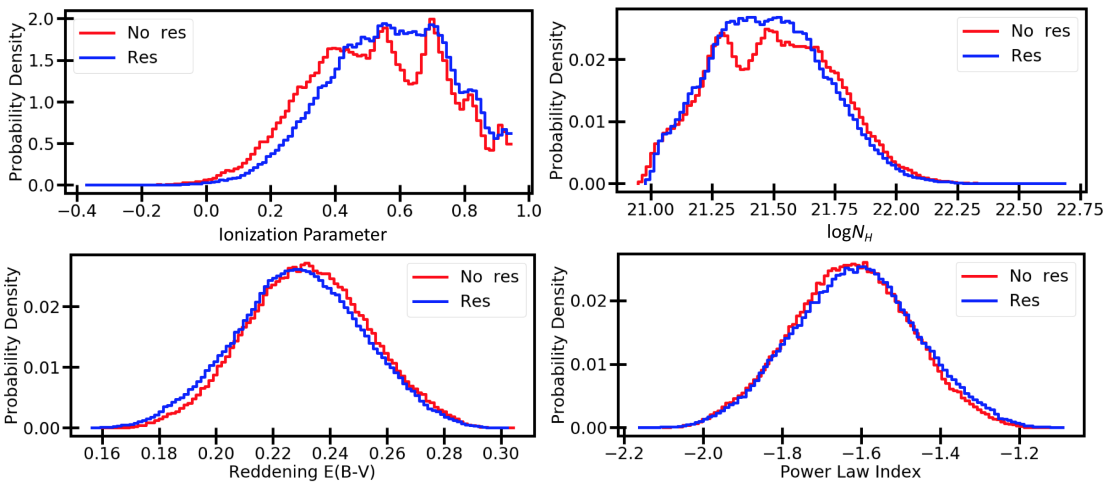

Figure 2.2: The posterior distributions for several parameters for the J0918 modeling runs, both with (blue) and without (red) resolution smearing. These posteriors can be seen to be mostly unaffected by implementing resolution smearing, which is what we would hope.

The significance of this is not simply computational. Some objects exhibit narrow lines with low column densities, which look like they could be caused by absorption from the interstellar medium (ISM) of an intervening galaxy along our line of sight. If this were the case they should have complete covering, since the quasar would be a point source from the intervening galaxy's point of view. If

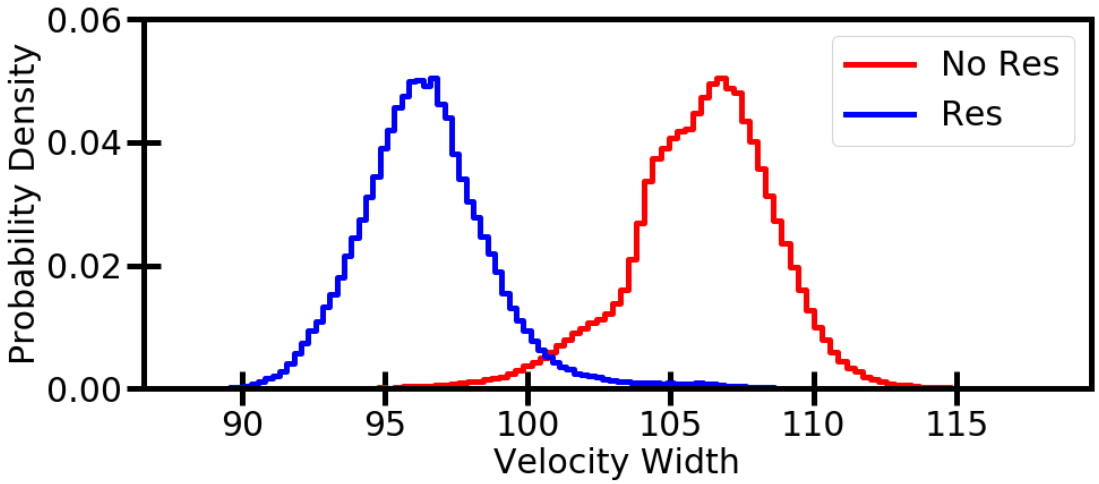

Figure 2.3: The posterior distributions of the line widths for the J0918 modeling runs, both with (blue) and without (red) resolution smearing. As expected, the posterior with the resolution smearing implemented covers lower velocity widths, as some of the line width is explained by limited resolution.

they are intrinsic to the quasar, on the other hand, they would probably exhibit partial covering. The lines in question have line ratios indicative of partial covering. However, the detector's finite resolution will smear the lines, altering the measured line ratio such that the ratios will appear to indicate partial covering even when there is full covering. This is where the RMF comes in. Without it, we cannot trust the partial covering result. With it implemented, the covering fraction is slightly closer to full, but still definitely partial (figure 2.4). We can now confidently say the lines are intrinsic, and keep them in our outflow sample.

Moving to J1019, I predict that the differences between the two implementations will be smaller. This is because J0918 has relatively narrow lines for an FeLoBAL. J1019, on the other hand, has quite broad lines (about four times as broad) and so would be less affected by having a small part of its line widths explained by limited resolution. Figure 2.5 shows that this is indeed the case. The posteriors lie mostly on top of one another, as the small change in intrinsic veloc-

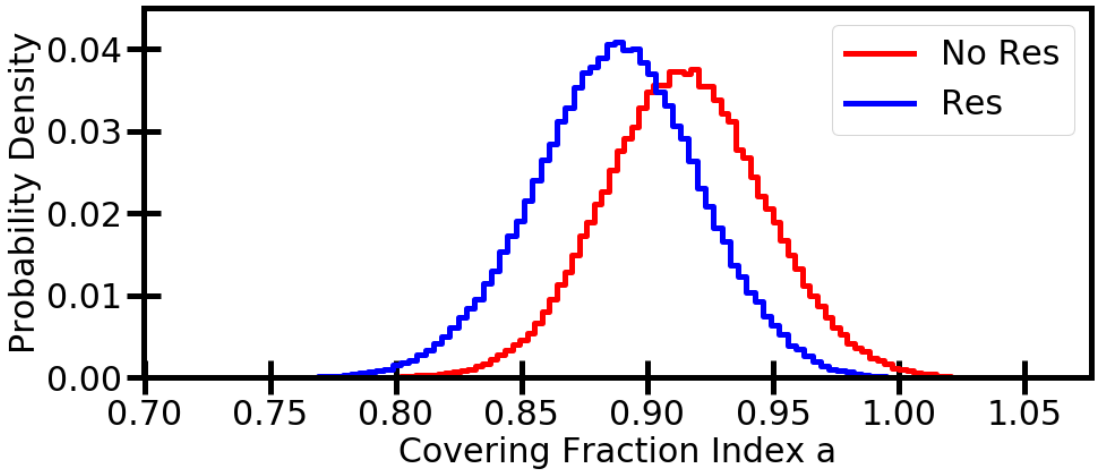

Figure 2.4: The posteriors of the covering fraction of J0918. Recall that a lower index means higher covering. The covering fraction is lower when resolution is not accounted for, as expected. Otherwise, the resolution smearing is attributed to partial covering.

ity width due to resolution smearing is negligible with lines this broad. Figure 2.6 demonstrates that this behavior holds for the covering fraction as well.

The reduced effect of resolution smearing for objects with very broad lines is important to be aware of because a SimBAL run with resolution smearing takes about 25% longer than one without. This means that it would be desirable to skip the resolution smearing if it will have a negligible effect. This analysis shows that for objects with very broad lines, resolution smearing can be safely ignored.

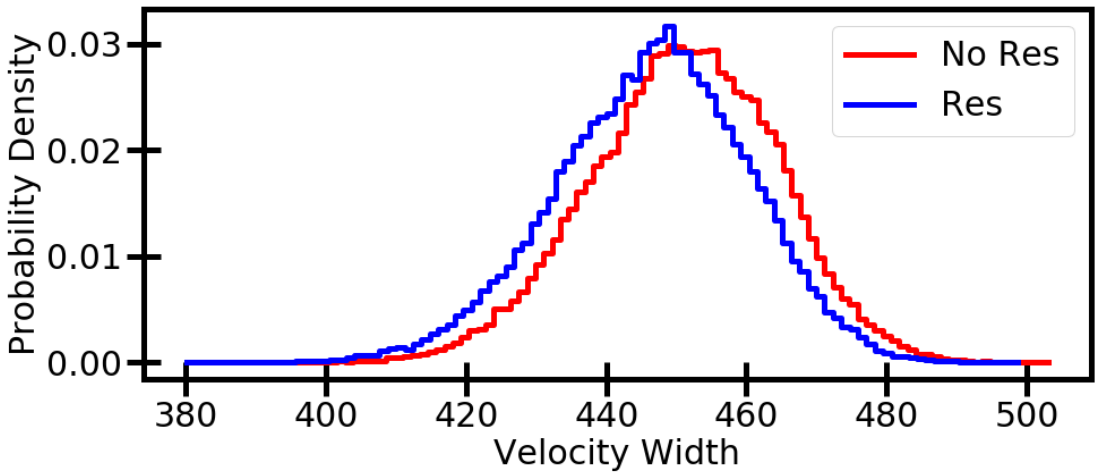

Figure 2.5: The posterior distributions of the line widths for the J1019 modeling runs, both with (blue) and without (red) resolution smearing. Compared to the same plot for J0918 (Figure 2.3), it can be seen that J1019 has lines over four times as broad. The difference between the posteriors for J1019 are almost nonexistent, as might be expected for such broad lines.

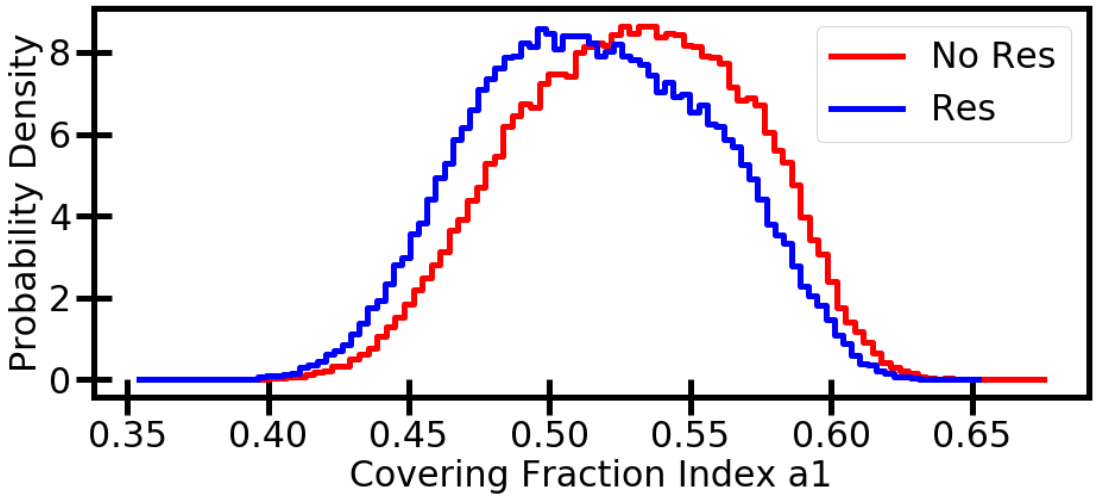

Figure 2.6: The posteriors of the covering fraction of J1019. Recall that a lower index means higher covering. The covering fraction is lower when resolution is not accounted for, as expected. Otherwise, the resolution smearing is attributed to partial covering.

Chapter 3

Cluster Dependence

3.1 Introduction

The information we are interested in is mostly contained in the absorption lines. However, accurate modeling of absorption lines requires modeling of the continuum and emission lines as well. The depth of the absorption lines depends on the placement of the continuum. Moreover, the continuum in this case includes both the contribution of the accretion disk and the emission lines, and the shape of both can vary from object to object.

To be able to model the many continuum phenotypes, the SimBAL group turned to principle component analysis (PCA). PCA is a method of dimensionality reduction that involves making a new basis for the data. The first eigenvector is set along the direction of greatest variance. The second is set along the greatest remaining orthogonal variance, and so on. In this way the eigenvectors are ordered by the amount of variance they explain. Then, hopefully, many of the later eigenvectors can be ignored without losing information, and the result will be a much lower dimension parameter space.

There are some issues with using standard PCA to model our quasar continua. Firstly, it assumes linearity in our data. Each point of a spectrum is not at all independent of its neighbors. If a point is part of a broad emission line, then the points next to it are very likely part of that same line, so this assumption fails immediately. Secondly, standard PCA has no way too account for missing or noisy data. Most of our quasar spectra have at least one or two bad points, and it is not uncommon to have noisy regions near the edge of the bandpass.

To solve the second problem, a special type of PCA known as expectation maximization principal component analysis (EMPCA) was used (Bailey 2012). It works on the same principles as regular PCA, but each point has a weight assigned to it. For our implementation we used $1/error^2$ for weights, where the error came from the data file and missing points assigned a weight of zero.

To negate the first issue, k-means clustering was implemented. K-means clustering breaks the data into a specified number of clusters by minimizing the distance between each point in the cluster and the cluster's mean. K-means with five clusters[1] was run on the coefficients needed by the PCA to reconstruct each spectrum, so each cluster member would have similar projections. Then only the cluster mean and the first four eigenvectors are kept for each cluster. Without the clustering, more eigenvectors would need to be used to accurately reconstruct the spectra.

We started with a sample of SDSS DR4 spectra that were identified by eye to have relatively narrow MgII emission lines and no significant absorption lines ($> 11,000$ objects). These were cleaned of narrow absorption lines and noise spikes, and fit between 2200 and 3200 Angstroms with a model consisting of a power law, individual emission lines including MgII, and an FeII template derived

[1]The number of clusters was chosen arbitrarily.

from a high signal to noise ratio Hubble Space Telescope spectrum of I Zwicky 1(see Leighly and Moore (2006) for details). Objects with MgII FWHM $>$ 3300 km/s were removed from the sample, leaving 5307 objects. Finally, the ones with signal-to-noise ratios less than the median near 2500 Angstroms were removed, leaving 2626 spectra. The continuum was estimated based on line-free bands and subtracted. Then the EMPCA and k-means clustering was performed.

BALQSOs are sometimes observed to have strong FeII emission and high FeII/MgII ratios (for examples of such spectra, see Leighly and Moore 2006). For example, Yuan and Wills (2003) observed z$\sim$2 BALQSOs in the infrared band and found that they have strong FeII and weak [OIII]. Such objects are somewhat rare in our sample, so we isolate a set of 75 objects that have high FeII/MgII ratios and performed a principal components analysis on them. This is the source of the sixth cluster seen in the analysis to follow. Figure 3.1 shows the cluster means.

