IMPROVING READABILITY, ORGANIZATION, AND SPEED IN A MACHINE LEARNING CODEBASE TO ESTIMATE REDSHIFT AND DETECT DAMPED LYMAN-ALPHA ABSORBERS IN QUASAR SPECTRA

By

Matthew Amir Hunt

A capstone project submitted for Graduation with University Honors

May 01, 2022

University Honors
University of California, Riverside

APPROVED

Dr. Christian Shelton
Department of Computer Science & Engineering

Dr. Richard Cardullo, Howard H Hays Jr. Chair
University Honors

ABSTRACT

Quasars in space send light across the universe to hit places including the Earth and light up and hit or scatter based on anything they hit. Quasars continuously undergo a redshift as they travel, and the light given off could be absorbed by gas clouds that they go through. Since there is a constant redshift, the different locations of absorptions in these clouds can tell us how far away they are from us. It is important for understanding the universe to calculate the redshift or the encounter of any Damped Lyman Alpha absorber (DLA). Not only this, but it is also much more efficient to automate the process than to do it by hand such as the current method does. The Garnett et al.'s MATLAB codebase calculates the redshifts and detects DLAs using a machine learning model known as a Gaussian process. In this work, we convert it to Python to make it much more accessible and versatile to future progress. This translation involved many challenges in finding replacement libraries for functions as well as speeding up the code, so it was comparable with the original MATLAB version. Most of the difficulties in the translation were found in specifically learning the Gaussian process model and testing each quasar to see how well the model does. The results of the translation provide a comparably slower but more accessible version of the code that can calculate redshift and detect DLAs within a very efficient time frame.

ACKNOWLEDGEMENTS

I would like to thank my faculty mentor, Dr. Christian Shelton for supporting and mentoring me throughout the creation of this paper and helping with translating the codebase into Python. I also would like to thank Ming-Feng Ho for his help in the background of this codebase as well as specifics with how the MATLAB codebase was created and used. Finally, I would like to thank my friends and family for all the support throughout the year to help me complete this project.

TABLE OF CONTENTS

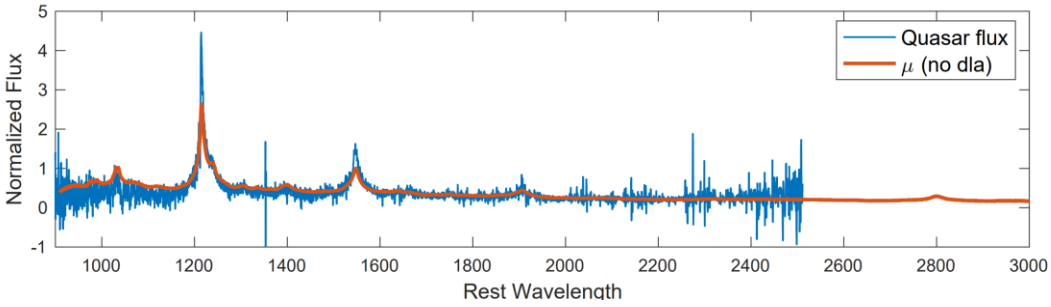
INTRODUCTION or BEGINNING OF CAPSTONE

BACKGROUND

The Sloan Digital Sky Survey catalog on their specific website lists that there are over 500,000 observable quasars in our universe that also have redshifts associated with them ("The Sloan Digital Sky Survey Quasar Catalog: Fourteenth Data Release | SDSS" n.d.). Quasars themselves are supermassive black holes that give off light that travels across the universe in every direction (Bañados et al. 2016). In more detail, the light from quasars is progressively shifted toward the red end of the spectrum as it travels through the universe due to the expansion of the universe (Liu et al. 2020). The broad emission lines they give off from the Broad Line Region (BLR) to understand more gas dynamics, accretion inflow and outflow, and black hole masses in space since they are usually comprised of one element or another, (Średzińska et al. 2017). In this work, we are concerned with using quasars to map and understand Damped Lyman Alpha Absorbers or DLAs which are natural hydrogen gas clouds with a high column density of over 10^{20} cm^{-2} (Ho, Bird, and Garnett 2020). DLAs are important because they are the only probe of high redshift normal galaxies (Garnett, Ho, and Schneider, n.d.).

The Sloan Digital Sky Survey website is a database with spectra information for over three million astronomical objects ("The Sloan Digital Sky Survey Quasar Catalog: Fourteenth Data Release | SDSS" n.d.). In catalog 12 alone, there are 2.5 million results in plates and spectroscopy made from light data that came into earth and has been filtered for quasars (Alam et al. 2015). The data is in forms of observations using spectroscopy which are necessary to ascertain quasars, study their position in detail, and improve the redshift estimates (Bikmaev et al. 2020). There is also the filtering and masking that they do to their own data that puts it into a much more numerical form for totals like quasar numbers or automated redshift data after using some piping analysis software (Alam et al. 2015).

