1. a) We have the following equation:

$$z - z_0 = a((x - x_0)^2 + (y - y_0)^2). \tag{1}$$

We can pick a new set of parameters to make this equation linear in the parameters by expanding the square brackets:

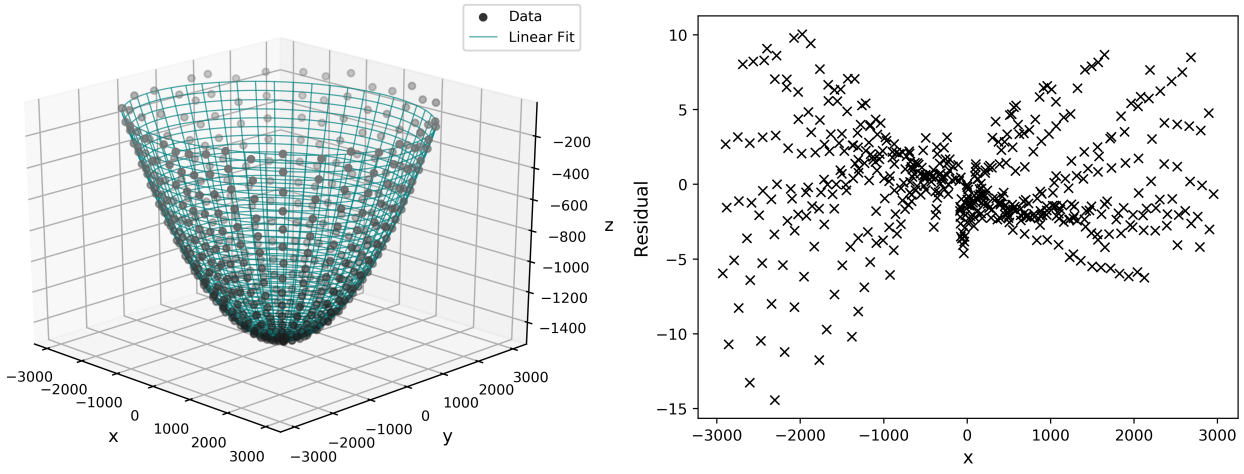
$$z - z_0 = a(x^2 - 2xx_0 + x_0^2 + y^2 - 2yy_0 + y_0^2). \tag{2}$$

From this we see that there are only two terms that are non-linear in our model parameters. Therefore, we can write a new parameter as $r_0 = x_0^2 + y_0^2$ and we can simplify to get rid of the coefficients of 2 writing $x_0' = -2ax_0$ and $y_0' = -2ay_0$. Furthermore, we can take all constants out and call them one variable $c_0 = ar_0 + z_0$.

$$z = a(x^2 + y^2) + x_0'x + y_0'y + c_0. \tag{3}$$

Now our model is linear in all our parameters and we can carry out a fit. We need to solve for x_0', y_0', c_0 and a. Furthermore, in our program we define a for x^2 and y^2 to be different to test whether the fit provides an accurate answer (i.e. ideally both of the a's should be the same). a_1 is for x^2 and a_2 is for y^2.

b) We now carry out the fit in "problem_1.py". We use the regular technique with no noise matrix whereby, $m = (A^T A)^{-1}(A^T \vec{z})$. In our case the matrix A can be defined with the following columns $A = [1, x^2, x, y^2, y]$ as these are all the variables that have coefficients we are interested in finding. Furthermore, note that we have written the usual $\vec{x}$ as $\vec{z}$ as the z values are what we will be multiplying by. The details of the code can be found in the python file. The result is shown in the following plots:



(a) Plot of the predicted paraboloid vs. data.

(b) Plot of the residuals in z against x-coordinates.

Figure 1: Plot of the Zenith dish data vs. the fit provided by the method of linear least squares.