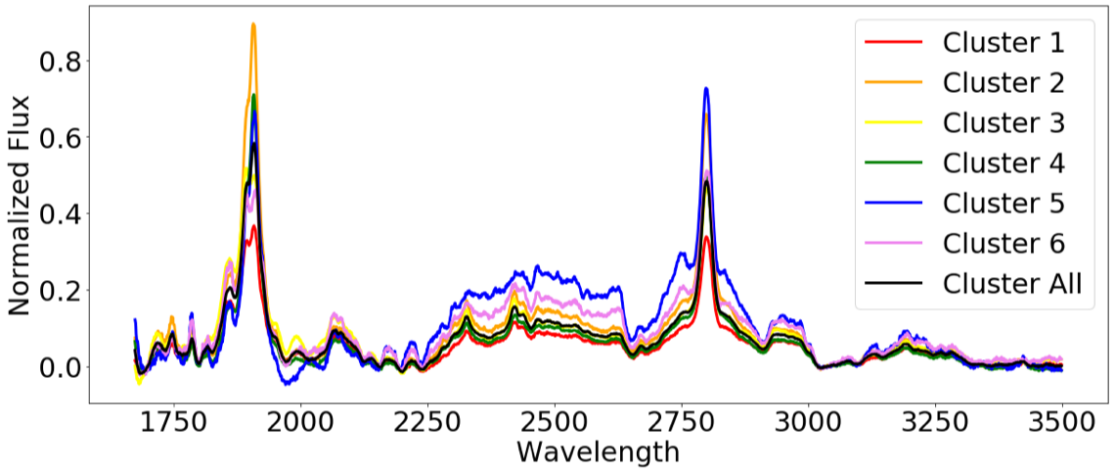

Figure 3.1: The means of each of the six clusters, as well as the overall parameter space (labeled Cluster All) are plotted here. Note that these clusters are purely empirical, so these cluster means do not necessarily correspond to physically meaningful distinctions between the objects.

All of this is well and good, but how does clustering the PCA affect our mod-

eling? Each object we model has a continuum that will be best constructed from one cluster's eigenspectra, but that doesn't mean it cannot be modeled by another cluster. Hopefully there is some way to statistically distinguish between clusters to determine the best one, as our current method is an educated guess based on what we know the cluster means look like. Additionally, it would be useful to know what, if any, systematic error is incurred by cluster choice. That is why in this chapter I present a comparative analysis of modeling an object with all of the clusters.

3.2 Analysis

Two FeLoBALs were modeled: SDSS J091854.48 + 583339.6 and SDSS J101927.36 + 022521.4 (the same objects used in Chapter 2). These will be referred to as J0918 and J1019 for the rest of the chapter. The objects were modeled with SimBAL using each of the 5 standard clusters, as well as the extreme iron cluster.

Figure 3.2 shows the maximum likelihood models from each of the six runs for J0918. Recall that for SimBAL, the likelihood is a χ^2 measurement between the model spectrum and the observed spectrum. The models all appear to fit the data well, and perhaps more importantly are all very similar to one another. This is a promising result, as it suggests that choosing the "wrong" cluster does not significantly impede SimBAL. However, the real test comes in how the posteriors compare to one another, since they are what the quasar properties are eventually derived from.

The likelihood sounds like an intuitive diagnostic, as it is how SimBAL measures the quality of a given point in parameter space. If one cluster is producing higher likelihoods than all of the others, then it is the cluster that should be used. Figure

3.3 shows that the runs do in fact produce distinct likelihood ranges. This means that SimBAL considers the quality of models being produced by the different clusters distinct, which is what we need to determine the appropriate cluster to use.

If we were to use likelihood as a cluster diagnostic, it would appear that the extreme iron cluster (Cluster 6), would be the best choice for this modeling run. Worth noting is the fact that this object was previously modeled with Cluster 5, based on our "educated guess" method. According to the likelihood, this is the cluster that produces the second best models. This is not too bad for a person just comparing mean spectra to the data, but there is little reason to accept second best when such a good diagnostic is built into SimBAL already. The only reason to use the guessing method is that it is much faster.

Looking at Figure 3.4, there is a clear cluster dependence for some of the absorption parameters. This confirms that there is a systematic error associated with cluster choice, so guessing at the appropriate cluster is probably unacceptable.

J0918 has a pretty noisy spectrum, so it might be tempting to dismiss these results as isolated to objects with low signal to noise ratios. To check this, we can look at the the posteriors for J1019 as well. J1019 is an overlapping trough object, meaning that its BALs are so broad that there is little to no unabsorbed continuum left. It also has a good signal to noise, so if it experiences cluster dependencies as well, it makes the case stronger for this being a property of the clusters themselves.

Again the cluster runs all produce fairly good models (Figure 3.5). There is a little bit more variance between the models this time, likely due to the fact that overlapping trough objects are harder to model in general and so have more uncertainty in the models. Looking at Figure 3.7, the same posterior dependence on cluster can be seen. In fact, the dependence may be more consistent and

pronounced in this object than J0918. This seems reasonable with the larger difference in likelihoods.

Since likelihoods are even more separated for this object, choosing the correct cluster is even more important. This confirms that the likelihood varies with cluster, and that the likelihood can be used to determine which cluster should be used to model any given object.

There seems to be one major issue with using the likelihood as a diagnostic: in order to tell which cluster should be used in a SimBAL run, one has to do a SimBAL run with every possible cluster. This is practically a brute force method, and six full SimBAL runs to determine one parameter is not a good return. Fortunately, while we do actually have to do the six runs, we don't have to do them for very long. In both of the analyzed objects the likelihoods converge to stable distributions within 1000 steps (Figure 3.8). It is already considered a good practice to do a brief SimBAL run just to make sure nothing is wrong before committing to the tens of thousands of steps of a full run. This additional check could be folded in easily enough. Some clusters could even be skipped, if there was a high confidence they would not be the optimal one.

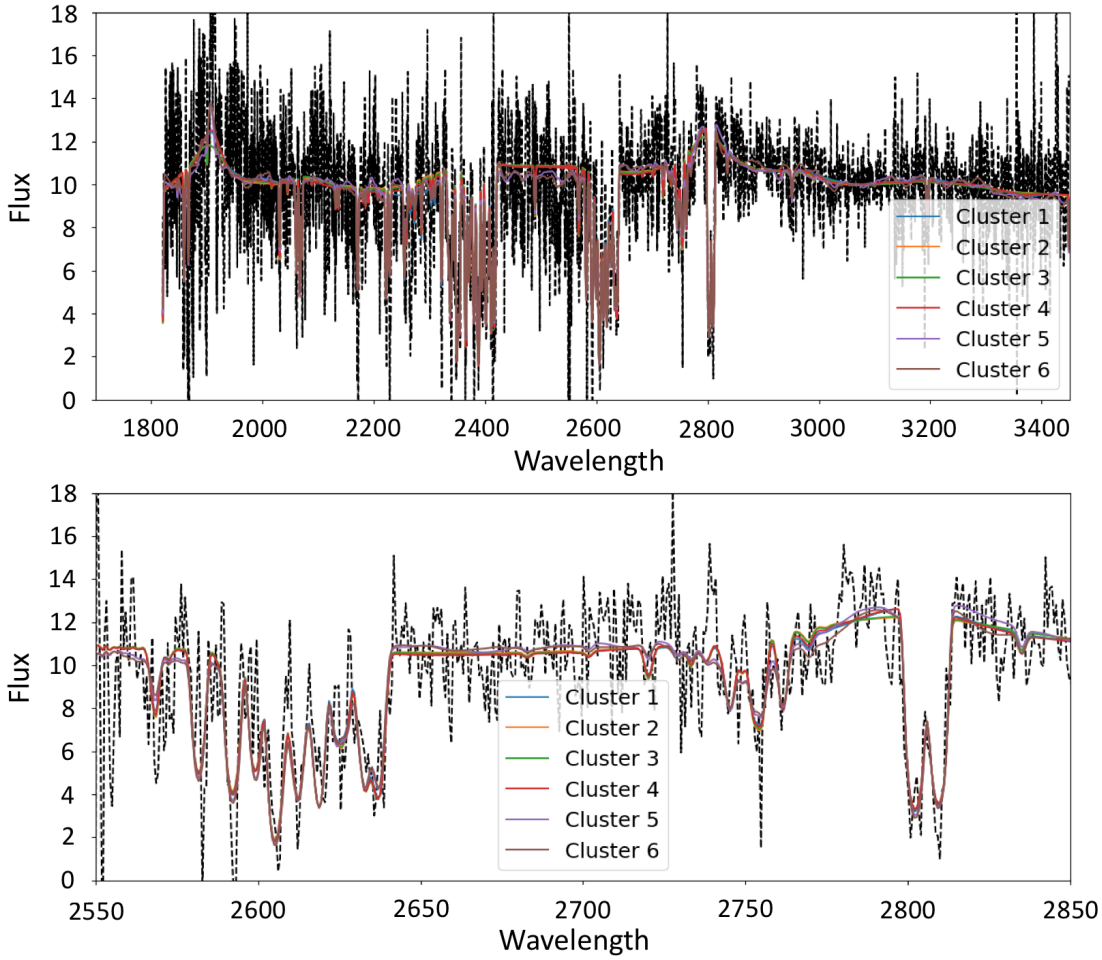

Figure 3.2: The median posterior models for the different cluster runs of J0918. The top plot is the entire spectrum, and the bottom plot is a zoomed in view to show line details. The models are fitting well and the spectra look almost identical for all of the different clusters, which is good.

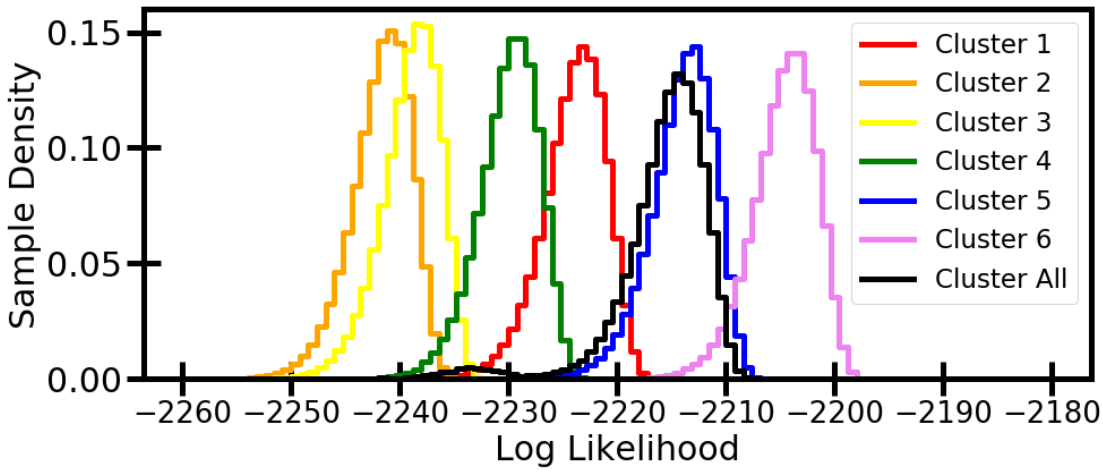

Figure 3.3: A histogram of the likelihood at each sampled point for each cluster's J0918 SimBAL run. The clusters have a clear hierarchy of likelihood ranges, so for this object at least likelihood is a good diagnostic.

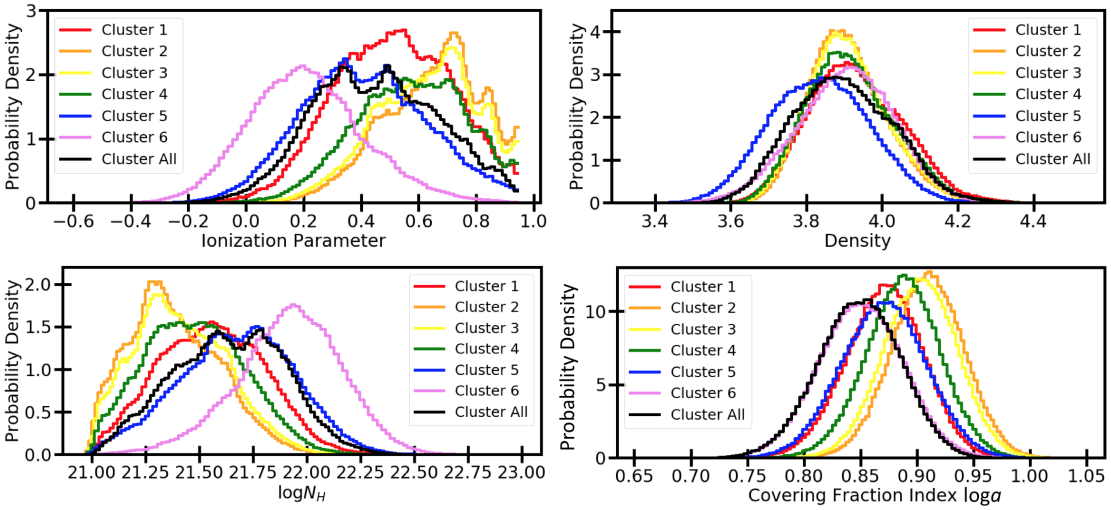

Figure 3.4: A selection of posteriors for absorption parameters of J0918. Three of the parameters (ionization parameter, column density, and covering fraction index) show marked cluster dependence, while density does not. This suggests that there is a systematic error on the absorption parameters stemming from cluster choice.

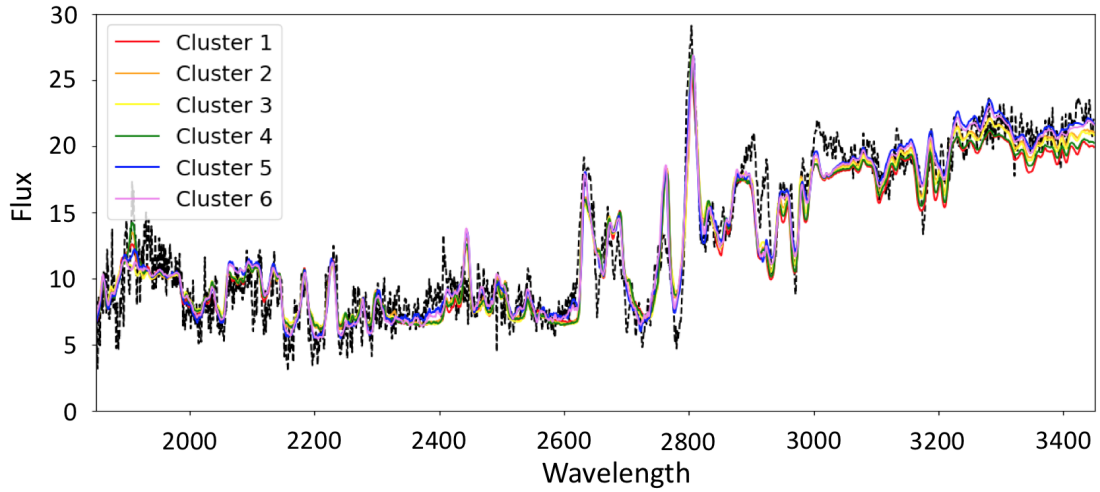

Figure 3.5: The median posterior models for the different cluster runs of J1019. Again, the models are fitting well and the spectra look almost identical for all of the different clusters, although there is noticeable separation at longer wavelengths.