The automation of DLA detection and redshift calculation is very important for the field of astronomy especially because the best method of calculation before this was through visual inspection which was very slow and cumbersome. (Lyke et al. 2020). There are several different analysis pipelines possible. In this work we focus on the use of Gaussian processes to model the quasar emission spectra as the previous work did (Fauber et al. 2020). These Gaussian processes give coherent results when applied to problems with multiple Gaussian random variables and allows for an approximation for the data (Rasmussen and Williams 2006). Gaussian processes are a machine learning technique that provide a Bayesian function interpolation which helps for understanding the underlying attributes or function and obstacles that the quasar light hit while traveling to the Earth (Ho, Bird, and Garnett 2020).



(Fauber et al. 2020)

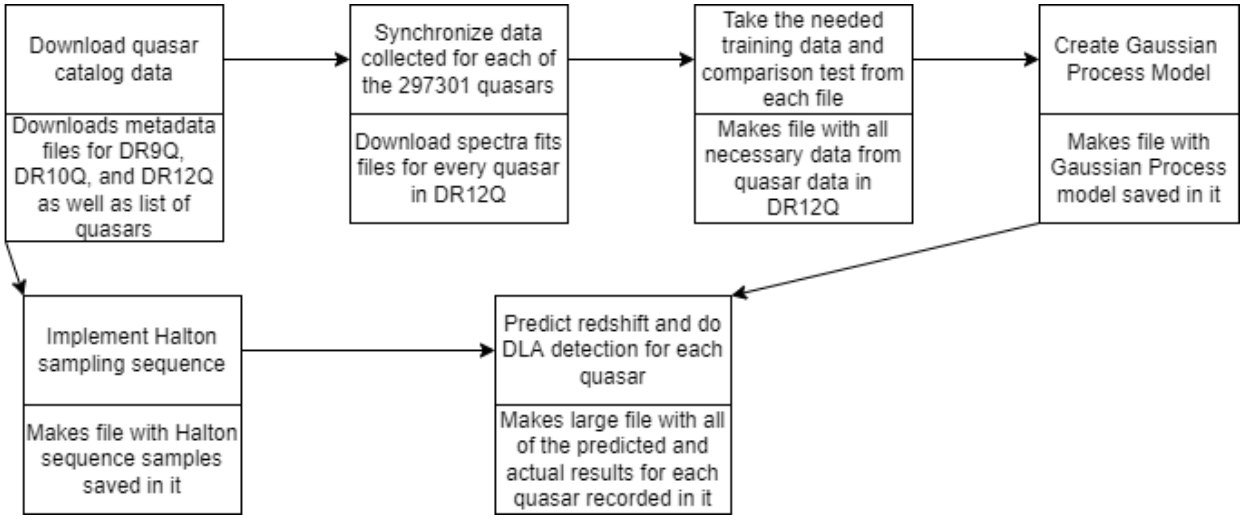
This graph from the MATLAB codebase's research paper shows how given quasar flux in the rest frame the Gaussian process can fit the data and give estimates for the quasar redshift and DLA properties (Fauber et al. 2020). The end result also gives a column density distribution function that is comparable to N12 according to the research done already as well as a good ROC curve for the redshifts (Fauber et al. 2020).

The current automation in MATLAB shows how accurate Gaussian processes are compared to other common methods that could be used (Fauber et al. 2020). This codebase also

builds on previous analysis and filtering the quasar data in *The Sloan Digital Sky Survey Quasar Catalog: Sixteenth Data Release* (Lyke et al. 2020). The method combines both DLA detection and quasar redshift is because these two calculations go hand in hand with how light from quasars gets absorbed by DLAs when they hit them (Parks et al. 2018).

FLOW OF CODEBASE

The codebase itself starts by downloading and storing all the data needed for learning and testing the model locally and then parsing the correct info from the fits files that the astronomical data is stored in. After it does this, it creates the learned Gaussian process model and then saves that for testing the redshift and DLA detection accuracy with a comparison with the catalog. The code performs inference to determine the distribution over the redshift of the quasar and the redshift of the DLA through sampling. This generates a posterior conditioned on the data distribution estimate for the quantities, from which the maximizing values can be selected as estimates for the redshifts.



This flowchart shows the overall steps that need to be executed to perform the redshift and DLA calculations. It also shows each part of the codebase that I modified in order as well.

All files use dill, pickle, numpy, pathlib, and os for file manipulation and the like to transition between modules. The first file is download_catalogs.py and it first imports libraries such as pandas and astropy to hold and manipulate data. It also imports the libraries ssl, wget, and tarfile to help in downloading and storing the files in directories for the codebase. Creating the directory hierarchy as well as the important files to print for the next phase are the main goals of this module. The next module performs the synchronization with the rsynch command that is best just used on the command line for ease of access and takes about three to four hours to download everything. Now that everything is downloaded, it can be parsed and consolidated into a single file in the next module preload_qsos.py. This script imports astropy and resource to read, write, and save important values from the quasar spectra data. The preloaded_qsos file that is created from this module is directly used by the actual creation of the gaussian process in learning.py.

