PHYS 788 - Statistical tools for Astronomers

University of Waterloo

Final Project Report

Written by:
Darshak Patel (ID: 20780253)

Date: April 22^{nd}, 2024

1 Summary of Assignments

1.1 Assignment #1:

The goal of the first assignment was to understand the numerical stability of the covariance matrix, calculate and plot the χ^2 distributions, and correct with the Hartlap factor in numerical covariance matrices. We were provided with a reference model and a true analytical covariance matrix, from which we generated noisy Gaussian datasets and calculated χ^2 values via the difference between the reference model and the noisy models using the following equation: $\chi^2 = D^T C^{-1} D$. We expect that at high degrees of freedom (k), the χ^2 distribution tends to be Gaussian, with the expected mean as k and the variance as 2k. Figure 1 shows the χ^2 distribution for noisy dataset 1, where the mean was found to be 899.96 and the variance is 1795.49. This is within expectation.

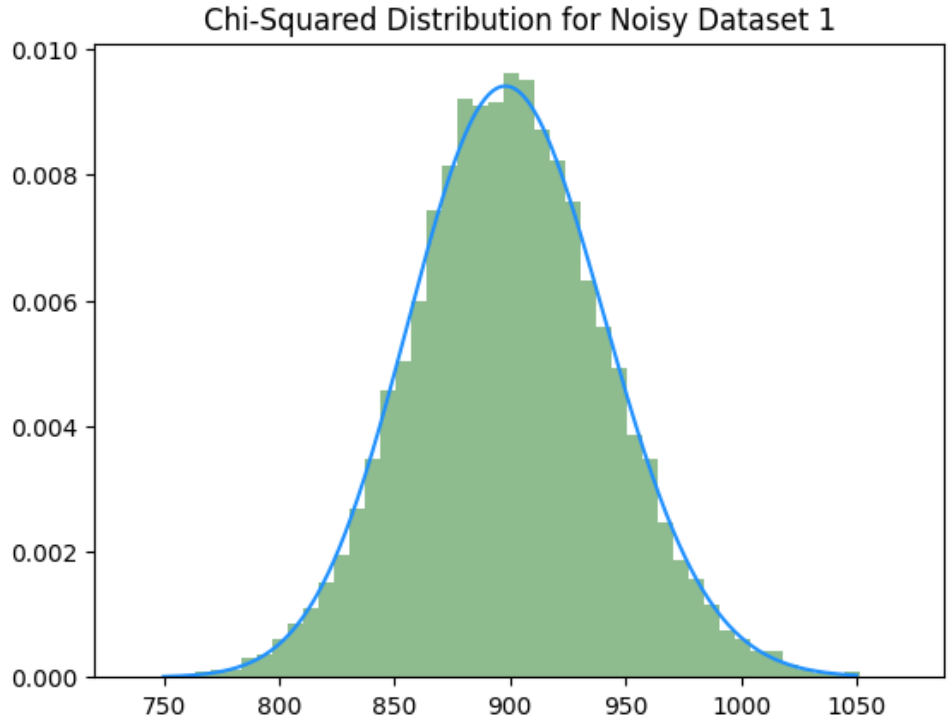

Figure 1: The χ^2 distribution for noisy dataset 1, which has a mean of 899.96 and variance of 1795.49. The light blue line is a Gaussian with a mean of 900 and a variance of 1800.

Following this, several numerical covariance matrices were generated from one of the noisy datasets with different amounts of realizations. Plotting the χ^2 distributions exposed the biased nature of these matrices. We learned that we must debias the inverse covariance matrix: $\tilde{C}^{-1} = C^{-1}/h$, using the Hartlap factor h, given by $h = \frac{n-1}{n-m-2}$. Figures 2 and 3 show the χ^2 distribution before and after applying the Hartlap factor for the 10k realization numerical covariance matrix. Applying the Hartlap factor fixes the mean