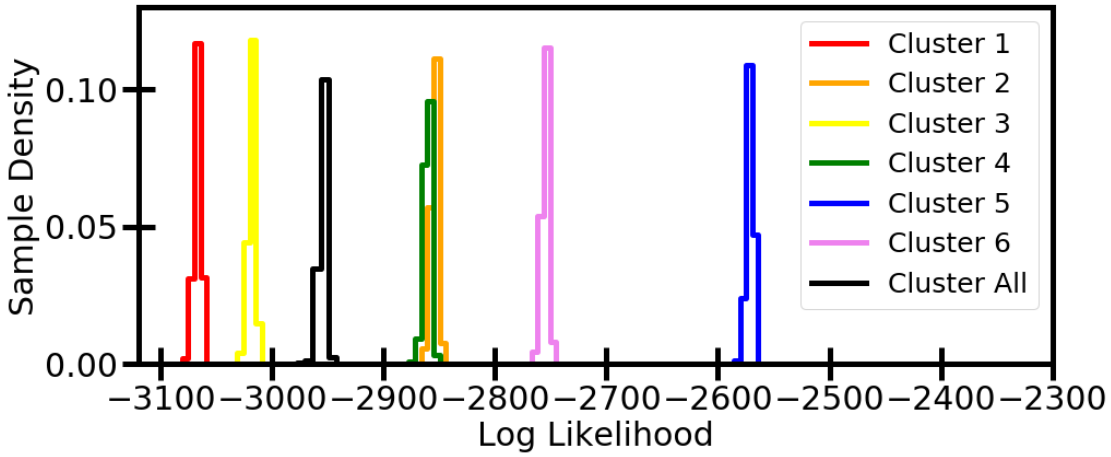

Figure 3.6: A histogram of the likelihood at each sampled point for each cluster's J1019 SimBAL run. The separation between clusters' runs are even more extreme for this object.

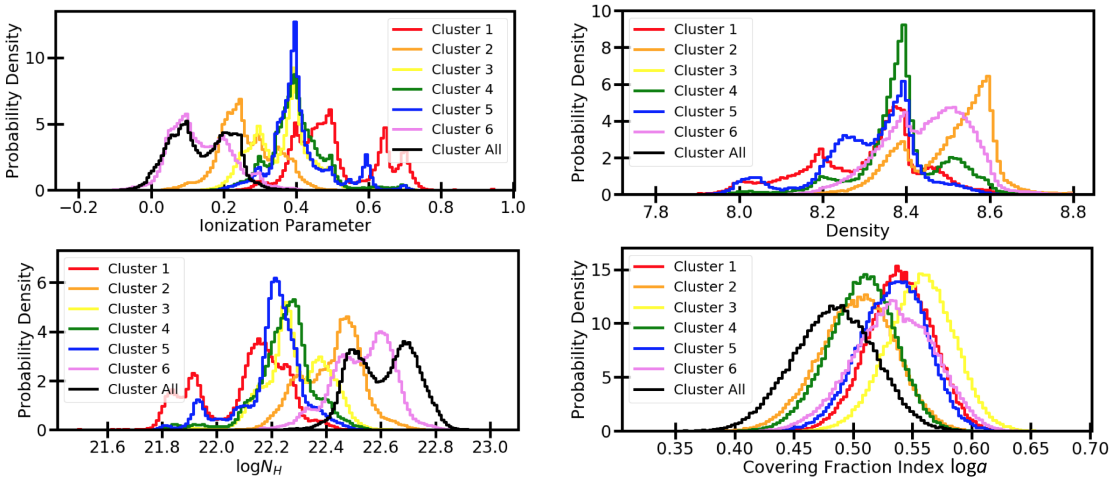

Figure 3.7: A selection of posteriors for the J1019 modeling runs. These posteriors show an even stronger dependence on cluster than the ones for J0918. It seems certain at this point that choosing the wrong cluster will introduce a systematic error into the parameters.

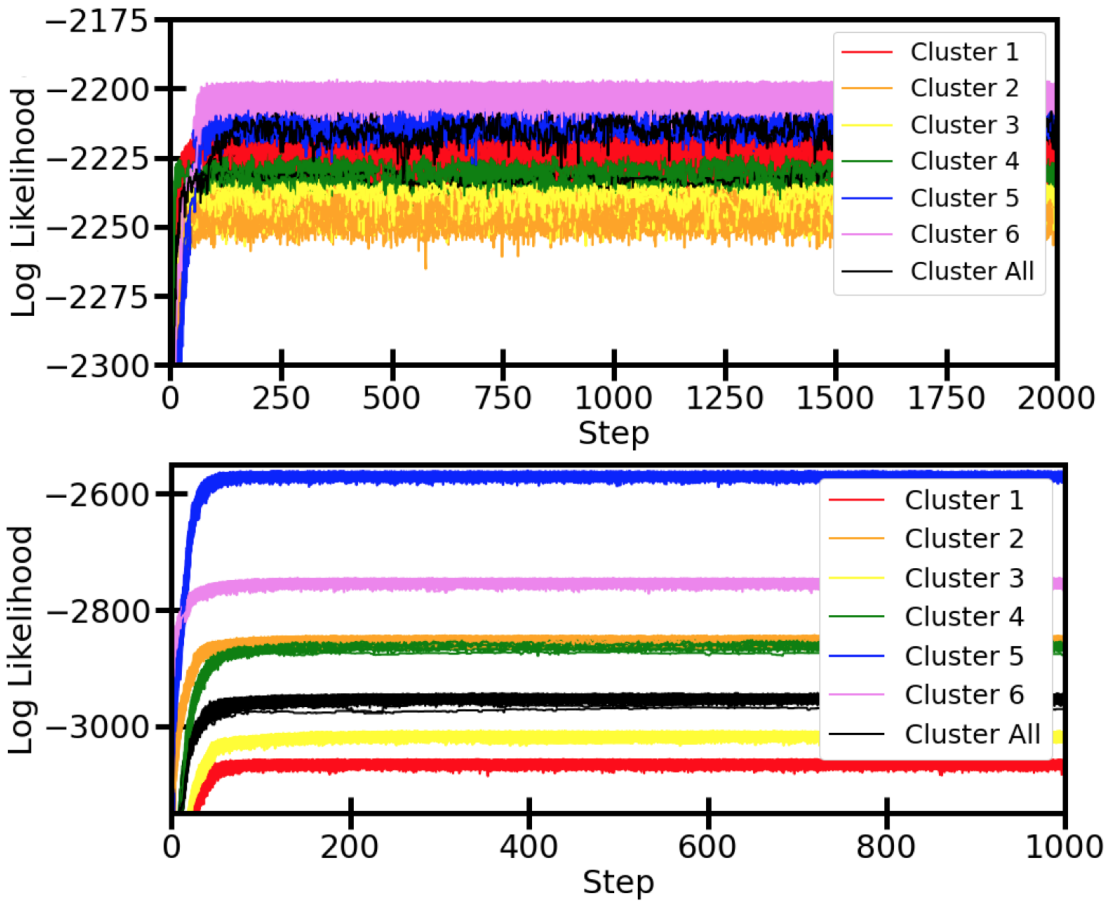

Figure 3.8: A histogram of the likelihood at each sampled point for each cluster's SimBAL run (J0918 for the top plot and J1019 for the bottom). The likelihood values stabilize very rapidly compared to the length of the full SimBAL run, so a short run can be used to determine what the likelihood will be for each different cluster run.

3.3 Conclusion

SimBAL's method for modeling quasar continua is based on a set of eigenvectors generated by performing PCA on a set quasars with narrow emission and no absorption. These eigenvectors were then seperated with k-means clustering, on the assumption that the range of continua could be broken up into five or six broad types, with each type primarily made up of its own subset of the eigenvectors. The issue with this arose from the inability to tell which cluster would be best for modeling an object beyond a reasonable guess, which as shown in the parameter posteriors will introduce a systematic error. A diagnostic was found in a property that SimBAL already has built in, likelihood. This measurement is ideal for comparing the clusters' quality of modeling, as determining the quality of a model is what it is made for. The likelihood was shown to have a consistent cluster dependence for very distinct objects, and so seems like a very promising solution to the problem of cluster selection. For further discussion into possible implementations, see Chapter 6.

Chapter 4

MCMC Comparisons

4.1 Introduction

SimBAL is a very powerful tool, but it has an issue when it comes to building its posterior distributions. Even with only a few velocity components, the parameter space quickly becomes far too large to measure a χ^2 value at each point. There needs to be a way to get a fair sample. A solution presents itself in the form of Markov Chain Monte Carlo (MCMC). MCMC is a class of algorithms that uses Markov chain methods to efficiently sample a parameter space and give a representative distribution. In this chapter I test various MCMC implementations to determine which is best suited for the problem at hand. In Section 4.2 (§4.2) I present the data that the MCMC algorithms will be run on. In §4.3 I give an explanation of the different MCMC methods as well as what properties will be compared between them. In §4.4 I discuss the results of the different processes and compare the performance of each method, especially the runtime of each method. In §4.5 I conclude that while there are advantages to each method, affine-invariant MCMC as implemented in `emcee` is the clear choice for the SimBAL project.

4.2 Data

In order to compare different MCMC processes, we first need to have a data set to run these processes on. I chose two sets of data to run the analysis on. The first is a toy data set made of two Gaussians on a constant background, with a Gaussian scatter added to the data (Figure 4.1). This is a nice test case because there are seven parameters (an amplitude, mean, and standard deviation for the two Gaussians, and an amplitude for the background), which is enough to see the effects of MCMC but not too many to clearly see the influence of each parameter in the final result. For testing SimBAL in the various implementations, the FeLoBAL FBQS J083522.7+424258 will be modeled. This object was chosen because there are already good models for it, so any irregularities will be easy to spot.

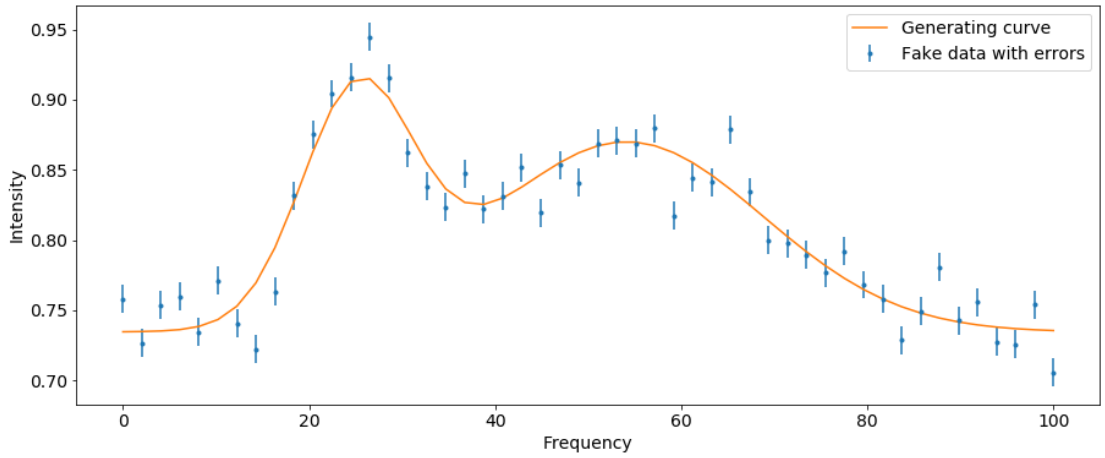

Figure 4.1: The toy data the MCMC methods are tested on. The orange line is the data's generating curve, to which an intrinsic scatter is added to make the data.

4.3 Analysis

In order to run MCMC, two things are needed: a parameter space and a probability function that can be evaluated at every point in that parameter space. The basic process of MCMC is as follows: a walker is placed at some point in parameter space, where it measures the probability. It then chooses a new location to step to, and decides whether or not to take the step or stay where it is. It is biased towards stepping to a point with a higher probability, but it always has some chance to step, no matter how much lower the probability at the new point might be. This is the critical point of MCMC, which differentiates it from methods which always move to a better point until they find some single optimal position. In order to be a fair sampler, it must be able to sample any point in parameter space. Or, to look at it another way, given an arbitrarily long time the walker will step to every possible position. Additionally, the trace of the walker gives the posterior probability distribution. This provides the benefit of avoiding often complex error propagation. It can also be argued that a posterior is a more realistic and meaningful result than a single point estimate, since any measurement has intrinsic uncertainty.

The simplest step method for MCMC is Metropolis-Hastings. For this method, the probability of the walker taking the step is

$$q = min\left(1, \frac{p_2}{p_1}\right)$$

where p_1 is the probability at the walker's current position and p_2 is the probability at the position the walker is considering stepping to. Clearly, the walker will always take the step if it is to a higher probability point. It will also take a step to a lower

probability point some of the time, dependent on how much lower the probability at the new point is.

Metropolis-Hastings is not the only MCMC method. Another way is slicing, which is most easily understood in the univariate case. The algorithm is as follows (Neal 2003):

1. Pick a parameter value to start at.
2. Randomly select a value between zero and the likelihood at that value.
3. Make a horizontal line across the likelihood curve at the height selected in step two.
4. Sample a point on the line segment under the likelihood curve.
5. Use the parameter value of the sampled point to start again.

To generalize to multivariate uses you simply step through each parameter in turn, much like a Gibbs sampler. The advantage of this is that the step size is not fixed, as it is in Metropolis-Hastings. It dynamically adapts to the shape of the probability curve, without any outside input needed. The disadvantage is that in order to do this the entire likelihood curve along that dimension must be known, which already suggests that this method will be too slow for SimBAL (Figure 4.2).