This module specifically has functions that it needs to work with imported that fit the end model, calculate spectra loss, and work as the objective function for minimization. The learning code itself also requires the usual ones, but also requires interpolation and optimization from scipy and sklearn as well to process each quasar before optimization. It uses Principal Component Analysis (PCA) which is a statistical technique for multivariate problems to reduce dimensions and size of information to interpret it better while minimizing information loss. PCA is used to initialize the matrix M that holds the important information for the model (Geetha and Divya 2021). The log-likelihood of the data is then maximized by tuning the M matrix to produce the best possible result for the Gaussian process. The result gets saved in a learned_model_data file after it calculates the maximized results that help for DLA detection and redshift calculation.

Testing requires a Halton sequence to be used to create the samples used to estimate where on the Probability Distribution Function (PDF) of the spectra is to reverse it back in time for redshift calculation (Fauber et al. 2020). That module uses scipy for everything it does from integration to creating a PDF between distributions to fit the data that is stored. The Halton sequence is quasi-random which is more efficient than completely random sequences (Fauber et al. 2020). The final module of testing imports all the files that were created in the previous modules as well as scipy and numba. Scipy is needed for interpolation of datapoints and to calculate a base model for the voigt profile to be used in calculating the probability of a certain quasar having a DLA. Numba is used to speed up the functions that are called 100000 times by precompiling them beforehand. Precompiling them on top of using classes and function calls allows for comparable calculations of the new database to the old MATLAB database. The predictions and actual results are saved in a processed_qsos file that can then be appended to the original catalog to have the predictions for each observation.

CODE ORGANIZATION

There are also some general key differences between this new codebase in Python and Garnett et al.'s MATLAB codebase. One of these differences is that the beginning modules that do the downloading and synchronization are all in Python in the new one while they were command line files in the old one. This makes sure that from the codebase is in the same language from beginning to end now which improves its organization. Another difference is using object-oriented programming with classes to help keep things a lot more organized, especially in the testing code with an inner for loop. A third difference between the two codebases is the set_parameters file that has changeable values for the other modules. In the MATLAB version, the file must run to load the variables into local memory before running any

of the other files. The Python version creates a file and loads all the needed parameters without need for extra user input before each one. Running them directly makes it easier to get to the result without overreliance on external instructions. It also reduces the amount of overhead or rerunning that would have to be done in the original MATLAB version. Some functions and classes are also separated out into different files in the Python version which helps keep things clean and readable to future researchers working with this codebase. Following good object-oriented programming rules and separation of concerns really helps to make the codebase much more accessible to astronomers using it.

There are also libraries used in the Python version that had to approximate the functionality of the MATLAB libraries. The scipy library used optimization for minimization in the code file that learns the Gaussian process. The MATLAB version uses minFunc which is written to be compiled in MATLAB with mex (Schmidt 2005). The scipy function does take a different path compared to minFunc but is comparable when the L-BFGS-B method is used. This still allowed the results to be compared and used in the testing code eventually. Another organization choice was making the code in the download_catalogs.py that creates the directory into a function so that it did not need to be copy and pasted multiple times like in the MATLAB codebase. By contrast, the download_spectra.py file is almost the same as the original codebase's version. Interpolation from scipy and solving matrix equations is also very different between MATLAB and Python which was one of the major contributors to the longer amount of time that the Python takes to execute. In general, MATLAB is a more optimized and faster when dealing with matrix math. Some of the libraries used are meant as replacements for the libraries used in MATLAB such as wget and astropy. Most of the functionality is very similar in terms of the output variables although it does also bring up a large point behind numerical accuracy between

the two codebases. The Python code creates the space to hold the large numbers like most programming languages while MATLAB holds the matrices directly on the CPU during some steps of the algorithm which has more space for significant figures and numbers which propagates throughout all the calculations that are made. It is important to remember that both languages do hold variables in data structures in general. This is what gives differences in numbers in the tenths and hundredths places for the p_dla and p_no_dla posterior probability variables in the testing code. For the most part, these differences do not completely change the answer that comes out of the testing code. It is very helpful for debugging and reading the different aspects of the codebase by adding in classes, functions, and reducing redundant code from being reran.

OPTIMIZATIONS

After translation of the codebase from MATLAB to Python, three files needed to be specifically optimized to complete in a timely manner somewhat comparable to the original codebase. One of these files was preload_qsos.py which unpacks the synchronized fits files and then parses and saves information to a large file at the end. In this module, 297,301 files are opened and closed in a row which provides one major problem of too many file handles open at the same time which needed to be fixed. The original method used to open and close these files was with a *with open* command which automatically closes the file after the indent in Python. This provides problems with so many file handles being used so it needed to be changed to a more direct closing.

```
#measurements = fitsread(filename, 'binarytable',  1, 'tablecolumns', 1:4)
try:
    hdl = fits.open(fileName, memmap=False)
    measurements = hdl[1].data
finally:
    hdl.close()
```

The finally means that every opened fits file is now force-closed in the loop so that the problem with file handlers no longer occurs. Testing and figuring this out required some adjustments with file handle limits as well.

```
soft, hard = resource.getrlimit(resource.RLIMIT_NOFILE)
resource.setrlimit(resource.RLIMIT_NOFILE, (hard, hard))
```

Making sure the limit was at the hard limit instead of soft allowed it to go farther but was not as helpful as changing the code to close file handles with the *finally* keyword was.