We see that the residuals grow as x gets larger as expected. Interestingly, the error gets larger when x becomes more negative. The parameters and their associated errors from the covariance matrix are shown below (units have been ignored):

$$c_0 \approx -1,512.32 \pm 0.08, \tag{4}$$

$$a_1 \approx (1.6637 \pm 0.0002) \times 10^{-4}, \tag{5}$$

$$x_0' \approx (4.3 \pm 0.3) \times 10^{-4}, \tag{6}$$

$$a_2 \approx (1.6701 \pm 0.0002) \times 10^{-4}, \tag{7}$$

$$y_0' \approx (-1.941 \pm 0.003) \times 10^{-2}. \tag{8}$$

c) We can estimate the noise in the data by calculating the sample standard deviation in the residuals. Carrying this out, we see that the noise is approximately $N = 3.60$mm. Firstly, to get an accurate value of a, we take the average of a_1 and a_2 to give $a = 1.66692 \times 10^{-4}\text{mm}^{-1}$. To find an accurate uncertainty for a, we proceed using the fact that $Cov = (A^T N A)^{-1}$ whereby in our case N is just a number that we can multiply by. Hence, if our previous covariance matrix is Cov', $Cov = \frac{Cov'}{N}$. Carrying this out, we find the uncertainties in $a1$ and $a2$. Combining the uncertainties as $\alpha_a = \frac{1}{2}\sqrt{\alpha_{a_1}^2 + \alpha_{a_2}^2}$, we arrive at $a = (1.66692 \pm 0.00008) \times 10^{-4}\text{mm}^{-1}$.

Furthermore, we can calculate the focal point, f, by realising that a paraboloid is similar to a parabola in two directions. In other words, $z = ax^2$ and $z = by^2$ (ignoring translations as they do not impact focal length). Using our mean value of a, we can therefore calculate f by rearranging to get $f = \frac{1}{4a}$. Furthermore, to find the error we can do a first order Taylor expansion taking f to be $f \pm df$ and a to be $a \pm da$ where df and da represent the respective errors.

$$f \pm df = \frac{1}{4(a \pm da)} = \frac{1}{4a(1 \pm (da/a))}, \tag{9}$$

$$\therefore \pm df \approx \frac{1}{4a}(1 \mp \frac{da}{a}) - f = \frac{1}{4a}(1 \mp \frac{da}{a}) - \frac{1}{4a} \approx \mp\frac{da}{4a^2}. \tag{10}$$

Calculating f and df in this way, we get a focal length of $f \approx (1,499.78 \pm 0.07)$mm. This is quite close to the actual value of 1.5m (although it is over 4σ away)! All the outputs for this question can be seen below.

```
[Running] python -u "/Users/jehandastoor/Phys512/assignment3/problem_1.py"
c0= -1512.321964408296 a1= 0.0001663725551801555 x0= 0.0004333158423712683 a2= 0.000167010731813347 y0= -0.019406547149939887
errs= [8.28010443e-02 2.18705404e-08 3.31977483e-05 2.12328150e-08
 3.16457966e-05]
Noise= 3.6021306047038903
a= 0.00016669164349675125 ± 8.030339124980001e-09
Focal= 1499.7752422116612 ± 0.07225139517232841

[Done] exited with code=0 in 1.801 seconds
```

Figure 2: Python output of the "problem_1.py" file for the linear least squares fit and following parts.

2. We begin by first updating the "wmap_camb_example.py" so that we can enter any array of l and it will work. We now calculate lmax and convert the result into an integer inside the function. This is changed on line 17. Lastly, we receive an error that the result is not broadcast correctly. This can be fixed within the function on line 20 so that the results are calculated between the minimum and maximum value of l. Running the code:

```
[Running] python -u "/Users/jehandastoor/Phys512/assignment3/problem_2.py"
chisq = 1588.4257873990618

[Done] exited with code=0 in 4.158 seconds
```

Figure 3: Python output of the χ^2 value calculated using our "problem_2.py" file.

This is similar to the value we expected and therefore, we conclude that $\chi^2 = 1588.42578....$

3. The Newton's method fitting script was written and uploaded to GitHub under "newton.py". We use the regular Newton procedure,

$$\delta_m = (A_m^T N^{-1} A_m)^{-1} A_m^T N^{-1} r, \tag{11}$$

$$\therefore m' = m_0 + \delta_m, \tag{12}$$

whereby, the usual definitions of parameters apply. The derivative matrix with respect to the parameters, A_m, was determined by taking the derivative at each l-point with parameter values $m + dx$ and $m - dx$. In this way we can calculate the derivative using:

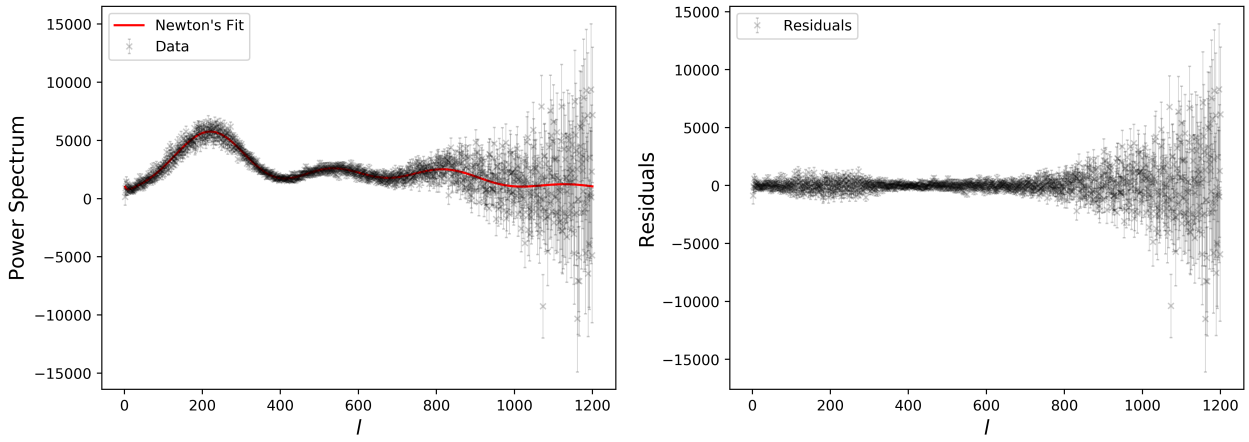
$$f'(m) = \frac{f(m + dx) - f(m - dx)}{2dx}. \tag{13}$$

We arbitrarily take dx to be $0.001m$ for each parameter. Furthermore, we repeatedly carry out Newton's method until the difference between our current χ^2 and our previous χ^2 is less than χ^2_{min} which is chosen arbitrarily to be 0.001. The errors in each parameter are calculated using the Covariance matrix, i.e. the first half of the equation for δ_m or $Cov = (A_m^T N^{-1} A_m)^{-1}$. Running our program, the resulting output is shown below.

```
[Running] python -u "/Users/jehandastoor/Phys512/assignment3/problem_3.py"
Chi 1:
        Parameter 1
        Parameter 2
        Parameter 3
        Parameter 5
        Parameter 6
        chi1= 1235.029822260181

Chi 2:
        Parameter 1
        Parameter 2
        Parameter 3
        Parameter 5
        Parameter 6
        chi2= 1227.9220289228338

Chi 3:
        Parameter 1
        Parameter 2
        Parameter 3
        Parameter 5
        Parameter 6
        chi3= 1227.920825084604

Chi 4:
        Parameter 1
        Parameter 2
        Parameter 3
        Parameter 5
        Parameter 6
        chi4= 1227.9208432017945

Parameters = [69.32410840430558, 0.022489338363623535, 0.11390875508037375, 0.05, 2.0423192424219546e-09, 0.9696872252684703]
Errors = [2.39514518e+00 5.39416211e-04 5.21873610e-03 3.89168919e-11
 1.35766205e-02]
Chi = 1227.920825084604

[Done] exited with code=0 in 127.406 seconds
```

Figure 4: Python output for the different iterations of Newton's method (represented as Chi 1, Chi 2, etc.) showing the calculation of derivatives for each parameter and the resulting χ^2 value for that fit.

We see that the χ^2 value for Newton's method (1227.9208...) is significantly less than that of the model script. Furthermore, our script correctly uses the fit for Chi 3 as we see that the χ^2 value for Chi 3 is actually less than that of Chi 4. Furthermore, the "Errors" printout only gives the errors for 5 of the parameters as τ is not calculated in this fit. To check whether our estimates of the derivatives are accurate, we graph the results and qualitatively judge our fit. The plot of the fit and the residuals can be seen below:



(a) Plot of the Chi 3 fit.

(b) Chi 3 residuals.

Figure 5: Plot of the Newton's method fit using the parameters given by the 3rd iteration of Newton's method (taking $\tau = 0.05$). The parameters and their errors can be seen in Figure 4.