A third method I looked at is a Hamiltonian method, specifically the No U-Turn Sampler (NUTS). Hamiltonian methods are very distinct from most other types of MCMC in that they are not based off of a random walk. Instead, it uses first order gradient information to speed convergence. These types of MCMC are highly dependent on the step size and the number of steps (Hoffman and Gelman 2011), both of which are set by the user. The faster convergence provides no

speed benefit if the walker still has to step through an unnecessarily large number of times, but if there are not enough steps then the random walk behavior this methods is made to avoid returns. To solve this, `PyMC3` (Salvatier et al. 2016) implements the No U-Turn Sampler (NUTS). It takes the tuning out of the user's hands by determining how many steps it should take based on when the walker begins to double back on itself.

All three of the previous methods are implemented in the Python package `PyMC3`. The last method, affine-invariant MCMC, is from an entirely different package called `emcee`. As the name of the method suggests, its steps are invariant under an affine transformation. Affine invariance means that the performance of this method is independent of the aspect ratio of anisotropic likelihood distributions (Goodman and Weare 2010). Since the likelihood distribution in our parameter space may or may not be very anisotropic, this is a nice property to have. The formula for a step using this method is:

$$X_k \rightarrow Y_k = X_j + Z[X_k - X_j]$$

where X_k is the walker's current position, X_j is a randomly selected other walker in the ensemble, Z is a random variable related to the probability of acceptance, and Y_k is the walker's potential new position. In order to preserve detailed balance, the acceptance probability must be of the form

$$q = min\left(1, Z^{N-1}\frac{p(Y_k)}{p(X_k)}\right)$$

where N is the dimension of the parameter space. This has been shown to to have a much shorter autocorrelation time than Metropolis-Hastings (Foreman-

Mackey et al. 2013). This can be done by stepping through each parameter in series. However, running the walkers in parallel would be desirable to shorten the computational time. The naive implementation of this would have every walker running in parallel based on the state of the ensemble at that time. However, this would violate detailed balance. Instead, the ensemble must be split into two equal subsets. All of the walkers in one group will update in parallel based on the state of the other group, and then the "active" group will switch so at any time half of the walkers are updating.

Each method was run with thirty walkers and 10,000 steps, and no parallelization was used on the toy data. An identical likelihood[1] was used for all methods. The real data was analyzed with 50 thread parallelization. I recorded the amount of time each run took, as the SimBAL project aims to analyze a very large sample and any reduction in analysis time can yield large savings in the long run. I also generated traceplots for each run. A traceplot is a plot of each walker's position at each step in the MCMC. Since it is a two dimensional plot, it can only show one dimension of parameter space at a time.`PyMC3` has a function for generating these built in, but for the `emcee` run I had to make it manually. Perhaps most importantly from a results point of view, I also generated histograms of all of the posteriors and compared them. I then plotted a point estimate (Figure 4.3) and confidence regions because it is always a good check to make sure the result looks reasonable.

[1]The same χ^2 likelihood that SimBAL uses.

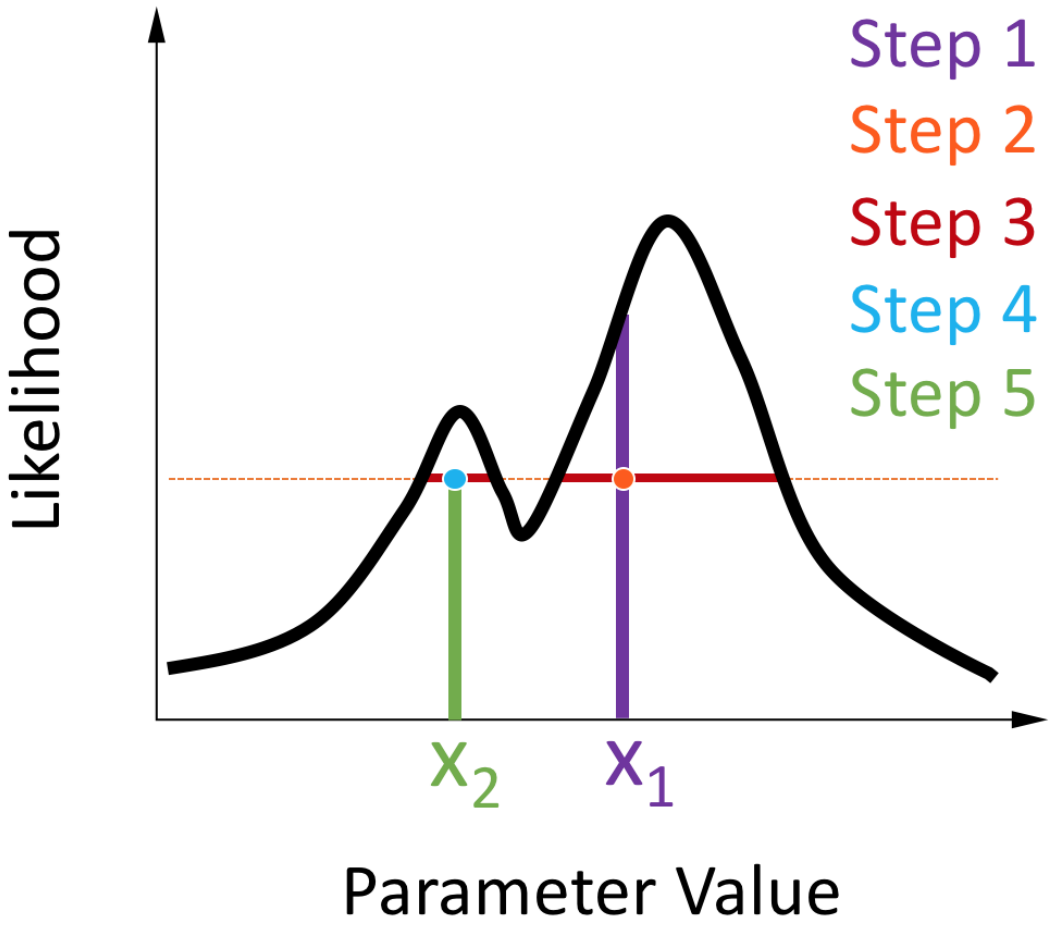

Figure 4.2: A color coded diagram of the steps for the slice sampler. Note that the value of the likelihood must be known everywhere along the line, which already suggests that this will be an impractical method for SimBAL.

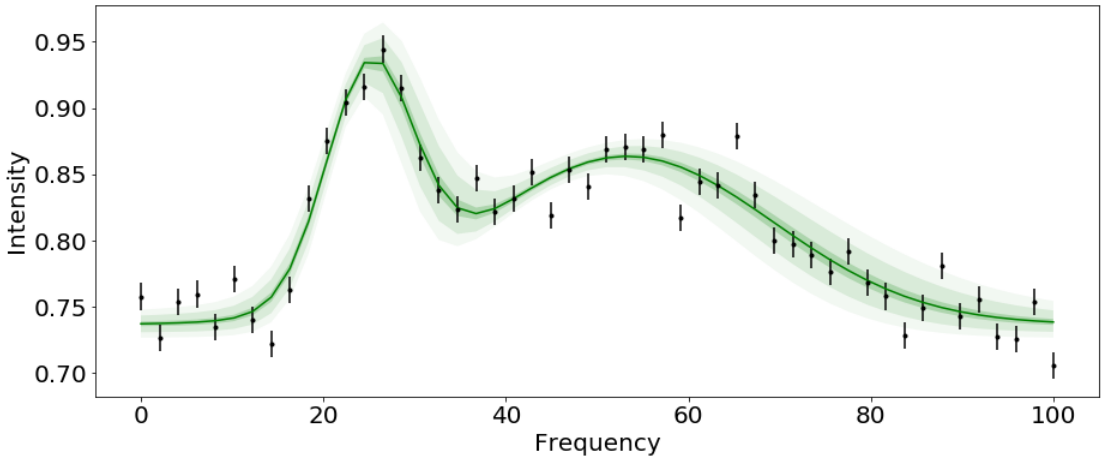

Figure 4.3: The results of using Metropolis-Hastings in `PyMC3` to run MCMC on the toy data set. The green line is the point estimate, while the three increasingly transparent regions are the one, two, and three sigma confidence intervals for the final model.

4.4 Discussion

The most important comparison between the different methods is making sure the posteriors match. If there are any major discrepancies between the distributions, it implies a serious issue with one or more of the algorithms. Looking at the posteriors (Figure 4.4), there is a clear visual difference between the posteriors for `emcee` and Metropolis-Hastings versus NUTS and Slice sampling. This pattern is consistent throughout the parameters. The reason behind this separation is unclear, but the point estimate from each of these posteriors is within the two sigma confidence region of every other posterior, so the difference is of questionable significance.

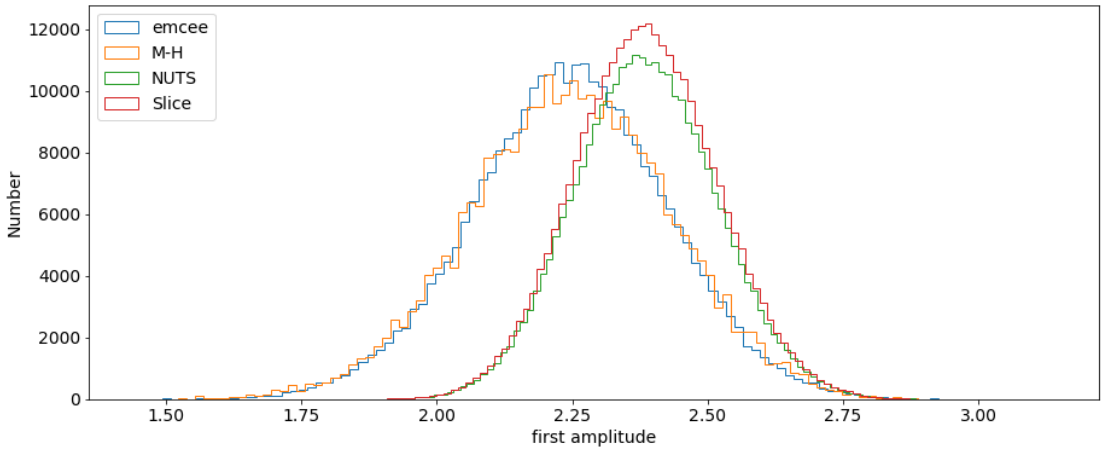

Figure 4.4: The posteriors for the amplitude of the first Gaussian for each MCMC method. The `emcee` and Metropolis-Hastings methods generate identical posteriors, and the Nuts and Slice samplers generate identical posteriors, but these two sets of posteriors are not identical to each other. This is a worrying result, as the results should not depend on the step method of the sampler.

The next most important factor to compare is arguably runtime. It might at first seem like the speed of convergence would be more important, since if the walkers converge faster then less steps are needed. However, there always need to be enough steps after the chain has converged to have a well sampled poste-

rior. The chains often converge very rapidly, making sampling the limiting factor. Additionally, the difference in the runtime between the methods was much more dramatic than expected (Figure 4.5), with every `PyMC3` method being consistently slower than `emcee` by a factor of at least 1.4 dex. And this is without utilizing parallelization, which both packages implement but only `emcee` has as a core principle of its algorithm. Slice especially is clearly hindered by its need to sample the likelihood over a single parameter's entire range each step.

Because `PyMC3` handles multiple walkers by simply running them in series, all of its walkers are independent, while by definition `emcee's` walkers are not. This might seems to imply that the results of thirty `PyMC3` walkers are more significant that those of thirty emcee walkers, but I would argue any such claim is spurious. With an equal number of walkers and steps, each method samples the same number of points. Furthermore, each of these methods maintains detailed balance, ergodicity, and every other prerequisite for an MCMC method to be a fair sampler, so I see no reason to consider one more meaningful than the other.

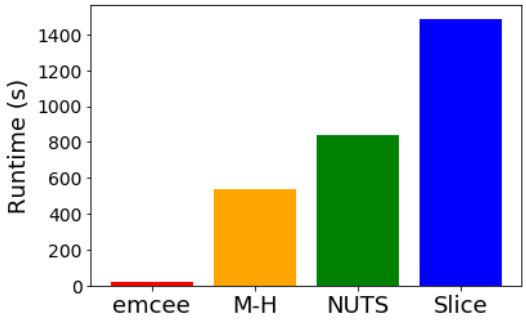

Figure 4.5: A comparison of the time elapsed while running the various MCMC methods. All of the `PyMC3` methods are much slower that of `emcee`.

The analysis of the real data leads to the same conclusions. There are slight differences in the posteriors. All of `PyMC3`'s methods are significantly slower than emcee. The paralellization does not seem to be doing anything to close the time gap between the two packages. In fact, `PyMC3`'s methods could not be run with as many walkers as `emcee` without making the runtime unacceptably long. There

is an argument to be made that the runtimes are already unacceptably long for these methods. All of this points to emcee as being the clear winner for running in SimBAL.

While not entirely relevant to the question at hand, I think it is worth pointing out that there are advantages to `PyMC3` which are simply not important for SimBAL in its current state. For instance, `PyMC3` has a variety of built in priors, while emcee does not. This does not count in its favor for the SimBAL project because a custom prior class has already been coded, but if I was considering this question at the beginning of a coding project it would certainly factor in.

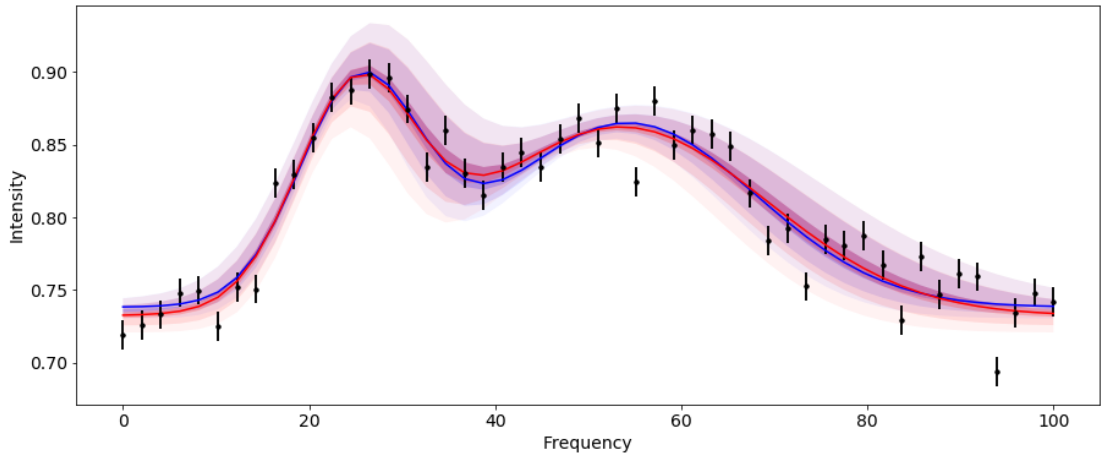

Figure 4.6: The results of using Metropolis-Hastings (in red) and NUTS (in blue)to run MCMC on the toy data set. Note that while the models do visibly differ, they are well inside each others confidence intervals.