The next file that had needed serious modifications was the learning file that created the Gaussian process. It specifically had problems by taking too long in the objective and spectrum loss function that were needed to minimize the log-likelihood of the M matrix.

```
@njit
def spectrum_loss(y,lya_1pz,noise_variance,M,omega2,c_0,tau_0,beta,num_forest_lines,all_transition_wavelengths,all_oscil
```

For example, using njit, which compiles spectrum_loss beforehand, saves a lot of execution time because spectrum_loss is called 100,000 times in the objective function.

```
for i in range(num_quasars):
    ind = (~np.isnan(centered_rest_fluxes[i, :]))
    #print("ind", ind, ind.shape, np.count_nonzero(ind))

    # Apr 12: directly pass z_qsos in the argument since we don't want
    # zeros in lya_1pzs to mess up the gradients in spectrum_loss
    zqso_1pz = z_qsos[i] + 1
    #print("zqso_1pz", zqso_1pz)

    [this_f, this_dM, this_dlog_omega, this_dlog_c_0, this_dlog_tau_0, this_dlog_beta] = spectrum_loss(centered_rest
```

Saving time on each of these function calls really can make the difference between weeks and months of compute time to finishing the minimization. The code had to be changed to allow numba, the library that allows precompilation of functions in Python, to work with it in terms of numpy functions. Analyzing the time that each iteration takes also required a callback function which is called every iteration as well. Checkpointing with this code is of particular importance since it takes weeks to finish execution. Saving the result of each objective function call can help the process be restarted and resumed without spending all the time necessary that it did in the beginning. This is also utilized in the testing code as well.

```
def genvoigtapprox(minvel=-9000000000,maxvel=+90000000000,npts=300000):
    from scipy.interpolate import interp1d

    sigma = 9.08537121627923800e+05 # Gaussian width          cm s⁻¹        */
    #/* Lorentzian widths       cm s⁻¹
    gammas = np.array([6.06075804241938613e+02, 1.54841462408931704e+02, 6.28964942715328164e+01, 3.17730561586147395e+01,
                       1.82838676775503330e+01, 9.15463131005758157e+00, 6.08448802613156925e+00, 4.24977523573725779e+00,
                       3.08542121666345803e+00, 2.31184525202557767e+00, 1.77687796208123139e+00, 1.39477990932179852e+00,
                       1.11505539984541979e+00, 9.05885451682623022e-01, 7.45877170715450677e-01, 6.21261624902197052e-01,
                       5.22994533400935269e-01, 4.44469874827484512e-01, 3.80923210837841919e-01, 3.28912390446060132e-01,
                       2.85949711597237033e-01, 2.50280032040928802e-01, 2.20224061101442048e-01, 1.94686521675913549e-01,
                       1.73082093051965591e-01, 1.54536566013816490e-01, 1.38539175663870029e-01, 1.24652675945279762e-01,
                       1.12585442799479921e-01, 1.02045988802423507e-01, 9.27433783998286437e-02])
    vels =   np.linspace(minvel,maxvel,npts)
    #moreParams.num_lines = 3
    voigts = voigt_profile(vels[:,None],sigma,gammas[None,:3])

    myapprox = [interp1d(vels,voigts[:,i]) for i in range(3)]
```

The testing code deals with a voigt profile approximation which is called for 100,000 iterations for each quasar tested. The voigt profile is used to model and analyze the the radiative transfer in the atmosphere which here is the light from the quasars and is like the squared

exponential kernel. The first thing we tried was to make the code into Cython so that it could be compiled and run faster. While this did increase speed a bit, it was very ineffective to use and did not speed up the time spent calculating the voigt_profile. It also made the code organization more clunky by requiring extra files and compilation steps that were cumbersome in the original codebase. It was necessary to speed up the execution time of voigt so we just made a one-dimensional interpolation using scipy's interp1d function that is like the voigt profile curve. In terms of accuracy, it is also a proficient approximation since there is the idea that the approximation is built on the original voigt_profile function and there is no major problems between the two. There is a slight possibility that any edge cases could be problematic, but since the underlying graph is relatively smooth, this does not actually introduce meaningful problems into the system.