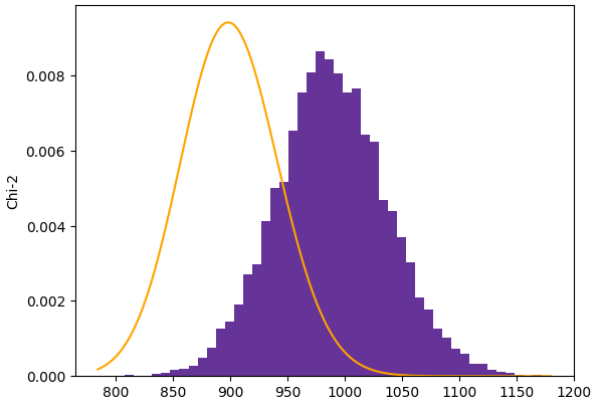

Figure 2: The χ^2 distribution for the 10k data vector numerical covariance matrix is shown. The orange line represents the expected distribution (as seen in Figure 1), where the mean is 900 and the variance is 1800. Here, the mean is 988.54 and the variance is 2372.37

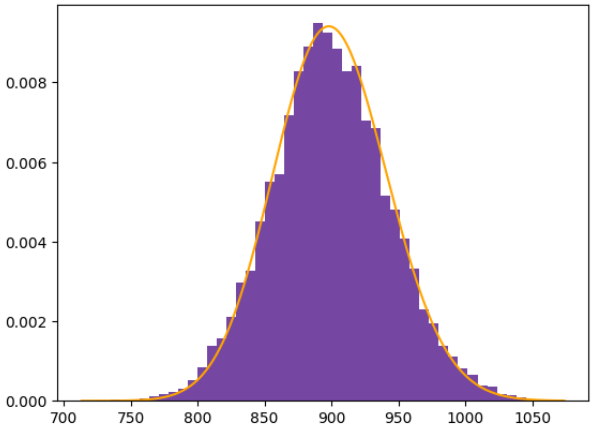

Figure 3: The χ^2 distribution after applying the Hartlap factor for the 10k numerical covariance matrix is shown. The mean is now 899.46 and the variance is 1964.09.

to converge to 900, but the variance requires many more realizations to converge. The key takeaways were: always check that your covariance matrix is invertible and positive semi-definite, always apply the Hartlap factor correction, and never calculate χ^2 values against the same data from which you generated the numerical covariance matrix.

1.2 Assignment #2:

The main focus of assignment two was to understand and train a neural network emulator to make accurate model predictions and to perform Principal Component Analysis or PCA data compression to see the power it holds in obtaining constraining power. Using Cosmopower to train our emulators, we determined appropriate values for our hyperparameters via trial and error. My final trained emulator consisted of 5 layers with learning rates of $10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$. The batch size used was 500, and the maximum number of epochs was set to 1000. Each hyperparameter impacts the overall accuracy and speed of the emulator in various ways. For example, the learning rate should not be too high, or else the optimizer will jump around and never find a minimum (whether that is a local or global minimum). As a result, the loss will fluctuate a lot but never seem to converge. We also learned that normalizing our data significantly improves the accuracy of our emulator. Figure 4 is a plot of the difference between the emulated features and the test features divided by the standard deviation. This plot demonstrates that the emulator is well-trained, and any deviations observed (on the black line) away from 0 are on very small orders of magnitude.

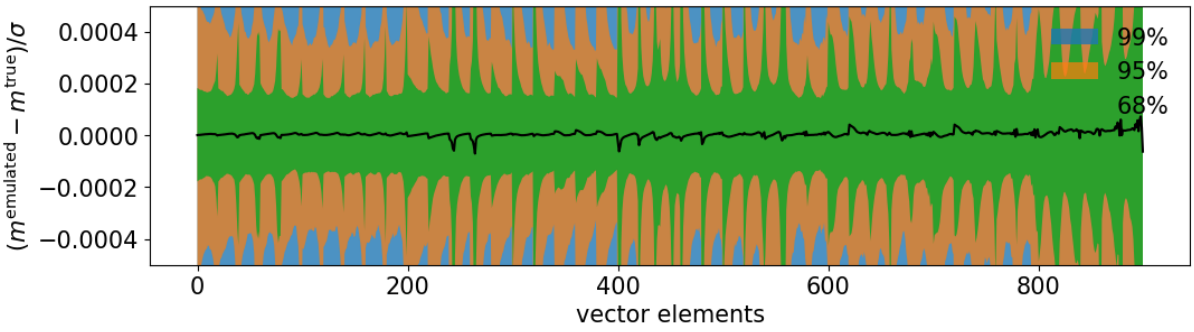

Figure 4: The measure of the accuracy of the trained emulator shows that deviations away from 0 are on a very small order of magnitudes, and the 1σ contour is, at worst, 0.04%