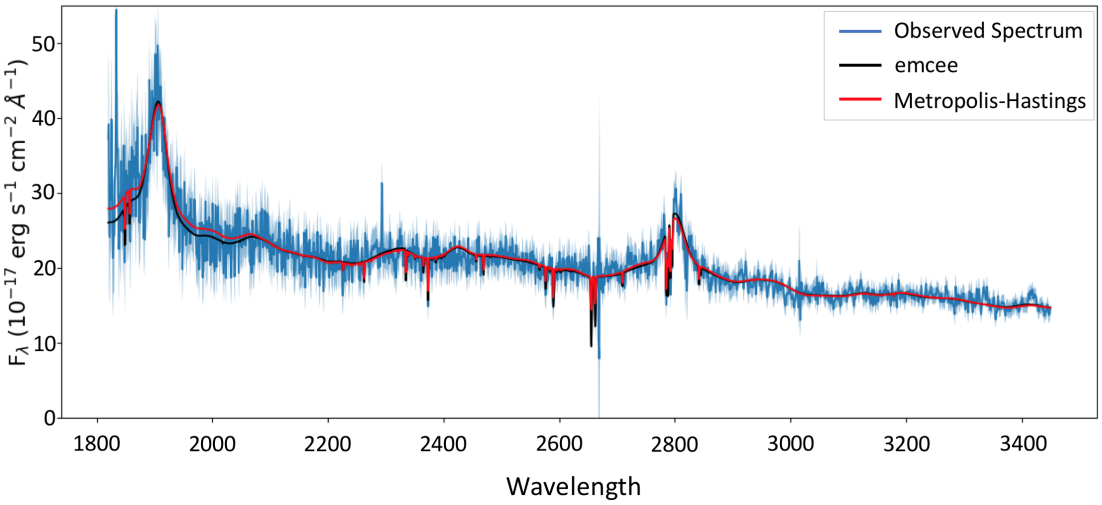

Figure 4.7: The results of modeling j081312 with `emcee` and `PyMC3`'s Metropolis-Hastings. The `PyMC3` run only used one walker, as any more than that causes the runtime to become impractically long. The models visibly differ, which further raises the issue of different step methods giving different results. This should be further investigated by the SimBAL team moving forward.

4.5 Conclusion

In general, there are advantages and disadvantages to each method and both packages. `PyMC3` has more built in functions, and can easily change the type of MCMC method being run. On the other hand, `emcee` naturally runs an entire ensemble of walkers. Most of the advantages of `PyMC3` do not apply to the SimBAL project, however. In the end, nothing in `PyMC3`'s favor outweighs the dramatic increase in time needed to run an equivalent amount of walkers and steps, a failing especially dire for a project that aims to analyze a large number of objects. The `emcee` package is definitely the correct choice for implementing SimBAL.

Chapter 5

Column Density Processing

5.1 Introduction

As mentioned before, the SimBAL program uses a grid of ionic column densities calculated from Cloudy models. These are actually stored in what we call product files. At each point in ionization parameter/density/$\log N_H - \log U$ space, there is an ionic column density for each level of each ion. In order to use these column densities, they must first be converted to line opacities. To save on calculations while SimBAL is running, this is partially done ahead of time according to the method described in Savage and Sembach (1991). The relation they give is

$$\tau(v) = \frac{\pi e^2}{m_e c} f\lambda N(v) = 2.654 \times 10^{-15} f\lambda N(v),$$

where λ is the line center wavelength, f is the oscillator strength for the transition that generates the line, $N(v)$ is the ionic column density per unit velocity, and $\tau(v)$ is the optical depth of the line per unit velocity. The product file is so called because it has the product $f\lambda N(v)$ already calculated as each of its entries. Since

each ion has many transitions, this increases the product file size dramatically. With 81 ionization parameters, 27 densities, 101 $\log N_H - \log U$ values, and 179 ions with a total of 6267 lines, this is 1,384,298,829 entries[1]. Further exacerbating the issue is that SimBAL is being shifted to use Cloudy17 (Ferland et al. 2017), where we will be considering 1455 ions and excited states. Assuming a similar ratio of ions to lines gives an estimated size of a Cloudy17 product file at eleven billion entries.

Reducing the memory taken up by these files is critical to making SimBAL practical to use, especially once it is released to the quasar community at large. There are many potential ways to reduce the size of these necessary external files. In this chapter I discuss my implementation of several alternative column density storage methods, as well as compare the various time and memory costs associated with each, and make a recommendation for which should be adopted as the standard for SimBAL.

5.2 Implementation

The current version of SimBAL uses product files, as described in the introduction. While one product file is big enough that loading it into memory can cause some machines to crash, there is actually an additional problem in the fact that there are several different product files for various parameter ranges. The reason for having several product files is that the appropriate density and column density ranges for a given object can be identified by eye, so there doesn't need to be a single massive file for that can be used on every object. Each one is around 11Gb so storing them is not a issue currently, but it will become extremely inconvenient when SimBAL

[1] $81 \times 27 \times 101 \times 6267$

moves to Cloudy17 and everything scales up by a factor of ten.

The first alternate implementation involves stepping back from the products to the ionic column densities. The product files were originally made by multiplying the ionic column densities by the wavelength and oscillator strength of each transition. This saved on operations that needed to be done each step of a SimBAL run at the cost of memory by doing all of the multiplication ahead of time, but needing to store a product for every transition of every ion at every combination of ionization parameter, density, and column density. Using the original column files is significantly less memory intensive, as you only need a single value for each ion at each position in parameter space, along with a single list of atomic data (wavelengths and oscillator strengths) for each ion. This reduces the number of entries at each point in parameter space from 6267 to 179, and the atomic data list is negligibly small compared to either the product or column file.

SimBAL's spectral synthesis code had to be changed, obviously. The multiplication had to be brought in, but this also provided an opportunity to restructure the line making code. Previously, SimBAL looped through an unordered list of line centers, adding the ones within the spectrum's bandpass. A slightly more intuitive way was to instead organize column densities by ion (Figure 5.3). This in turn makes it easy to implement another time and memory saving change, trimming the files. This is the checking of the atomic data list for transitions outside of the spectral bandpass before beginning to synthesize spectra. Any transitions outside of the wavelength range are removed from the list at the very beginning, so no check needs to be performed as the spectrum is being made. Furthermore, if all of the transitions for a particular ion are outside of the range, the column density for that ion in the column file can be removed as well.

The column file implementation can be expected to provide significant memory

savings, but there are further ways to try and compress the information. At each ionization parameter and density, every ion can be thought of as having a curve of ionic column density versus $\log N_H - \log U$. Figure 5.1 is a visualization of this. Several different ions have their column density curves plotted. Since they are curves, they should be able to be fitted by some method. Then only the fit parameters would need to be stored, reducing memory usage even further. Of course, reconstructing the curve adds additional operations to perform in the spectral synthesis code, likely increasing the runetime even further.

Many of the curves are sigmoidal. This is due to the hydrogen ionization front, which is the transition between the region of ionized hydrogen and neutral hydrogen. At low column densities the gas is entirely in the ionized hydrogen region, and so there are very energetic photons throughout. This means that ions like single ionized iron (Fe+) cannot exist without being immediately further ionized, so the column density for that particular ion will be very low. As the gas gets thicker, it eventually becomes thick enough to absorb all of the photons energetic enough to ionize hydrogen. That means that ions like Fe+ have a chance to exist, so their ionic column densities jump up dramatically.

A tolerance of 0.1 dex was decided upon for a good fit. Polynomial, spline, and regression (linear, lasso, and ridge) fitting were all tried and abandoned for various reasons, usually having to do with an inability to fit well with a low enough number of fit parameters. The method that worked turned out to be one very familiar to the SimBAL project, principal component analysis[2]. In retrospect it is a very sensible method to try, since the problem is fundamentally one of dimensionality reduction.

In order to be advantageous relative to the regular column file, the number of

[2]See Chapter 3 for a detailed description of PCA

eigenvectors kept needs to be low enough to offset keeping both eigenvectors and coefficients. Putting it concretely, the relation

$$N_{ionpar} * N_{den} * N_{\log N_H - \log U} * N_{ion} < (N_{ionpar} * N_{den} + N_{\log N_H - \log U}) * N_{ion} * N_{eigenvec}$$

must be true. For the parameter ranges being used in this comparison, $N_{ionpar} = 81$, $N_{den} = 27$, $N_{\log N_H - \log U} = 101$, $N_{ion} = 179$, and $N_{eigenvec}$ is our variable. This works out to be that there must be fewer than 96 eigenvectors to see any memory savings, although ideally there will be far fewer than that to make the inevitable increase in runtime worth it.

PCA was performed at each ionization parameter and density, across all ions. It worked extraordinarily well, being able to reconstruct all points within tolerance when only keeping twenty out of 179 eigenvectors (Figure 5.2). This seems sensical, as looking back at Figure 5.1 there are several clear categories that each curve falls into. This suggests that the space could be well represented by a small number of archetypal eigenvectors. From the relation above, it is expected that the PCA column file will be about a fifth the size of the regular column file.

Figure 5.3 gives a schematic view of the three different implementations. In the original product file implementation, each point of the three dimensional gas parameter space has an unordered list of every line under consideration, and each line's entry is the $f\lambda N$ product. In the separate column file and atomic data file implementation, both files are organized as a list of ions. In the column file, each ion's entry is the three dimensional gas parameter space, and each point of that space is an ionic column density. In the atomic data file, each ion's entry is a list of every transition modeled for that ion, and every transition has the wavelength and oscillator strength. The ionic column density, wavelength, and oscillator strength are

only combined once SimBAL is running. In the PCA column file implementation, the atomic data file is identical to the previous implementation. The difference is the way the ionic column densities are stored. For every point in $\log U$, $\log n$ space there is a set of eigenvectors. These eigenvectors are the same no matter the ion. Each ion has a set of coefficients as well. The coefficients and eigenvectors are then combined in SimBAL to get the ionic column densities, which are then combined with the atomic data. This last implementation has by far the most steps.

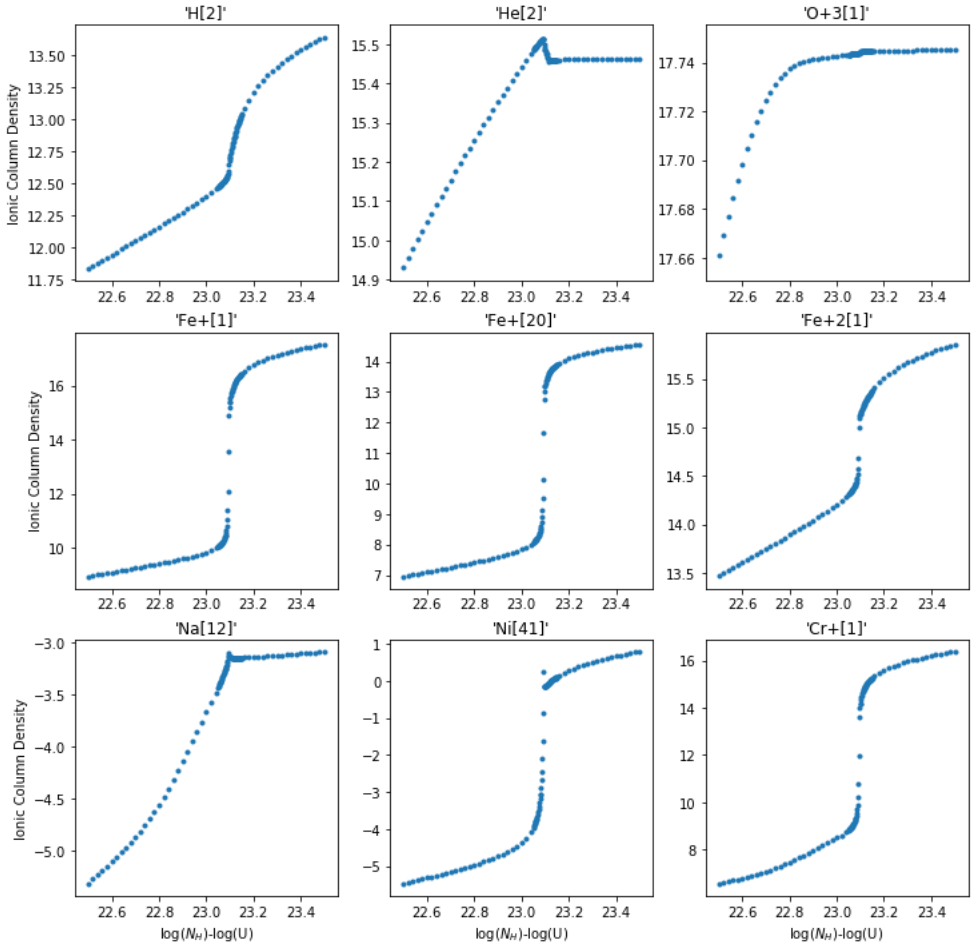

Figure 5.1: A selection of ionic column density curves. The title shows the ion with the excitation level in brackets (the ground state is [1]). The sigmoid-like behavior seen in several of them is due to the hydrogen ionization front, where they are oversampled to attempt to help map the rapid changes. The variety of shapes means fitting them is not trivial.