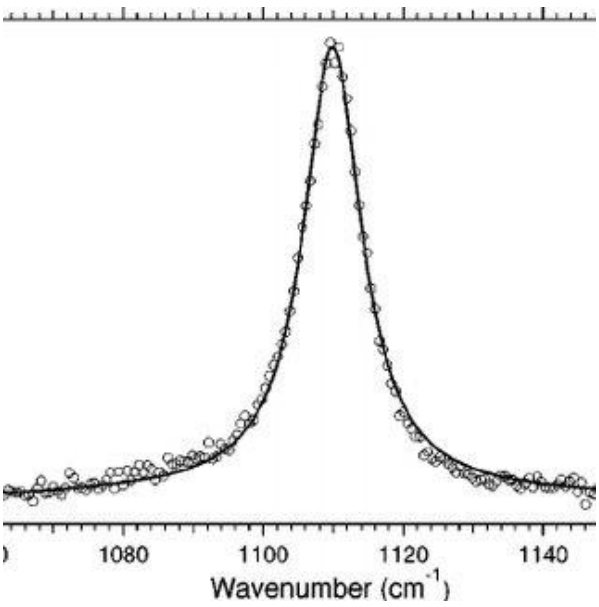


Figure shows an example of what the voigt profile graph looks like and how an example of data points being mapped onto it similar to what is done in this codebase (Datchi 2004).

The lookup time for an interpolation of this function which is one-dimensional (f(x)) is less costly then the scipy function by about 30%. This improvement is one important factor for

making sure the estimation for redshift and DLA parameters for each quasar takes as little time as necessary. It was also critical to be able to use numba on the log_mvnpdf_iid function that computes the multivariate normal distribution and log_mvnpdf_low_rank which computes a low rank approximation of it instead. Figuring out the timing for these two parts of the code required a profiled analysis of the execution of one quasar in Python. One last thing that seemed to improve the execution time per quasar in the testing code was separating the inner for loop and variables into different classes to allow the program to work better with multiprocessing. These improvements got the overall time per quasar from over two hours to around a half an hour in total.

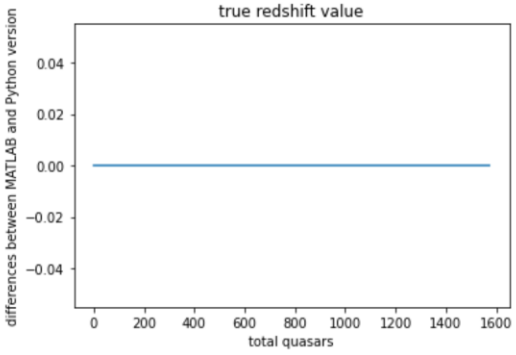
The timing table was made after analyzing different methods regarding the learning code for the Gaussian Process. It found that the code took 4 times as long as the MATLAB code did to complete the needed 4000 iterations. This means there needed to be changes when it came to different things like solving equations involving matrices. Cholesky Decomposition originally takes advantage of the form the matrices that come in to save computation time, but this saving is not done in Python as shown by the table. Skipping over the decomposition allowed an improvement of about 200 seconds. The numba njit also allows for a much faster version for this spectrum_loss function of about 300 seconds saved. This is the version that is used to create the model. There was added code that involved changing numpy arrays to tensors used by the torch library in Python, but some unit testing on it resulted in worse results than using the precompiled version. It is also coupled with checkpointing to allow intermediate results to be saved and used as the model due to the incredible amount of time needed for each iteration. Another key

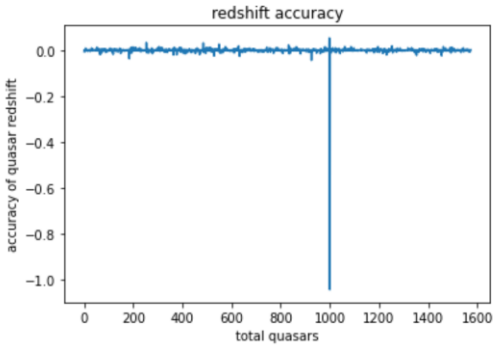
optimization that needed to be made was splitting up the quasar testing calculations into ten different jobs running at the same time. Doing this significantly reduced the total 30-50 days required to complete the redshift and DLA estimation to 5 days.

Precompiled Cholesky version	940.0832874774933 s
Precompiled no Cholesky version	724.1295721530914 s
Interpreted scipy version	752.8827285766602 s
Interpreted No Cholesky version	1110.870219707489 s
Interpreted Cholesky version	956.1211833953857 s
Torch tensor version	889.5732045173645 s

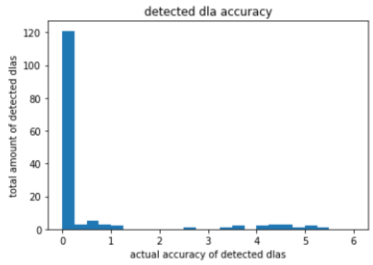
RESULTS

The results between the two codebases in testing the quasars is very simple to compare with three major parts. The three parts are the Maximum A Posteriori (MAP) estimate of the redshift of the quasar, the posterior probability of the DLA in the spectrum, and the MAP estimate of the DLA redshift. For example, the z_true values, which are taken from the database, show that the files were loaded in correctly.