From Figure 5, we see that the fit is quite accurate and from the output shown in Figure 4, the χ^2 value is significantly reduced from previously. This is some evidence that our derivatives are accurate, however more evidence will be given later. We summarise the resulting parameters below (where $\tau = 0.05$):

$$H_0 \approx (69 \pm 2)\text{kms}^{-1}, \tag{14}$$

$$\omega_b h^2 \approx (0.0225 \pm 0.0005), \tag{15}$$

$$\omega_c h^2 \approx (0.114 \pm 0.005), \tag{16}$$

$$\tau \approx (0.05), \tag{17}$$

$$A_s \approx (2.04 \pm 0.04) \times 10^{-9}, \tag{18}$$

$$Slope \approx (0.97 \pm 0.01). \tag{19}$$

Now, we proceed with the same script and procedure, this time allowing for τ to be changed. We expect the errors to be much larger as there are more variables changing. The output of the script has been shown below.

```
[Running] python -u "/Users/jehandastoor/Phys512/assignment3/problem_3.py"
Chi 1:
        Parameter 1
        Parameter 2
        Parameter 3
        Parameter 4
        Parameter 5
        Parameter 6
        chi1= 1235.744800980528

Chi 2:
        Parameter 1
        Parameter 2
        Parameter 3
        Parameter 4
        Parameter 5
        Parameter 6
        chi2= 1227.9700076208383

Chi 3:
        Parameter 1
        Parameter 2
        Parameter 3
        Parameter 4
        Parameter 5
        Parameter 6
        chi3= 1282.569909869489

Parameters = [68.81683766702088, 0.02238040318520416, 0.11478924969527218, 0.020706590238563652, 1.9270977297701473e-09, 0.9659081318793571]
Errors = [3.27129099e+00 7.67560555e-04 6.61375725e-03 1.50860358e-01
 5.99141506e-10 2.29600711e-02]
Chi = 1227.9700076208383

[Done] exited with code=0 in 120.994 seconds
```

Figure 6: Python output for the different iterations of Newton's method (represented as Chi 1, Chi 2, etc.) when τ is not fixed and the resulting χ^2 value for that fit.

We see that the χ^2 value is slightly larger than previously at 1227.97... compared to 1227.92... and that each variable seems to have larger errors as expected. Furthermore, fewer iterations are needed before the χ^2 value ends up actually increasing (2 instead of 3). The new parameters have been summarised below:

$$H_0 \approx (69 \pm 3)\text{kms}^{-1}, \tag{20}$$

$$\omega_b h^2 \approx (0.0224 \pm 0.0008), \tag{21}$$

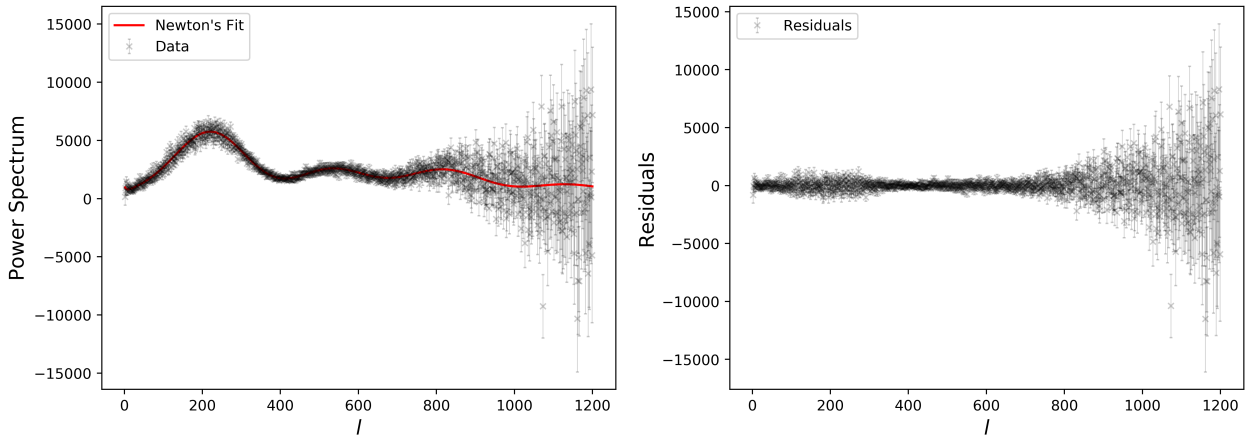
$$\omega_c h^2 \approx (0.115 \pm 0.006), \tag{22}$$

$$\tau \approx (0.0 \pm 0.1), \tag{23}$$

$$A_s \approx (1.9 \pm 0.6) \times 10^{-9}, \tag{24}$$

$$Slope \approx (0.97 \pm 0.02). \tag{25}$$

In addition, the plots of the fit for when τ is floating have been shown below.