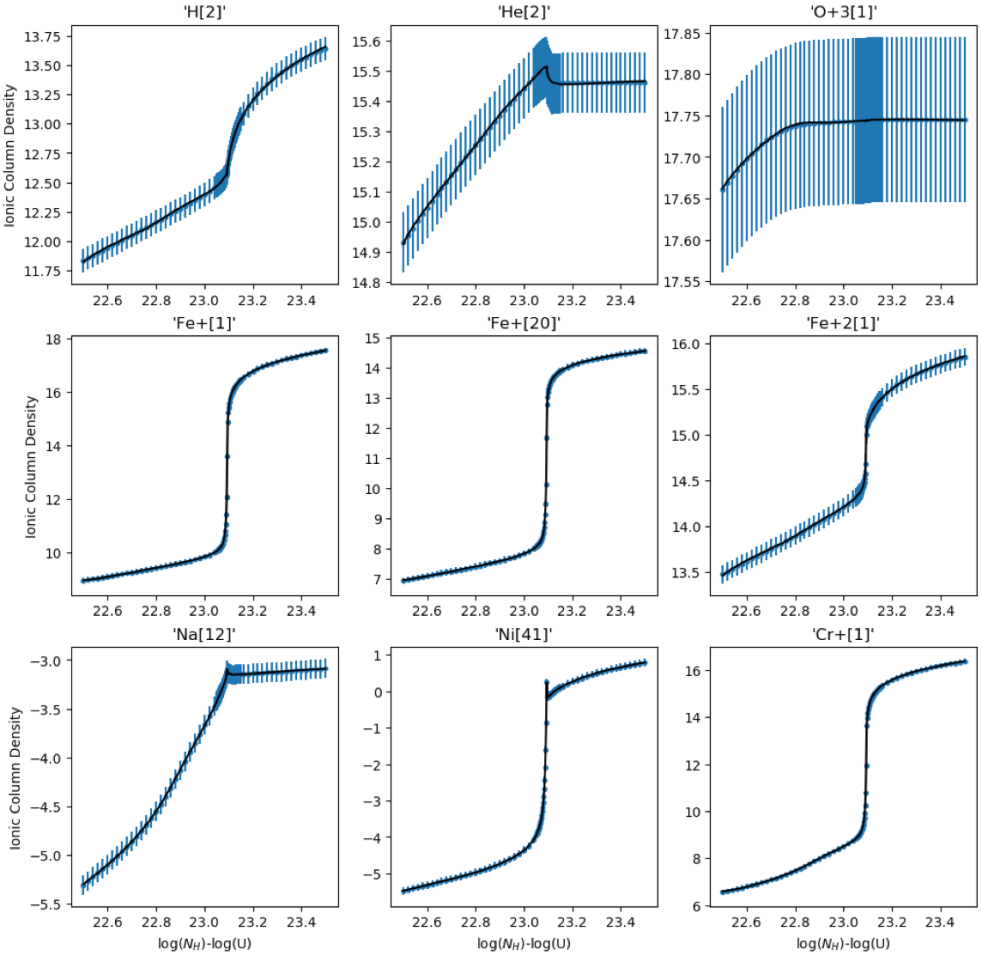

Figure 5.2: The same ionic column density curves as in Figure 5.1, but with the tolerance shown as blue bars and the curve reconstructed from the PCA in black. The PCA reconstruction is within tolerance for every point of every curve (even those not shown) with twenty eigenvectors.

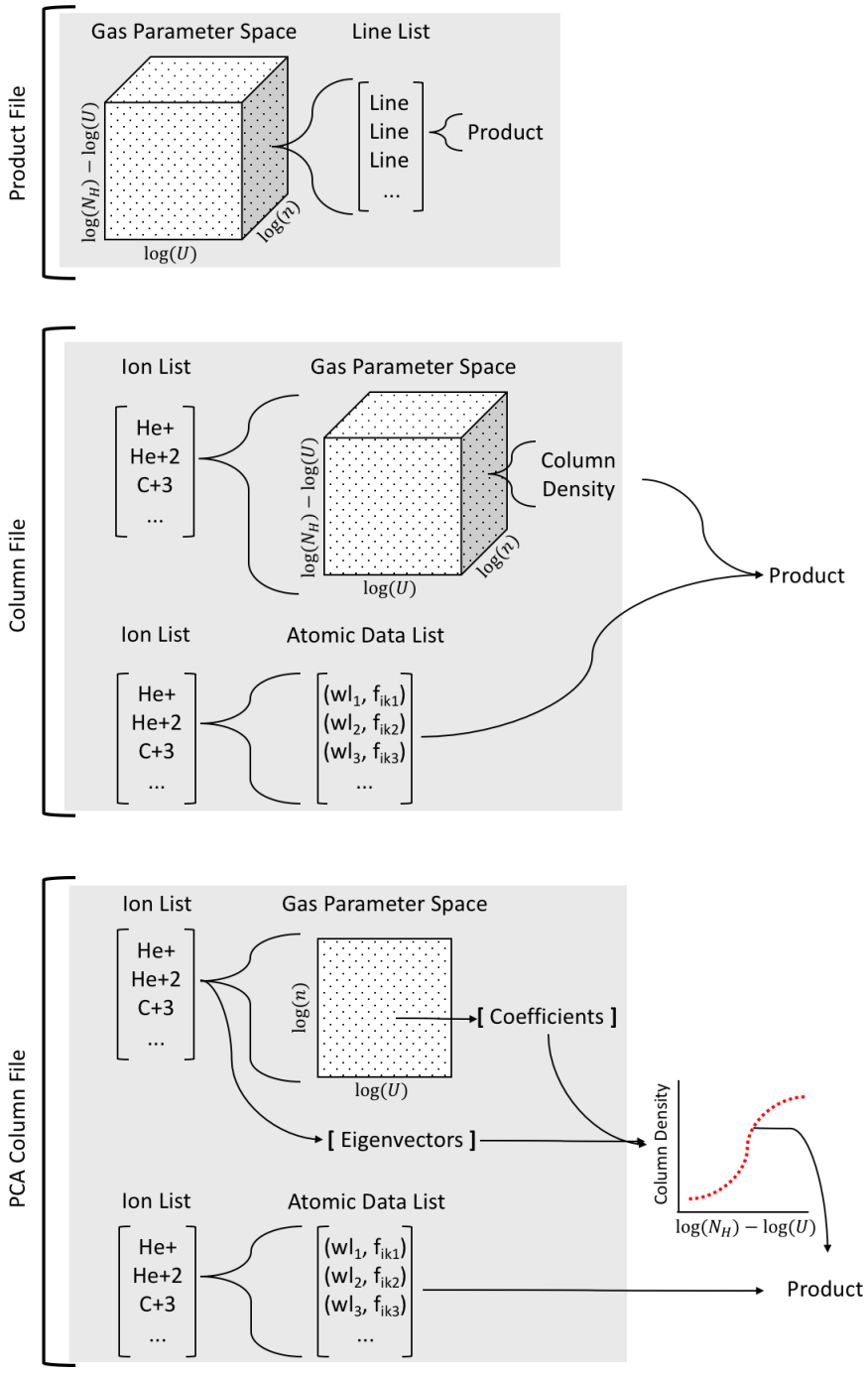

Figure 5.3: A schematic view of the three different ionic column density storage methods. Any of the sections in grey are done before the MCMC in the file. Everything else has to be done by SimBAL at each step. The complexity increases moving downward, indicating a likely increase in runtime to match.

5.3 Discussion

The three SimBAL implementations were used to make varying numbers of spectra with random input parameters. The runtimes for all three can be seen in Figure 5.4. The curves all start off fairly flat. This is due to the constant amount of time taken loading in the product/column file at the beginning of every run. The product file takes the longest to load by far due to its size. There is a much smaller difference between the column and PCA files, as they are much closer in size. As the number of spectra made increases, the initial loading time becomes a smaller percentage, and the trend becomes linear. In this regime the product file implementation is faster, as expected. The PCA version is the slowest, most likely due to the large number of operations needed to first reconstruct the ionic column density versus $\log N_H - \log U$ curve. The blue region is the range that SimBAL is usually working in, and so is the most important for comparison purposes.

The other property to consider is memory usage. The product file is about 11 Gb, the column file is 767 Mb, and the PCA column file is 273 Mb. The atomic data list needed to use either product file is <1 Mb, and so can be safely ignored for this comparison. The memory decrease when moving from the column file to the PCA column file is less than predicted, but this is likely due to file size not being completely linear with number of values stored.

The question when considering this information about the different implementations is, are the memory savings worth the time loss? In the region of interest, the time increase in changing from a product file to a column file is about 235%. This seems like quite a large slowdown, and it is, but the memory usage is reduced by a factor of 14.3. That is a dramatic reduction in size, and would most likely make the eventual Cloudy17 column files smaller than the current Cloudy13 prod-

uct files despite the huge increase in number of lines being considered. Comparing the PCA column file to the product file gives a runtime increase of 978% and a memory decrease of 40.3 times. This makes it seem like the PCA implementation continues the trend of significant time losses but even more significant memory savings. However, comparing the PCA column file to the regular column file tells a different story. The runtime increases by 416%, but the memory only decreases by a factor of 2.8. This is a much higher cost and lower return than the first change was.

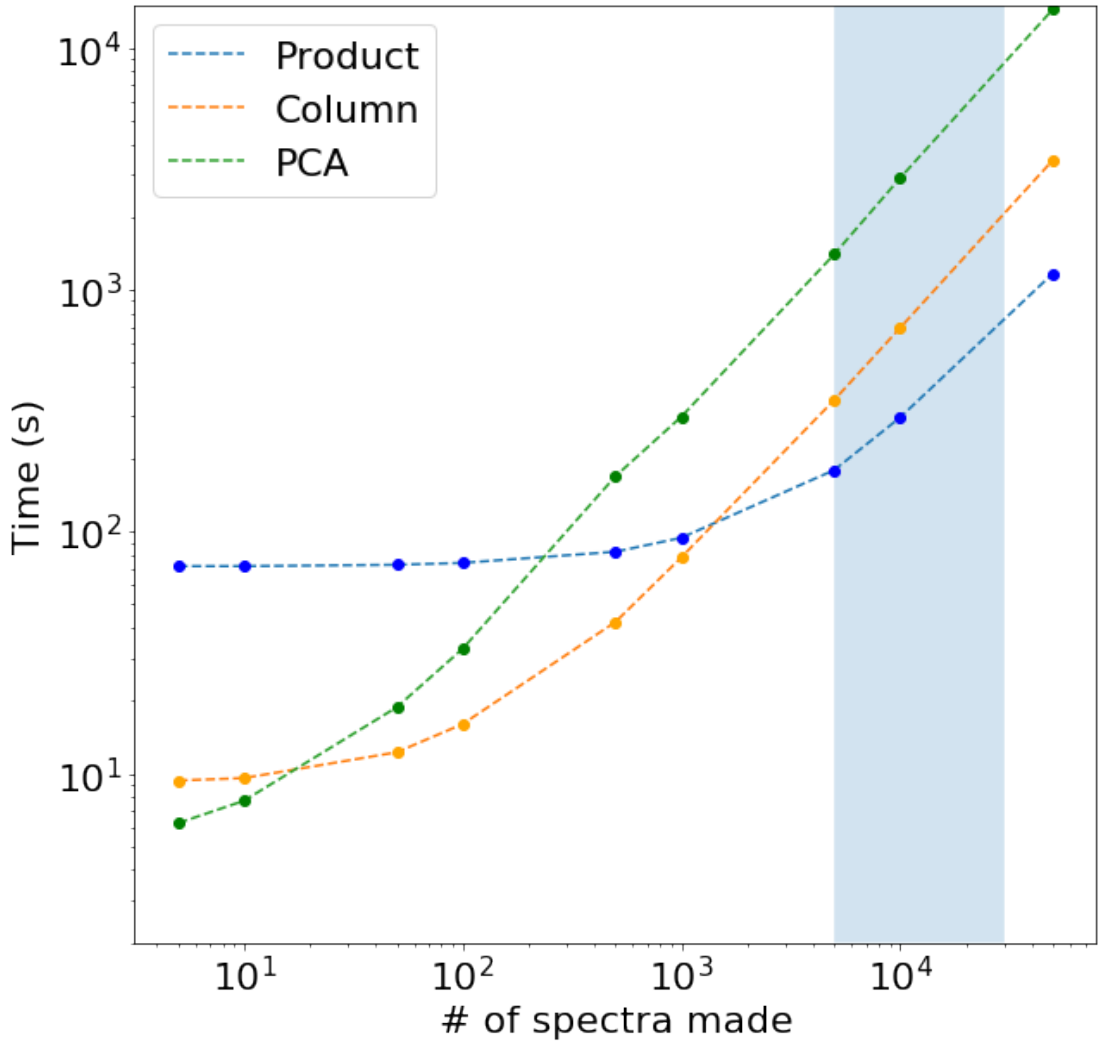

Figure 5.4: This plot shows the times it takes to make various numbers of spectra with the different ionic column density storage formats. The region colored blue is the range that a typical SimBAL run will be in. At low numbers of spectra, the time to load the product or column file into memory is a significant factor in the total runtime, which is why all three curves show a flattening at the lower end. For large numbers of spectra made, the behavior becomes linear as the constant time contribution from the single initial loading becomes a negligible percentage of runtime.

5.4 Conclusion

In this chapter, three implementations of SimBAL were compared. The product implementation is the current version of SimBAL, and uses a large product file with many operations already done to save time. The column implementation moves the product creation inside the spectral synthesis code to allow a smaller external file. Lastly, the PCA implementation builds off of the column implementation by column information as a series of eigenvectors and coefficients, further reducing the external file size and adding additional operations to the synthesis code. All were timed making various numbers of spectra, and the timing information was weighed against the file sizes.

The significant memory savings of switching to a column file implementation, combined with the diminishing returns of further shrinking the file with PCA, lead me to recommend the column file implementation for SimBAL going forward. It allows the move to Cloudy17 to proceed without concern over memory usage, and requires a relatively modest increase in runtime in exchange. I find it hard to justify the additional reduction gained by the PCA column file with the time loss it incurs. The column implementation keeps SimBAL usable, both in the amount of memory and time needed to run it.

Chapter 6

Future Work

SimBAL is going through a transitional period as it changes over to Cloudy17. This presents an opportunity to make improvements to SimBAL that will pay dividends for years to come. In this brief chapter I will cover what has been done and what still needs to be done to take advantage of the analysis in the paper.

The implementation of resolution smearing is complete, and is in use by most members of the SimBAL team already. It was a fairly self contained, resolution being separated as it is from the source of most of the rest of the spectrum's features out at the quasar. The only major change it required was an alteration of how the spectra data files were treated. Bad points now have to be masked out rather than simply deleted, as missing pixels would cause the flux to smear incorrectly. All of this has been adopted, and this change should be considered finished.

Cluster dependence was somewhat unexplored before the analysis presented here, and so there is still room to investigate further. In particular, the number of clusters generated by the k-means process was chosen somewhat arbitrarily. Could a further increase in the number of clusters better fit the real types of

quasar continua and lead to higher likelihoods? Of course, increasing the number of clusters would make choosing the correct one for modeling more difficult, but using the likelihood as a check could counter that issue. The process of doing several brief SimBAL runs to determine which cluster is making the highest likelihood models could be automated fairly easily. The cluster selection could even be automated, although that might be taking too much of the human checking out of it. SimBAL is not yet at a stable version, and so over-automation should probably be avoided at this stage.