The second aspect of this assignment was PCA analysis. We used Fisher analysis/matrices to obtain the maximum constraining power values for Ω_m and w. Looping through all PCA elements, we plotted how the constraining power for each parameter changed (see Figures 5 and 6). We found that we only need about 500 and 100-200 PCA elements to obtain 1% and 10% maximum constraining power.

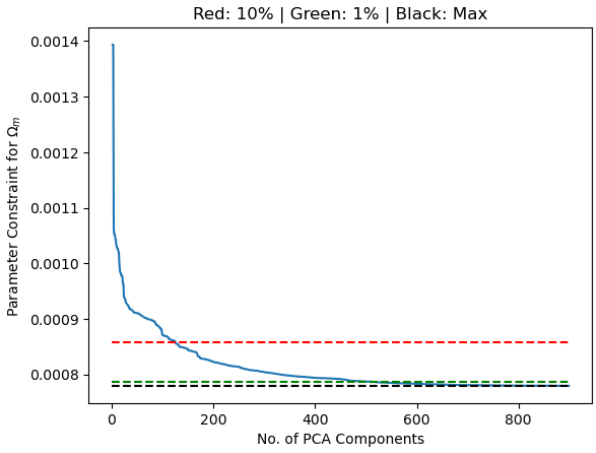

Figure 5: Plot of the constraining power for Ω_m as a function of PCA elements. The Black dashed line represents the max constraining power where the red and green are the 10% and 1% max constraining power.

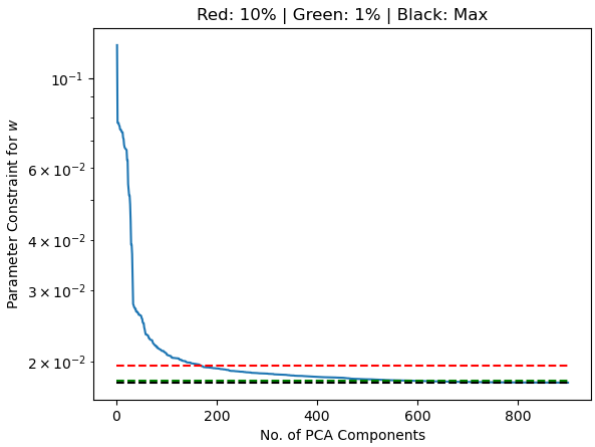

Figure 6: Plot of the constraining power for w as a function of PCA elements.

1.3 Assignment #3:

In the final assignment, we ran various Monte-Carlo Markov Chains (MCMCs), to find best-fit parameters for Ω_m, Ω_b, A_s, and w. We ran an MCMC using various numerical covariance matrices (Hartlap corrected), PCA + Hartlap, and using a noise-free reference model. Figures 7 and 8 show the posterior distribution and χ^2 distribution from a normal MCMC run. We make note that the χ^2 distribution peaks above the degrees of freedom, which is 896. This is due to the nature of MCMC walkers; as they are always "jumping around," as they get closer to the minima, they never will hit the minima perfectly.