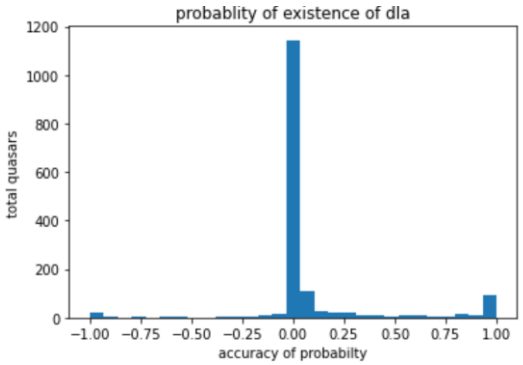


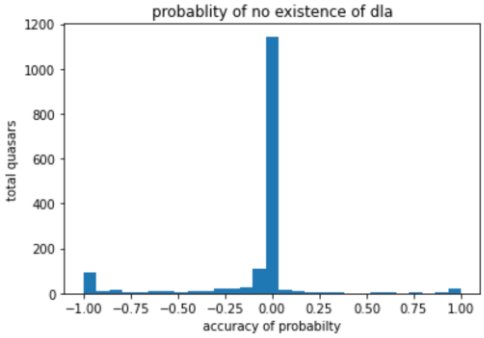


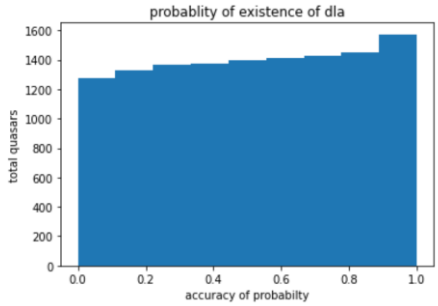
The left plot shows how both codebases use the same quasars since the difference between the z_true values is exactly 0 for all 1589 of the tested quasars. The right plot shows that the Python codebase has good accuracy for the redshifts for all the quasars except one outlier. Taking both together shows that the redshift MAP calculations are entirely accurate in the newer codebase with the bonus of being more accessible now as well. MAP redshift calculation is not all that the MATLAB codebase did either. It also calculated posterior probabilities and redshifts of DLAs that the quasar light encountered on the way to Earth. Both codebases have a similar plot for the dla_true variable representing the assumption that the quasar has encountered a DLA. Using these values with the calculated DLA redshift z_dla_map gives the following plot.



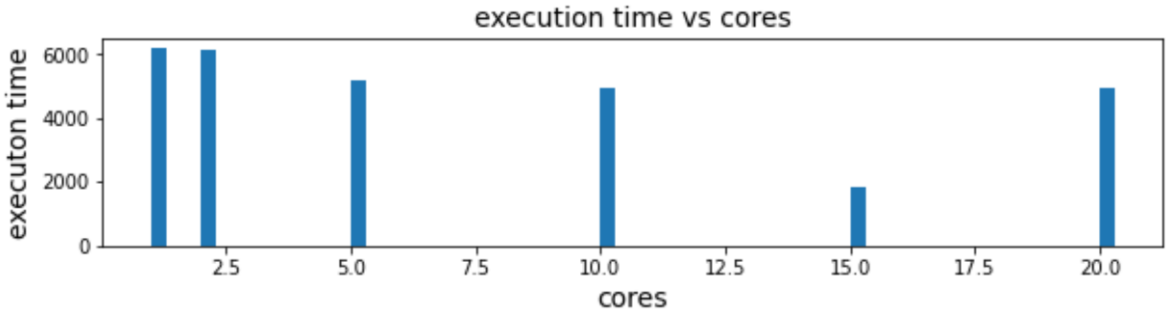
This plot shows that for the DLA calculations that mattered, about 150 quasars, most of the calculations are accurate between the two codebases with 120 near 0 meaning that 20% of the calculations are inaccurate by as much as 5.5. There are similar results when it comes to the posterior probability of DLA detection as well.