(a) Plot of the Chi 2 fit.

(b) Chi 2 residuals.

Figure 7: Plot of the Newton's method fit using the parameters given by the 3rd iteration of Newton's method (when τ is floating). The parameters and their errors can be seen in Figure 6.

We also plot the absolute value of the derivatives for the first chi iteration below. Obviously, the derivatives for A_s will be extraordinarily large as dx is much smaller for it, hence we multiply each derivative by dx to allow for easier comparison.

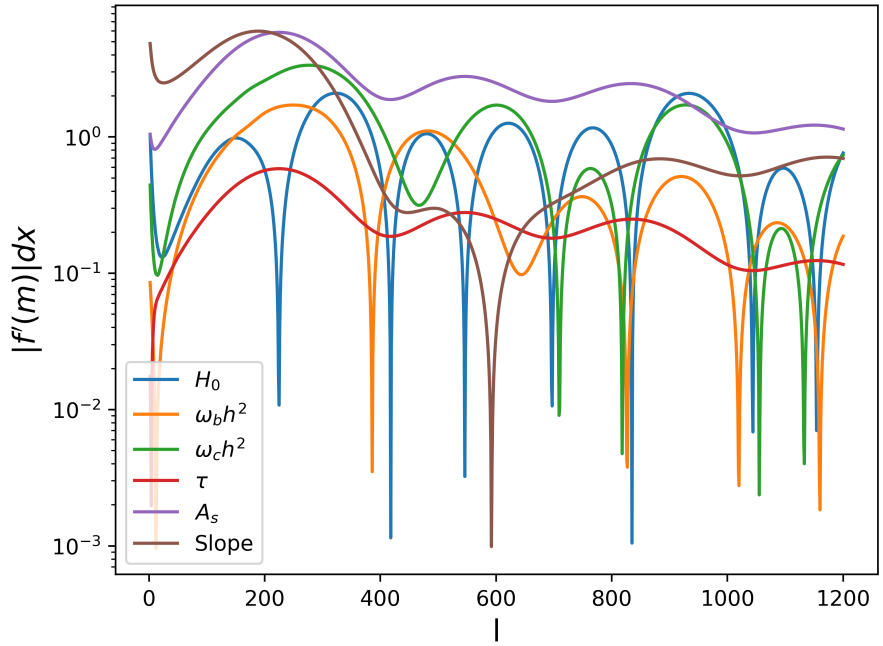


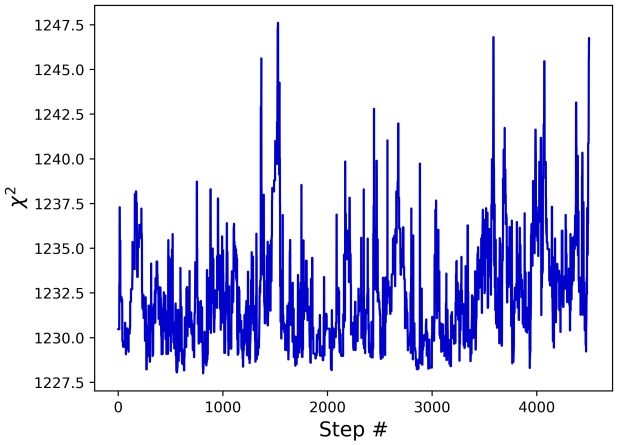
Figure 8: Plot of the derivatives for the first chi iterations where τ has been floated.

Our derivatives seem to all be continuous and on similar orders of magnitude and therefore our value of dx is accurate. If they were not on the same order of magnitude, this would imply that the rate of change depends largely on some parameters which does not make sense, i.e. our value is too large. Hence, we take this as proof that our derivatives are accurate and our value of dx is acceptable.