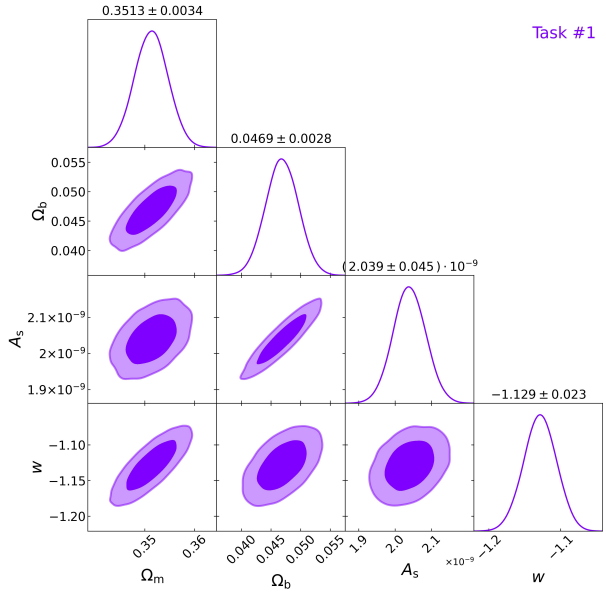

Figure 7: Posterior distribution from a normal MCMC run using Emcee with 3000 total steps, 1000 burning steps, and 100 walkers. We see that for all 4 parameters we obtain Gaussian distributions and converged contours/constraints.

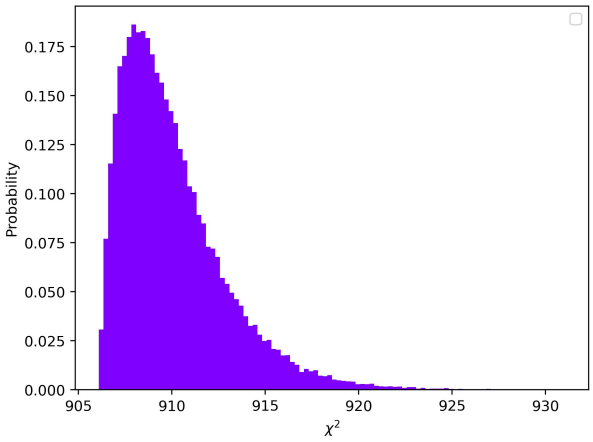

Figure 8: The χ^2 distribution for the posterior distributions shown in Figure 7. Despite the peak being at a χ^2 value larger than 896, this distribution is still within expectation as discussed in the previous paragraphs due to the nature of walkers.

Figures 9 and 10 below show the posterior distributions obtained when running the MCMC using three different numerical covariance matrices but applying the Hartlap factor correction only in Figure 9. The takeaway from these results was that the Hartlap factor works to broaden the posterior distributions. This effect is more pronounced with the red contours for the 1.5k numerical covariance. This is expected since the value of h increases as n decreases, meaning the Hartlap factor is correcting more. Figures 11 and 12 are the result of our investigation of the effects of PCA and PCA+Hartlap. The main takeaway is that PCA data compression is very effective when using the analytical covariance matrix as we see that PCA 600 is nearly identical to PCA 900 in Figure 11.

The other takeaway is that PCA data compression and the Hartlap factor have opposing effects when used together. As we decrease the number of PCA elements, m goes to 0, and so, h goes to 1. This means that there is less of a correction being applied by the Hartlap factor. PCA wants to broaden the posterior distributions as we decrease the number of elements, but due to the lack of correction by the Hartlap factor, it wants to tighten the constraints. We find there is a "sweet spot" between these two effects between 550-600 PCA elements; above which the constraints get slightly larger again. Figures 13 and 14 show the constraining power for two of the four parameters - Ω_m and A_s as a function of PCA elements used. We see that after about 550 PCA elements the constraints slightly increase again (kind of hard to make out but you can see it!). This increase occurs after we pass the "sweet spot".

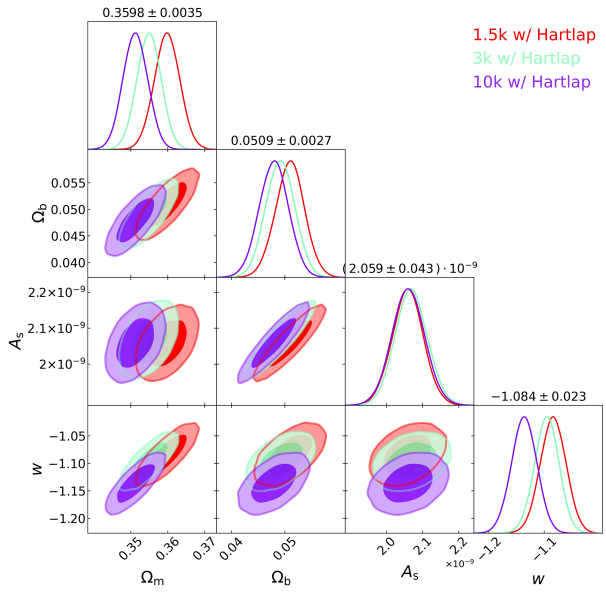

Figure 9: Posterior distributions for an MCMC run using three different numerical covariance matrices (1.5k, 3k and 10k) and applying the Hartlap factor correction.