The MCMC comparison was interesting, and shed a lot of light on how the various types of MCMC operate, but the end result is a recommendation to not change anything. `emcee` is such an efficient way to sample the parameter space that no other process could reasonably compete, at least not without unacceptable increases to runtime. The results all confirm that abandoning the `PyMC3` version of SimBAL early in development was a wise decision. It is possible that a custom MCMC program optimized for the needs of SimBAL could outperform `emcee`, but that is probably not the best use of resources for an already ambitious development project.

The ionic column density storage analysis was probably the most urgent of the topics covered here, as the Cloudy17 implementation is rapidly approaching, bringing with it a large increase in memory usage that needs to be counterbalanced. The analysis posed an interesting question about where the balance between memory and runtime should be struck, and there is no single right answer. All three methods were full implemented in SimBAL, and so any could be used going forward. I personally recommend the column file implementation, as it provides the necessary reduction in file size compared to the produce file with a relatively modest runtime increase compared to the PCA column file method. There are certainly more ways

to try to reduce memory usage and increase the speed of the code, so many so that there is not much point in suggesting a single one here. It may be that the PCA reconstruction could be streamlined in some way I did not think of, making it a more attractive option. Or maybe computer memory will continue to grow at such as rate that this whole issue is moot in a few years. However, as it stands I believe the standard column file is the smartest choice.

References

Bailey, S., 2012, PASP, 124, 1015

Fabian, A. C., 2012, Annu. Rev. Astron. Astrophys., 50, 455

Farrah, D., Urrutia, T., Lacy, M., et al., 2012, ApJ, 745, 178

Ferland, G. J., et al., 2013, RMxAA, 49, 137

Ferland, G. J., et al., 2017, RMxAA, 53, 385

Ferrarese, L., Merritt, D., 2000, ApJ, 539, L9

Foreman-Mackey, D., Hogg, D. W., Lang, D., Goodman, J., 2013, PASP, 125, 925

Green, R., 2013, The Quasar Luminosity Function. In: Fifty years of quasars. [online] Available at: http://www.astro.caltech.edu/q50/pdfs/Green.pdf

Hoffman, M. D., Gelman A., 2011, arXiv, 1111, 4246

King, A., Pounds, K., 2015, Annu. Rev. Astron. Astrophys., 53, 115

Leighly, K. M., et al., 2018, submitted to ApJ

Leighly, K. M., Moore, J. R., 2006, ApJ, 644, 748

Neal, R. M., 2003, The Annals of Statistics, 31, 705

Savage, B. D., Sembach, K. R., 1991, ApJ, 379, 245

Salvatier J., Wiecki T.V., Fonnesbeck C., 2016, PeerJ Computer Science 2:e55m,

DOI: 10.7717/peerj-cs.55.

Appendix

MCMC Comparison Graphs

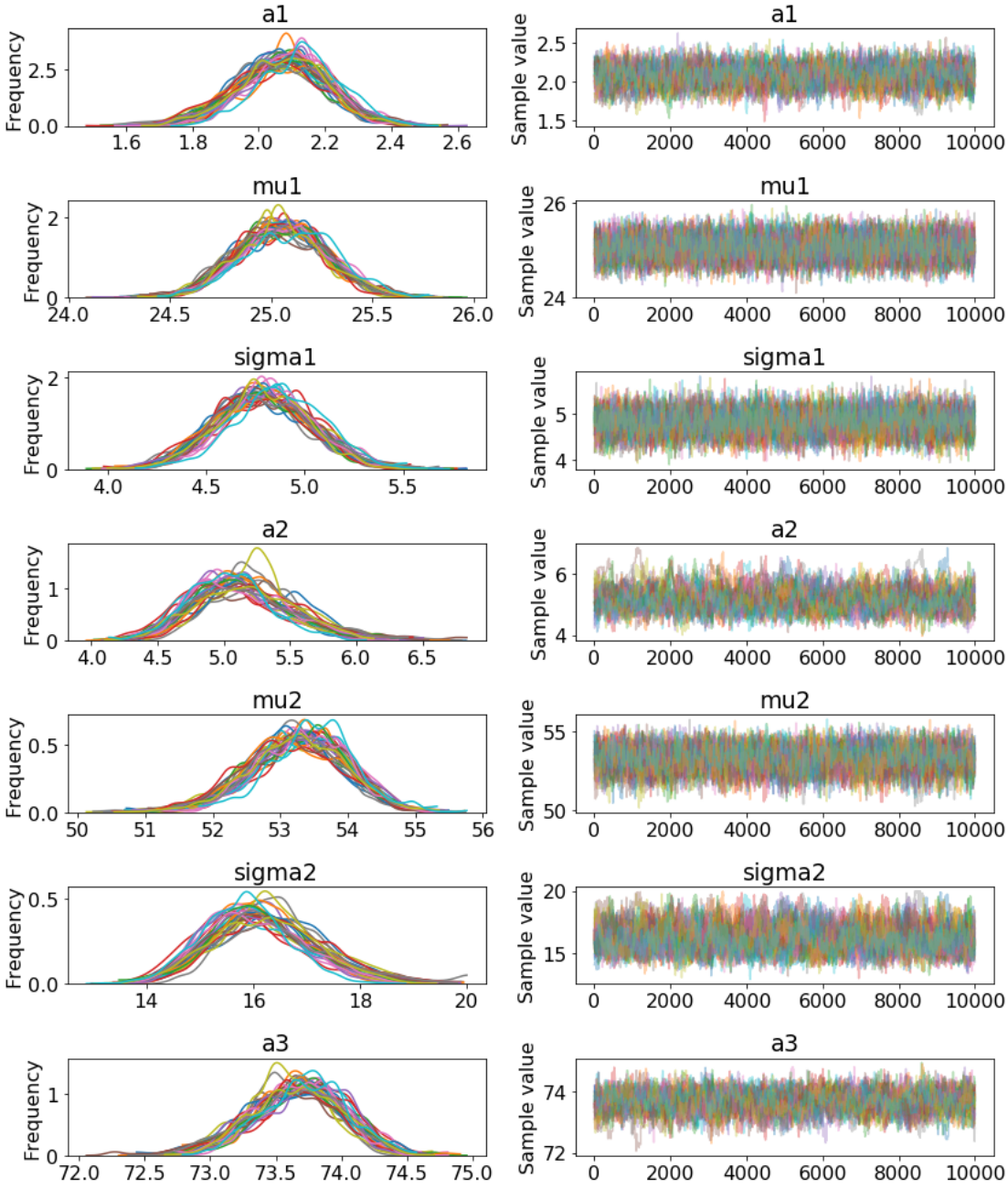

Figure 6.1: The traceplot from using Metropolis-Hastings in `PyMC3` to run MCMC on the toy data set. The traces on the right hand side show that the posteriors are well converged.

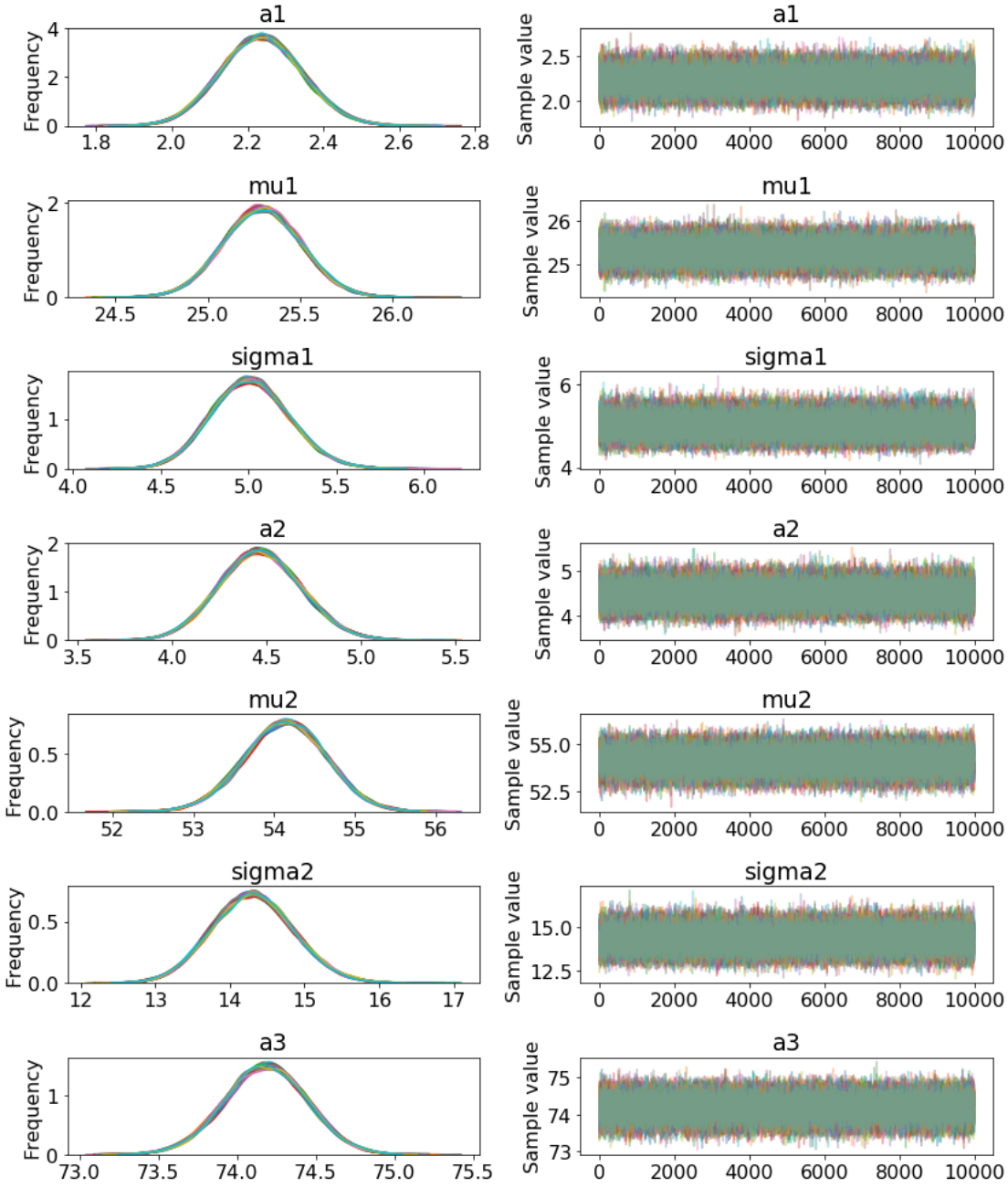

Figure 6.2: The traceplot from using NUTS in `PyMC3` to run MCMC on the toy data set. The traces on the right hand side show that the posteriors are well converged.

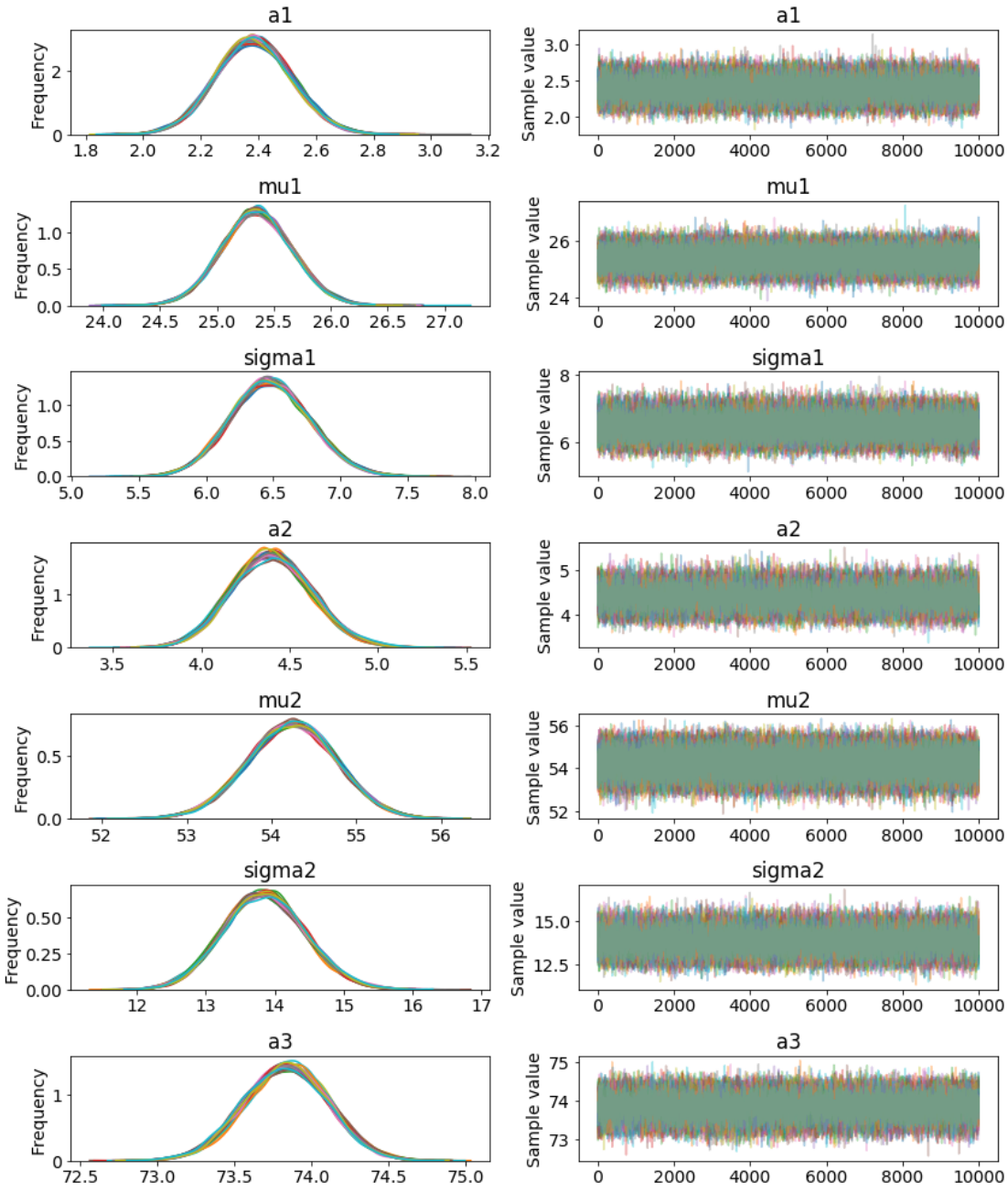

Figure 6.3: The traceplot from using Slice in `PyMC3` to run MCMC on the toy data set. The traces on the right hand side show that the posteriors are well converged.