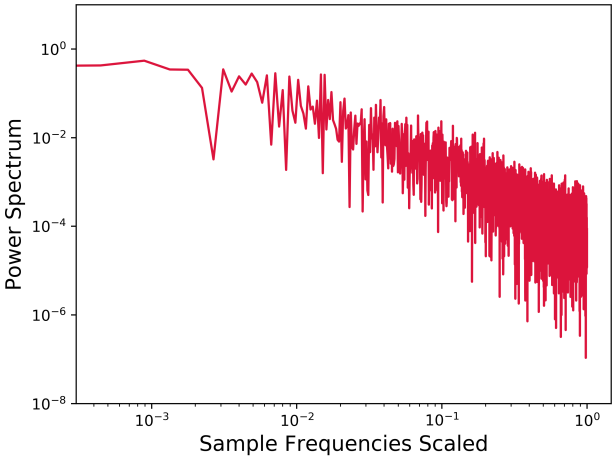
Furthermore, both fits to the data look quite similar, however the main discrepancy lies in the difference of the errors for each parameter. One can see this by comparing Figure 4 to Figure 6. The difference can especially be seen for A_s which, for the floating fit, has an uncertainty one order of magnitude larger than the previous uncertainty.

4. Firstly, in order to save time when running our Markov-chain Monte Carlo program, we save the results of the Newton fit so that we can just load them each time rather than recalculating them. For our first chain, we use the covariance given by Newton's method and the parameter values given by it. Testing the "problem_4.py" code (which calls our Markov-chain Monte Carlo program titled "markov.py"), without adjusting our step size, for 50 steps our acceptance rate (number of trials that succeed vs. trials that fail) remains around 2-6%. Hence, we arbitrarily adjust the step size by a constant factor until we achieve a higher success rate. We do not want all trials to succeed as then it will take too long for the CMB parameters to converge. A success rate of approximately 25% sounds like a good start.

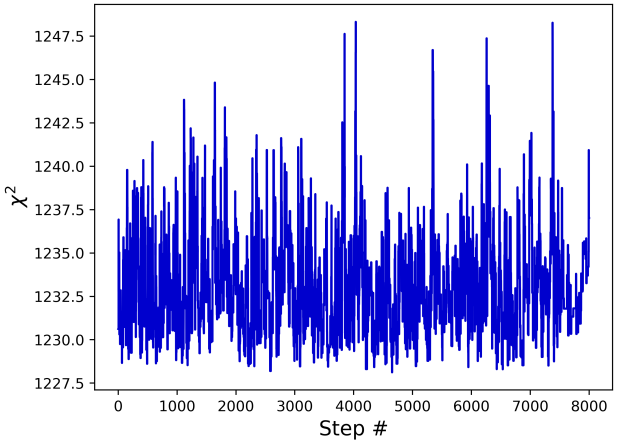
We repeat this process for 3 different chains, each time we start a new chain, we take the covariance of the parameters of the previous chain using np.cov and use this for our next Cholesky decomposition to find the step sizes (as we now our data is correlated). Furthermore, we take the mean of the parameter values and use this as our starting parameter values (making sure to cut off our burn in). Then again we scale the step size arbitrarily till we achieve the acceptance rate we desire. The number of steps per chain was 5000, 12000 and 5000, respectively (including burn-in) and their acceptance rates were 34.8%, 25.0% and 18.9%. Rather than plotting each variable, we will just plot χ^2 for each chain to determine whether our chains have converged. Furthermore, we will plot the power spectrum using scipy.signal.periodogram and scale the x-axis so that it is maximum 1, and this way we can read off the number of independent trials almost directly. For simplicity, we assume the number of independent trials determined by χ^2 matches that of each parameter.



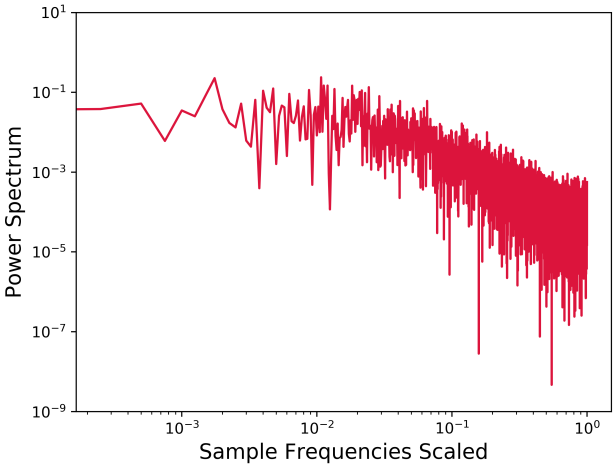
(a) Plot of χ^2 for chain 1 after removing burn-in.