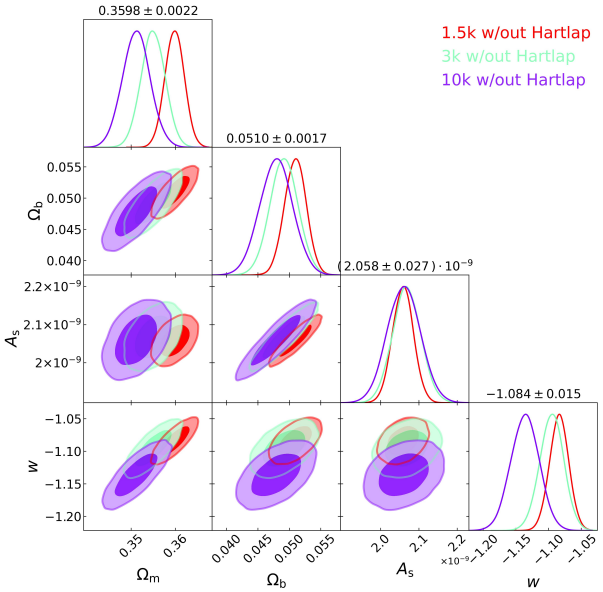

Figure 10: Posterior distributions for a MCMC run using three different numerical covariance matrices (1.5k, 3k and 10k) without the Hartlap factor correction.

2 Methods Used

Data compression

We use PCA as our chosen method for data compression as compared to something like Massively Optimised Parameter Estimation and Data Compression (MOPED). The main reasons for this choice had to do with implementation. Compared to MOPED, PCA is far simpler and easier to understand and implement. Additionally, PCA is more easily interpretable in that we can plot and visualize the eigenvectors and see which ones

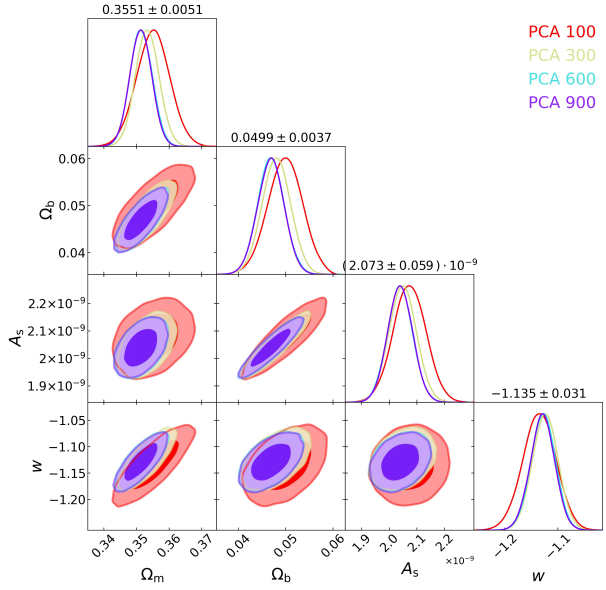

Figure 11: Posterior distributions for an MCMC run using the analytical covariance matrix but applying PCA data compression for different numbers of PCA elements.