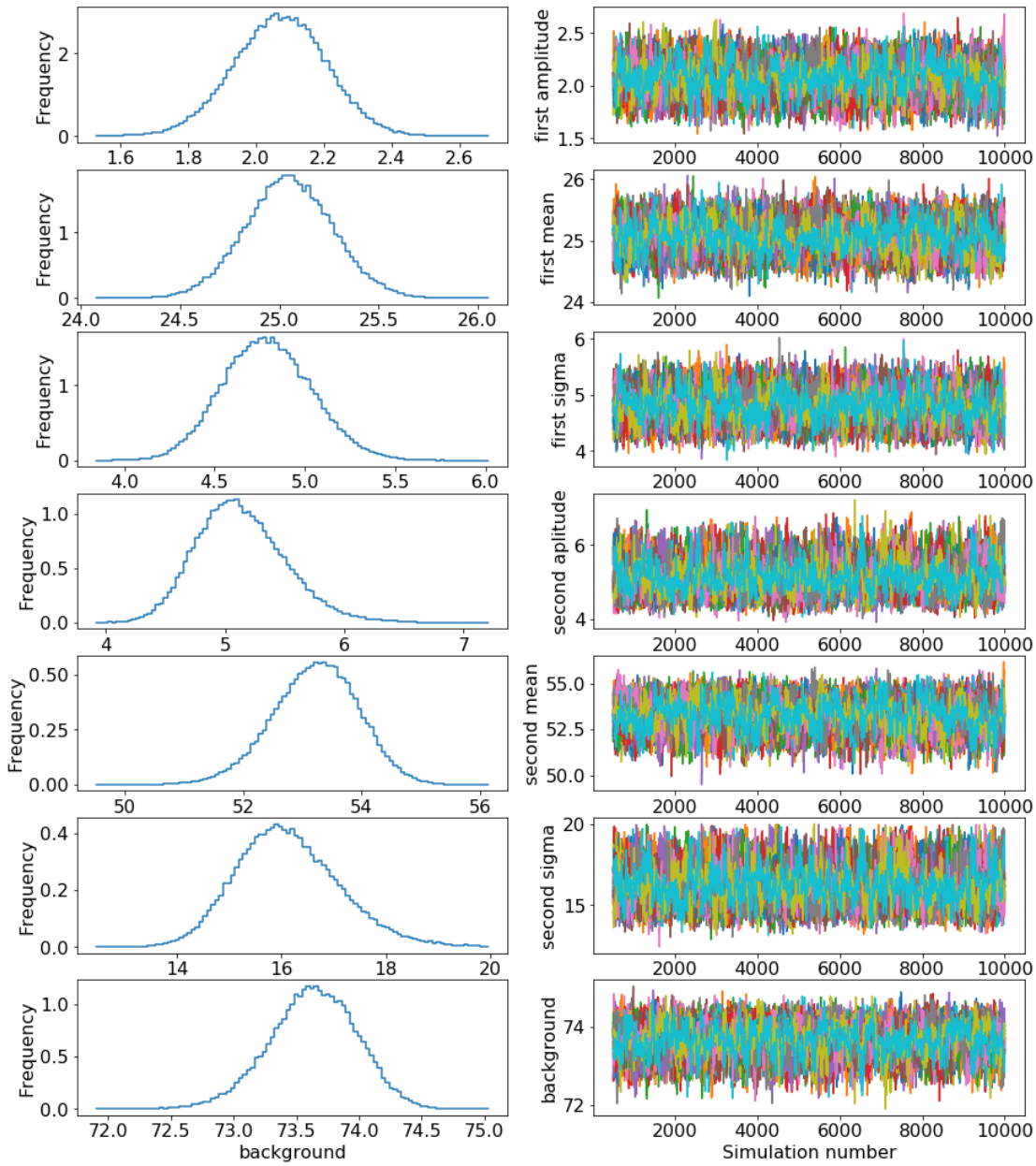

Figure 6.4: The traceplot from using `emcee` to run MCMC on the toy data set. The traces on the right hand side show that the posteriors are well converged.

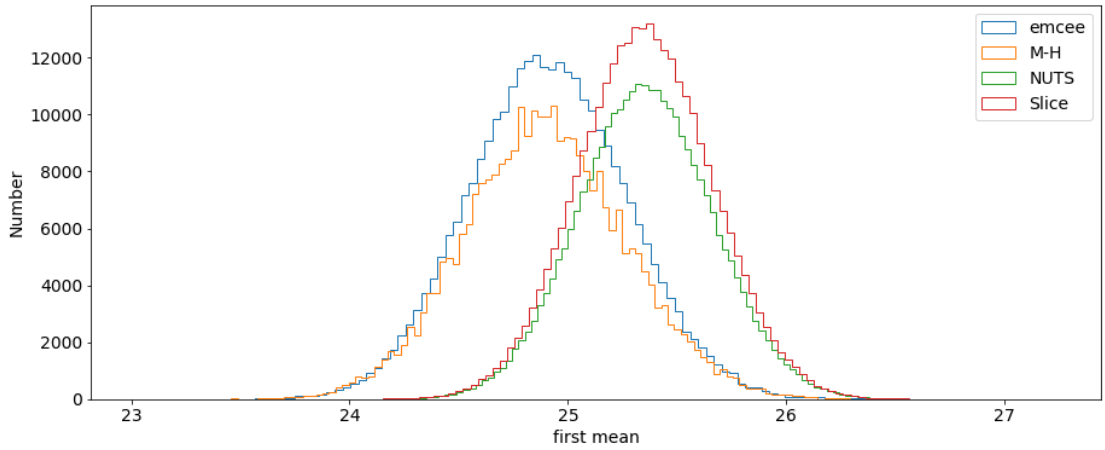

Figure 6.5: The posteriors for the mean of the first Gaussian for each MCMC method. The differences in the posteriors are present for this parameter as well.

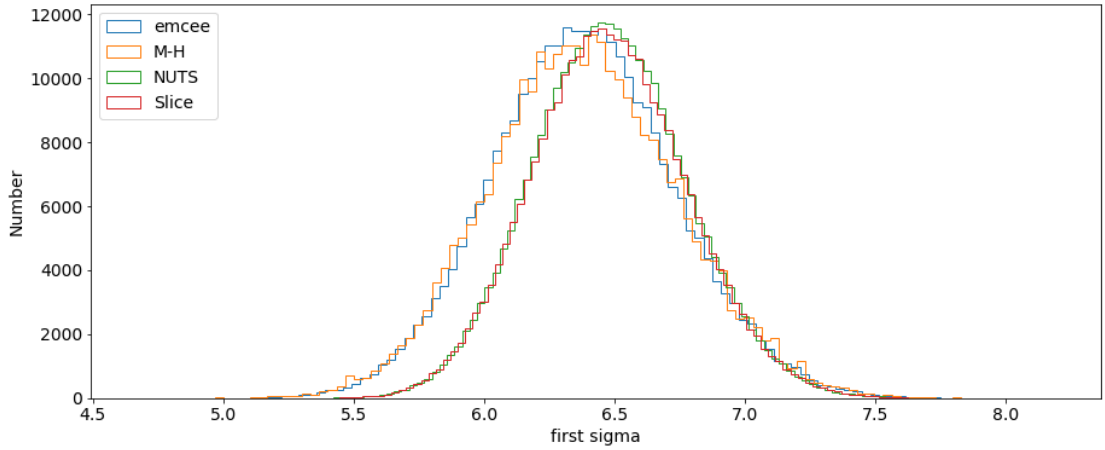

Figure 6.6: The posteriors for the sigma of the first Gaussian for each MCMC method. The differences in the posteriors are present for this parameter as well, although the difference is very minor.

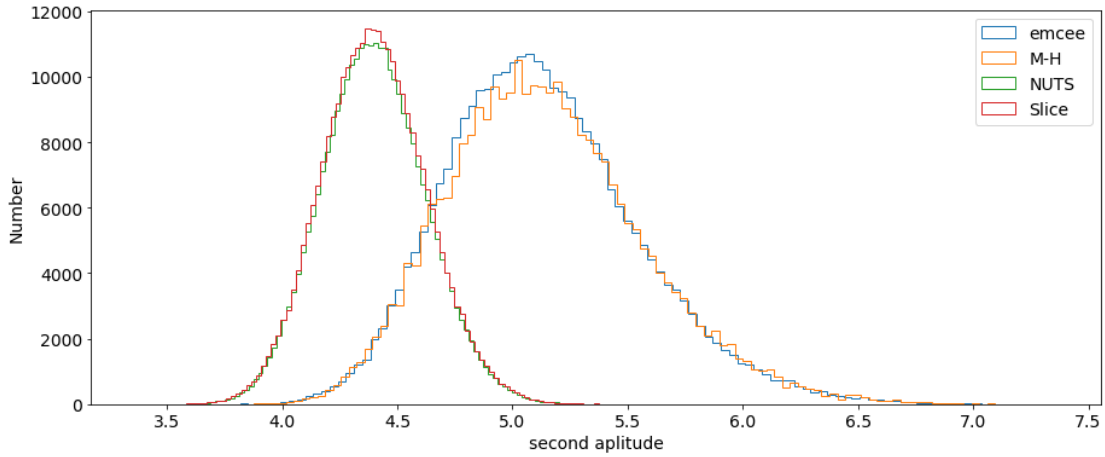

Figure 6.7: The posteriors for the amplitude of the second Gaussian for each MCMC method. The differences in the posteriors are present for this parameter as well.

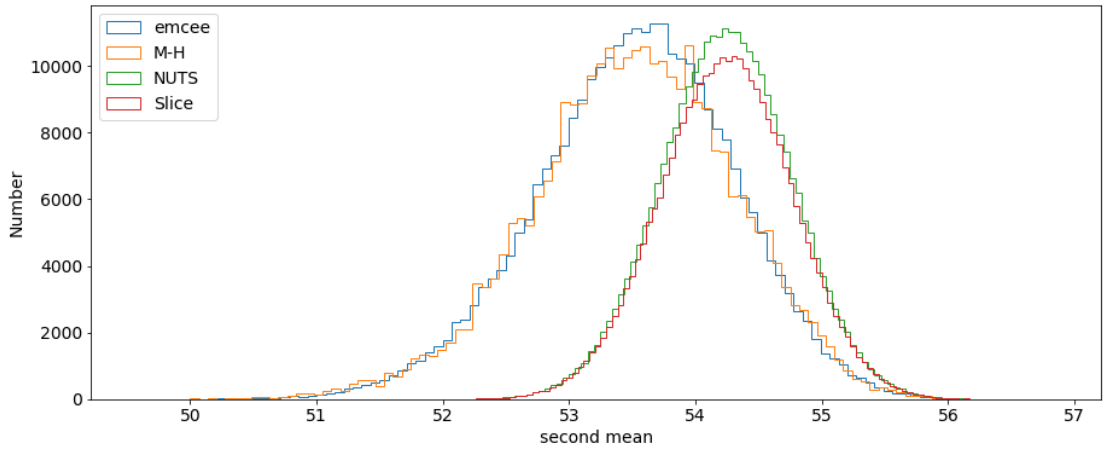

Figure 6.8: The posteriors for the mean of the second Gaussian for each MCMC method. The differences in the posteriors are present for this parameter as well.

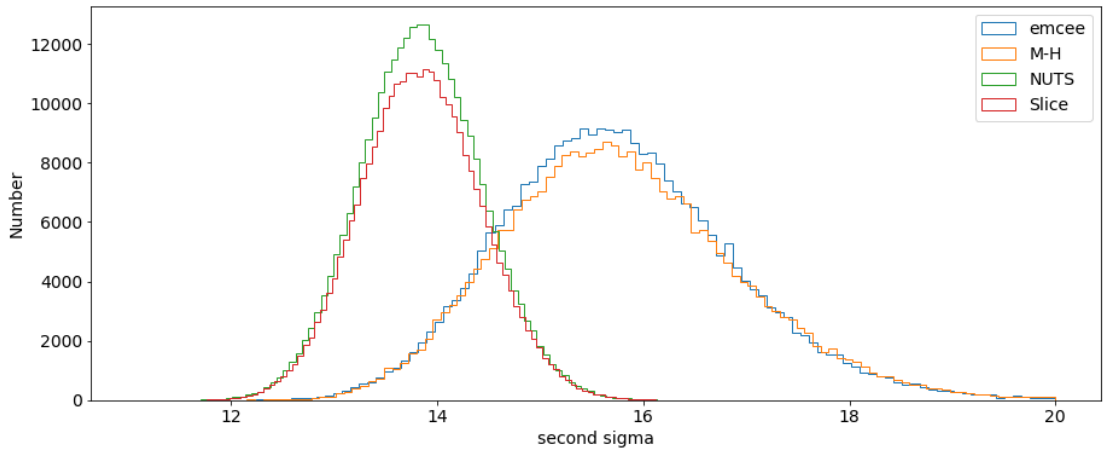

Figure 6.9: The posteriors for the sigma of the second Gaussian for each MCMC method. The differences in the posteriors are present for this parameter as well.

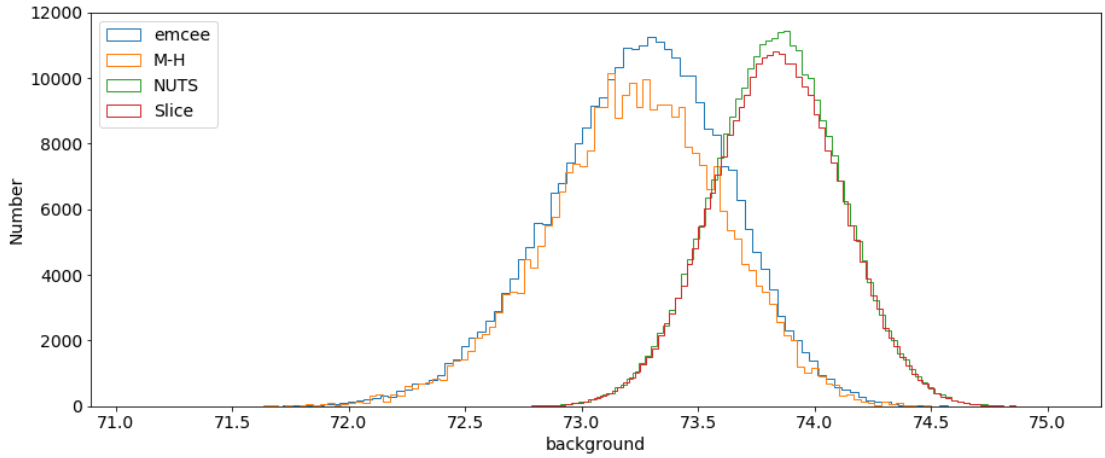

Figure 6.10: The posteriors for the amplitude of the background for each MCMC method. The differences in the posteriors are present for this parameter as well.