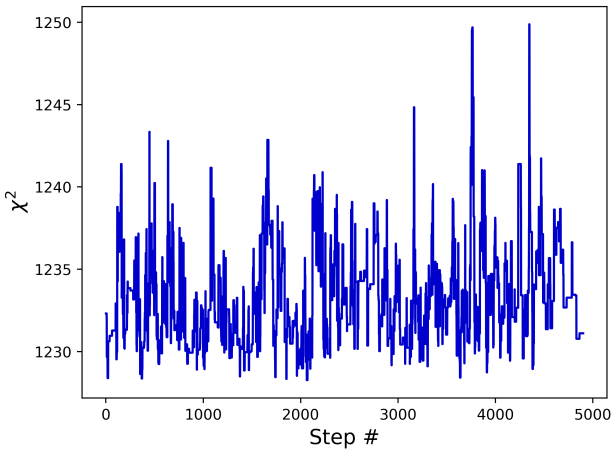
(b) Fourier transform for chain 1 without burn-in.



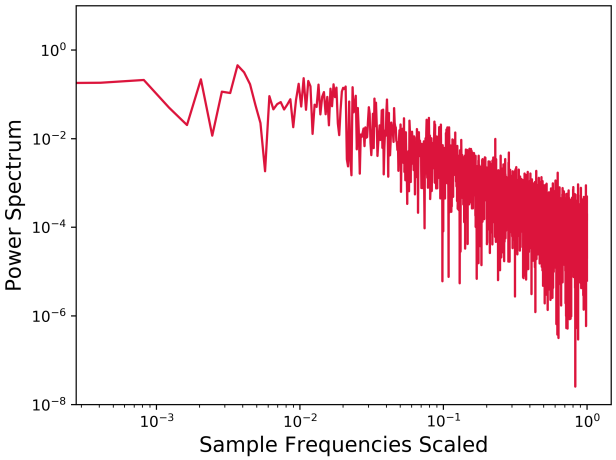
(c) Plot of χ^2 for chain 2 after removing burn-in.



(d) Fourier transform for chain 2 without burn-in.



(e) Plot of χ^2 for chain 3 after removing burn-in.



(f) Fourier transform for chain 3 without burn-in.

Figure 9: Plot of the χ^2 values from a Markov chain Monte Carlo fit for 3 different chains that build off of the information given by the last chain.

Interestingly, we see that our parameters seem to be correlated as our power spectrum's have more noise on a large scale. We observe convergence when our power spectrum flattens off over a large scale on the left. Looking at each respective chain, chain 1 does not seem to be well converged, however chain 2 and 3 appear to be converged as they are flat over a large scale, although more steps could probably help. However, due to time constraints we proceed with these two chains assuming convergence.

We can estimate the number of independent samples from the power spectrum where the kink begins. This seems to be at approximately 0.025 and 0.021, respectively, i.e. there are $8000 \times 0.025 = 200$ and $4900 \times 0.021 \approx 103$ independent samples, respectively (the number of samples is not the same as the number of steps used as we have removed burn-in). We can now calculate our parameter values by taking the mean value of our parameter over the chain length (excluding burn-in) and their uncertainties by calculating the standard deviation in each parameter and dividing by the square root of the number of independent samples. Furthermore, we can calculate the parameter values for both of these chains and then average the results to get an even more accurate estimate, and sum the uncertainties in the same way described in problem 1. In this way we achieve our parameters and their limits (described by their uncertainties). The results are:

$$H_0 \approx (72.7 \pm 0.3)\text{kms}^{-1}, \tag{26}$$

$$\omega_b h^2 \approx (0.02329 \pm 0.00007), \tag{27}$$

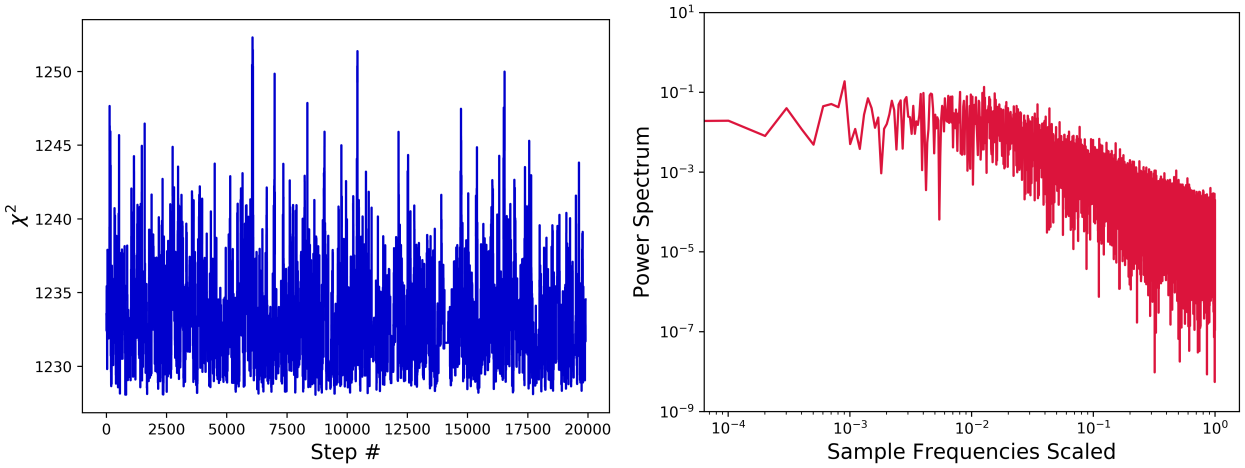
$$\omega_c h^2 \approx (0.1084 \pm 0.0005), \tag{28}$$

$$\tau \approx (0.144 \pm 0.005), \tag{29}$$

$$A_s \approx (2.47 \pm 0.03) \times 10^{-9}, \tag{30}$$

$$Slope \approx (0.999 \pm 0.002). \tag{31}$$

5. We begin by updating our previous "markov.py" code to account for our new interval for τ. We do so by assuming that the uncertainty given by the Planck satellite is the 1σ deviation (as is convention), and hence we have a normal distribution centered about 0.0544. Randomly taking samples from this distribution, we use this as our new value for τ each time. Hence, the value of τ does not stray too far from the origin and we can check this by taking the average of τ over the chain length. The details of the code are explained in the python file. We run a chain of 20,000 steps using the final parameters given by Newton's method where τ is not floated and also use the covariance matrix given by this fit to take correlated steps for the other parameters (not τ). We then calculate the power spectrum, as done previously, to determine whether we have reached convergence, and the number of independent samples we have. The plots are shown below.



(a) Plot of χ^2 for the chain after removing burn-in.

(b) Fourier transform for the chain without burn-in.

Figure 10: Plot of the χ^2 values from a Markov chain Monte Carlo fit for a single chain with 20,000 steps where we know are sampling the value of τ about a normal distribution centered at 0.0544 with $\sigma = 0.0073$.

The overall chain has an acceptance rate of 29.68% which is around the amount we want. We see that we have reached convergence due to the fact that the plot of χ^2 looks similar to white noise, and the fact that the plot of the power spectrum is flat over a large scale and then bends down. We also see that our parameters are still correlated due to there being more noise at larger scales. The kink begins at approximately 0.016, therefore there are approximately 318 independent trials (0.016×19900 as we are accounting for burn-in). Hence, we can calculate the mean of our parameters over the chain and now the errors in the mean by taking the standard deviation of each parameter over the chain and dividing by the square root of the number of independent samples. We also calculate the value of τ from the chain as τ is still allowed to vary. The results are:

$$H_0 \approx (69.5 \pm 0.1)\text{kms}^{-1}, \tag{32}$$

$$\omega_b h^2 \approx (0.02255 \pm 0.00003), \tag{33}$$

$$\omega_c h^2 \approx (0.1139 \pm 0.0003), \tag{34}$$

$$\tau \approx (0.0536 \pm 0.0002), \tag{35}$$

$$A_s \approx (2.058 \pm 0.002) \times 10^{-9}, \tag{36}$$

$$Slope \approx (0.9714 \pm 0.0008). \tag{37}$$

Comparing these parameters to that of the previous Markov chain and Newton's method, we see that these parameters more closely resemble that of Newton's method, but with a greater precision. Furthermore, we note that the value of τ has not varied greatly from that of 0.0544, however it now also has a greater precision. We can also compare these results to that of the previous chain by determining the χ^2 value for each and comparing. We see that χ^2 for the previous chain is approximately $1,241.38$, while χ^2 for the new chain is approximately $1,227.97$. We see that the new chain has a χ^2 around that of Newton's method, while the old chain has a χ^2 that is larger. Hence, we can conclude that our new chain is more accurate.