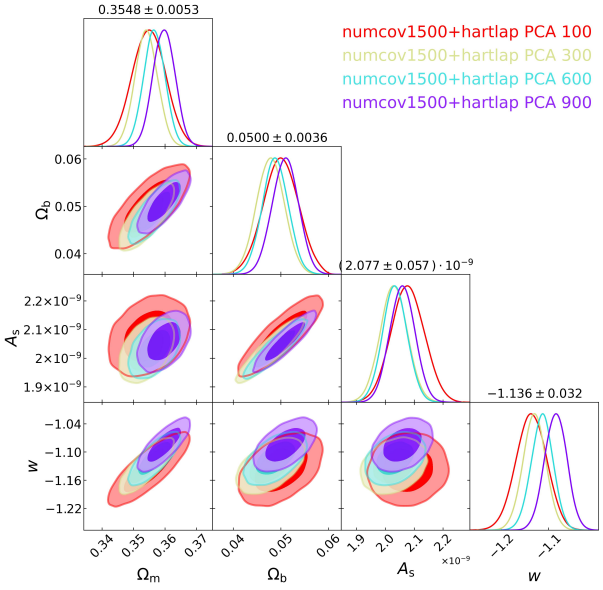

Figure 12: Posterior distributions for an MCMC run using the 1.5k numerical covariance matrix, applying both Hartlap factor correction and PCA data compression.

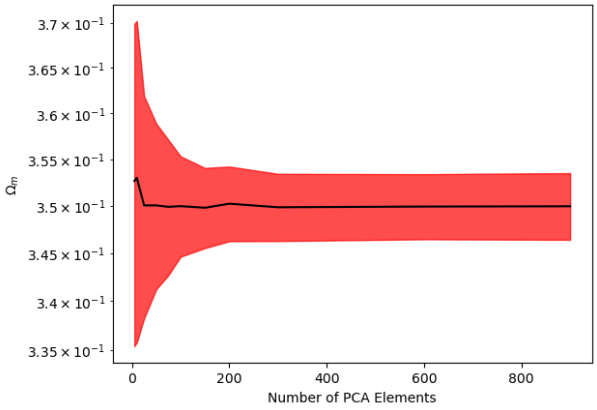

Figure 13: Constraining power of Ω_m as a function of PCA elements.

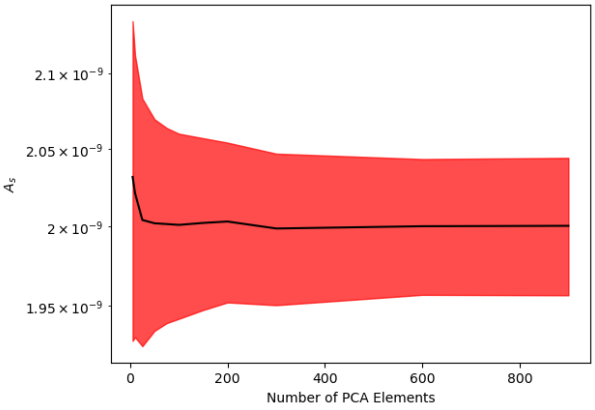

Figure 14: Constraining power of A_s as a function of PCA elements.

are most important in preserving data information. A big disadvantage of PCA, which wasn't an issue in this project, is that PCA does not handle multicollinearity whereas MOPED can work with data where multiple variables are correlated with one another. Choosing the number of PCA elements that is appropriate for the research being done is dependent strongly on the data you are working with. It is quite obvious that taking too few PCA elements will result in severe loss of information and poor constraining power whereas, taking too many will defeat the purpose of data compression. A scree plot is a

helpful aid in visualizing and determining the most effective number of PCA elements to use.

MCMC

The choice of using Emcee for MCMC can be primarily attributed to familiarity. Tensorflow has the benefit that it is better optimized to run multiple chains in parallel, thus, decreasing computation time as compared to Emcee. One disadvantage is that using tensorflow would require everything to be put in tensor formats, which might be challenging and take more time than necessary in comparison to using a more straightforward API with Emcee. Tensforflow also demands a lot more memory than Emcee. There exist other MCMC packages, such as zeus, but we would have had to test these other alternate options to see whether they are faster/more efficient.

For the MCMC that was inevitably run, the hyperparameters chosen were 3000 total steps, 1000 burning steps, and 100 walkers. The choice of these parameters was somewhat arbitrary in that I wanted to ensure that there were enough walkers to sample the parameters space, but also, enough total steps so the walkers could converge without any issue of them potentially getting stuck in some local minimum.