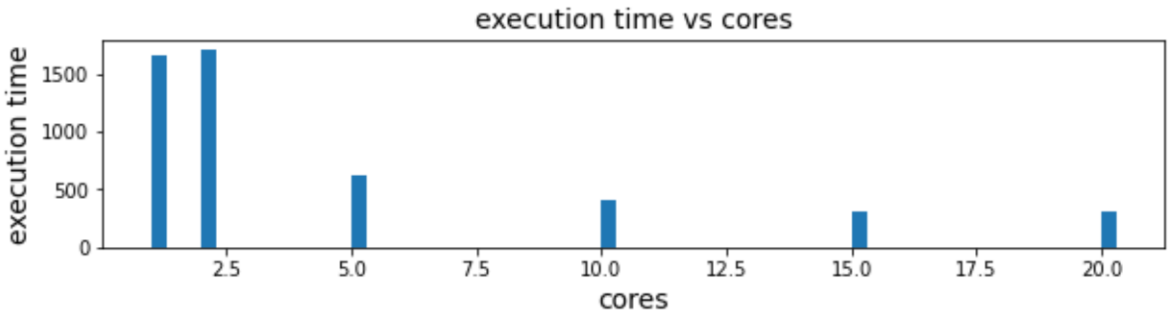




The left and right plots show the differences between the p_dlas and p_no_dlas calculations in the Python and the MATLAB codebases. They are mirror images since the probability that there is no DLA encountered by the quasar light is just one subtracted the probability the quasar light does encounter a DLA. The bottom plot shows the cumulative distribution of the posterior probabilities for the DLAs showing very simply how about 80% of the calculation differences are close to zero. It can also be seen that about 20% of the calculations are inaccurate with more false positives than false negatives. This means that the translation from MATLAB to Python is very accurate for the redshift calculations, and only about 80% accurate for the DLA detection and DLA redshift calculations. One final difference is in the length of time taken regarding multiprocessing execution on multiple cores.



This bar graph shows how depending on the number of cores used how the Python version improves with about half the execution time and goes up when too many cores are used. In comparison, the MATLAB has much more consistent and better results.



The MATLAB code has ¼ of the total time taken per quasar with more cores than the Python version does lending those calculations are more optimized for multiprocessing in that codebase. Even so, it should be considered that there are some meaningful differences between the results in Python and MATLAB. Some such differences are the amount of time the learning minimization code ran for being much shorter in Python and the Halton sequence being different in the Python version as well. Nonetheless the numbers are easily comparable showing the validity of this translation.

CONCLUSIONS

The differences in the results may be due to a few possibilities. One is that there is a bug in the translated code that seriously affects the DLA calculations. If some calculation error was uncaught in the testing code, it is probable that 20% of the calculations could come out wrong. Future work with the Python codebase could take this into account and do more checking at different sections of the code. Another important consideration is the difference in the minimization functions between the two codebases. As described earlier, MATLAB uses minFunc to optimize the model while Python uses L-BFGS-B method which does produce slightly different results. The number of iterations was a lot less for the Python version as well, with only 1000 while the MATLAB used 4000 to optimize the model. This can easily propagate to the DLA calculations. Checking the differences between the learned MATLAB model and

learned Python model could show if that is a major factor in the errors in some of the calculations. We might also investigate whether the errors are due to numeric instability in the common algorithms in both codebases. Further analysis into this could reveal possible problems or problems in robustness of the Gaussian process that may also be present in the MATLAB codebase.

It could be that the tested quasars that detected DLAs in the Python version and no DLAs in the MATLAB version could fit the model a lot less in terms of their flux and rest wavelength. Finally, the underlying Halton sequence being different in different places could give differences in the Python version for some of the quasars while most of the ones that matched came out closer to zero as shown in the plots for p_dlas and p_no_dlas. Checking what happens with a directly taken Halton sequence from MATLAB could give some worthwhile results if this is something that has affected the calculations for some of the quasars.

Some important next steps that can be taken involve analyzing using a direct array of data from MATLAB for the Gaussian Process and for the Halton sequence separately and then together to see how integral to the calculations those differences are. Another step that can be taken is taking the subset of quasars that gave different DLA detection probabilities than expected and then saving their fluxes and rest wavelengths to analyze how well they correspond with the model like the above graph. Some problems between known emission lines could show underlying instabilities that lead to the differences in the calculations of DLA redshifts or detection probabilities. The Python database can be improved overall as well. Some more research into the inner *for loop* could seriously reduce the time it takes to analyze each quasar binging the total amount of time needed for the Python closer to the MATLAB version. Currently the MATLAB version takes approximately ten minutes per quasar while the Python

version takes 30-45 minutes for each. Some applications and research specifically in using pytorch which is a python library specifically designed for use in GPU would speed up a lot of the testing calculations for each quasar if the code is written correctly.

Overall, the codebase has been replicated through the testing of the quasars with main differences only in creating the Gaussian process model, generating the Halton sequence, and testing the accuracy of each quasar in calculating the redshift and detecting any DLAs. While the original codebase in MATLAB managed to accomplish its tasks of calculating all these important features related to quasar spectra data, it was also proprietary and redundant in many areas requiring rerunning a set_parameters.m file if MATLAB was closed in between any steps. There is also some slight problems with it that have been fixed in the Python version such as the process_qsos.m file redefining all_exceptions and all_posdeferrors after the checkpoint code was loaded in ruining the ones saved in the checkpoint file. Keeping things in classes and clear as just seven files to run in order makes it a lot less of a headache to interpret and work with. Improving the readability of the codebase will help further work know what areas to change and experiment with as well. Making key changes in organization and optimizing the Python code to work more similarly in terms of speed and accuracy will really allow for a much larger path now to working with analyzing quasar spectra data using machine learning.

REFERENCES

Alam, Shadab, Franco D. Albareti, Carlos Allende Prieto, F. Anders, Scott F. Anderson, Brett H. Andrews, Eric Armengaud, et al. 2015. "The Eleventh and Twelfth Data Releases of the Sloan Digital Sky Survey: Final Data from SDSS-III." *The Astrophysical Journal Supplement Series* 219 (1): 12. https://doi.org/10.1088/0067-0049/219/1/12.