3 Connection to Personal Research

In my research, the statistic of interest is $\Delta\Sigma$, the measure of the excess surface mass density which encodes information about the halo mass galaxies reside in. The current pipeline I use utilizes the Jackknife method to output errors for my $\Delta\Sigma$ profiles and covariance matrices. In regards to the modelling side of my research, I am using a conditional luminosity-based HOD model to populate mock halos with mock galaxies. This model has 13 base parameters and generating $\Delta\Sigma$ profiles from simulation has proven to be computationally expensive. Thus, I have trained an emulator and with this emulator have run multiple MCMCs to determine best-fit parameters for my HOD model. Figure 15 is one of many MCMCs that I have run so far in my research. What I've learnt is that MCMCs are not going to be perfect first try. As shown in the figure, not all of my model parameters have converged; many have hit the edges of their prior ranges. So MCMC is an interactive process in that with each successive run we can better understand each parameter, make adjustments to the emulator and prior ranges, and re-run the MCMC. This is a very powerful tool!

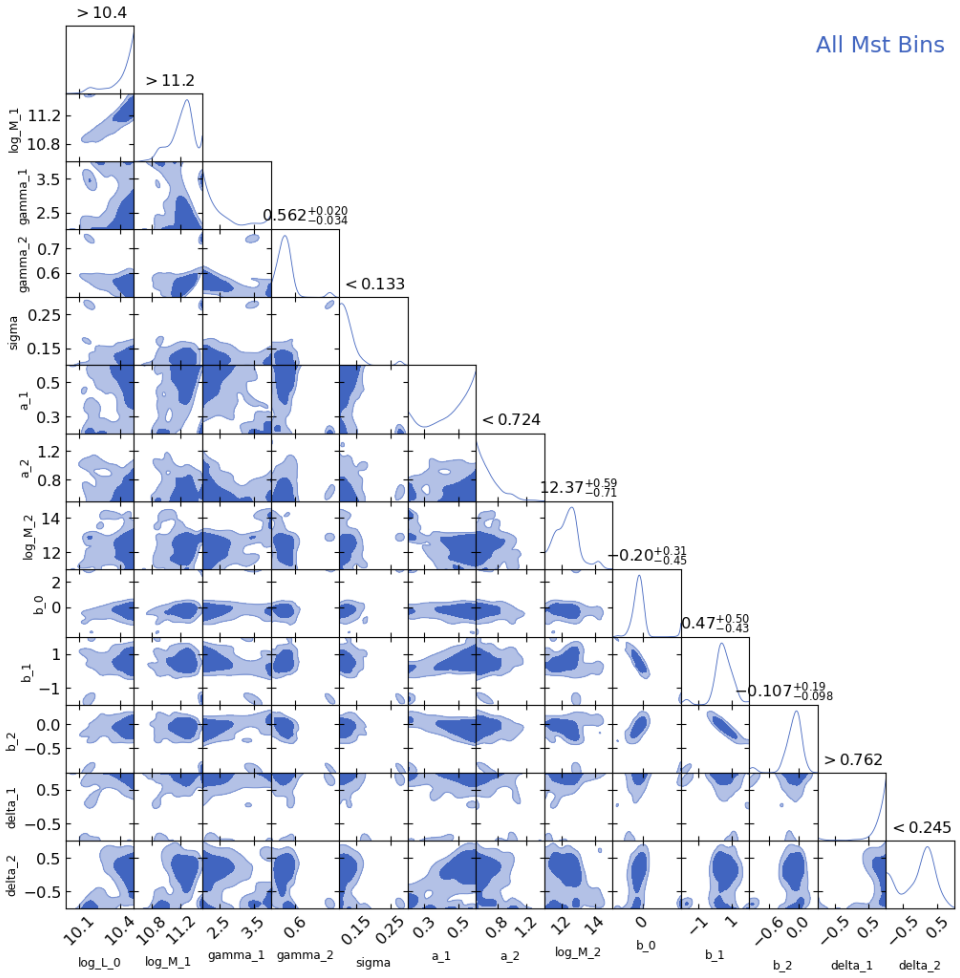

Figure 15: Posterior distributions for one of my many MCMC runs to find best-fit model parameters for my HOD model.