Astropy Collaboration, A. M. Price-Whelan, B. M. Sipőcz, H. M. Günther, P. L. Lim, S. M. Crawford, S. Conseil, et al. 2018. "The Astropy Project: Building an Open-Science Project and Status of the v2.0 Core Package." *The Astronomical Journal* 156 (September): 123. https://doi.org/10.3847/1538-3881/aabc4f.

Bañados, E., B. P. Venemans, R. Decarli, E. P. Farina, C. Mazzucchelli, F. Walter, X. Fan, et al. 2016. "THE PAN-STARRS1 DISTANT z $\greater$ 5.6 QUASAR SURVEY: MORE THAN 100 QUASARS WITHIN THE FIRST GYR OF THE UNIVERSE." *The Astrophysical Journal Supplement Series* 227 (1): 11. https://doi.org/10.3847/0067-0049/227/1/11.

Bikmaev, I. F., E. N. Irtuganov, E. A. Nikolaeva, N. A. Sakhibullin, R. I. Gumerov, A. S. Sklyanov, M. V. Glushkov, et al. 2020. "Spectroscopic Redshift Determination for a Sample of Distant Quasars Detected by the SRG Observatory Based on RTT-150 Observations. I." *Astronomy Letters* 46 (10): 645–57. https://doi.org/10.1134/S1063773720100047.

Datchi, Frédéric. (2004). Raman spectrum of cubic boron nitride at high pressure and temperature. Phys. Rev. B. 69. 10.1103/PhysRevB.69.144106.

Fauber, Leah, Ming-Feng Ho, Simeon Bird, Christian R. Shelton, Roman Garnett, and Ishita Korde. 2020. "Automated Measurement of Quasar Redshift with a Gaussian Process." *Monthly Notices of the Royal Astronomical Society* 498 (4): 5227–39. https://doi.org/10.1093/mnras/staa2826.

Garnett, Roman, Shirley Ho, and Jeff Schneider. n.d. "Finding Galaxies in the Shadows of Quasars with Gaussian Processes," 9.

Geetha, K., and S. Divya. 2021. "Principal Component Analysis of Sorghum Landraces for Yield Contributing Traits." *Electronic Journal of Plant Breeding* 12 (3): 1033–36. https://doi.org/10.37992/2021.1203.142.

Ho, Ming-Feng, Simeon Bird, and Roman Garnett. 2020. "Detecting Multiple DLAs per Spectrum in SDSS DR12 with Gaussian Processes." *Monthly Notices of the Royal Astronomical Society* 496 (4): 5436–54. https://doi.org/10.1093/mnras/staa1806.

Liu, Tonghua, Shuo Cao, Marek Biesiada, Yuting Liu, Shuaibo Geng, and Yujie Lian. 2020. "Testing the Cosmic Opacity at Higher Redshifts: Implication from Quasars with Available UV and X-Ray Observations." *The Astrophysical Journal* 899 (1): 71. https://doi.org/10.3847/1538-4357/aba0b6.

Lyke, Brad W., Alexandra N. Higley, J. N. McLane, Danielle P. Schurhammer, Adam D. Myers, Ashley J. Ross, Kyle Dawson, et al. 2020. “The Sloan Digital Sky Survey Quasar Catalog: Sixteenth Data Release.” *The Astrophysical Journal Supplement Series* 250 (1): 8. https://doi.org/10.3847/1538-4365/aba623.

Parks, David, J. Xavier Prochaska, Shawfeng Dong, and Zheng Cai. 2018. “Deep Learning of Quasar Spectra to Discover and Characterize Damped Lya Systems.” *Monthly Notices of the Royal Astronomical Society* 476 (1): 1151–68. https://doi.org/10.1093/mnras/sty196.

Rasmussen, Carl Edward, and Christopher K. I. Williams. 2006. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press. http://www.gaussianprocess.org/gpml/chapters/RW.pdf.

Schmidt, M. minFunc: unconstrained differentiable multivariate optimization in Matlab. http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html, 2005.

Średzińska, J., B. Czerny, K. Hryniewicz, M. Krupa, A. Kurcz, P. Marziani, T. P. Adhikari, et al. 2017. “SALT Long-Slit Spectroscopy of Quasar HE 0435-4312: Fast Displacement of the Mg II Emission Line.” *Astronomy & Astrophysics* 601 (May): A32. https://doi.org/10.1051/0004-6361/201628257.

Su, Yanrui, Li Zhang, Bin Jiang, Jiaqi Liu, and Fabao Yan. 2019. “The Quasars’ Redshift Estimation Method Based on Piecewise Gaussian Fitting.” *International Journal of Distributed Sensor Networks* 15 (5): 1550147719847128. https://doi.org/10.1177/1550147719847128.

“The Sloan Digital Sky Survey Quasar Catalog: Fourteenth Data Release | SDSS.” n.d. Accessed March 8, 2021. https://www.sdss.org/dr14/algorithms/qso_catalog/.