1 Question 1

a)

In the test script, the parameter values were as follows: [60, 0.02, 0.1, 0.05, 2.00e-9, 1.0]. From this, a χ^2 value of 15267 was calculated. To determine if this is a good fit, we can look at the χ^2 distribution for large n (in this case there are 2507 data points). At large n, the χ^2 distribution approximates a normal distribution with mean k and variance $2k$ where k is the number of degrees of freedom ($k = 2507 - 6 = 2501$). Therefore a 95% confidence interval for the χ^2 value can be constructed by adding 2 standard deviations to the mean:

$$95\% \text{ confidence interval} = [k - 2\sqrt{2k}, k + 2\sqrt{2k}] = [22360, 2642]$$

In this case, a value of 15267 is very many standard deviations away from the mean and is indicative of a bad fit.

Now using the parameter values: [69, 0.022, 0.12,0.06, 2.1e-9, 0.95], a χ^2 value of 3272.21 was calculated. This is much closer, however it still corresponds to roughly $+11\sigma$ away from the mean. Therefore this is much better than the previous parameters, however it is not yet an acceptable fit. The χ^2 has to be brought down to something that is statistically significant for the fit to be acceptable.

b)

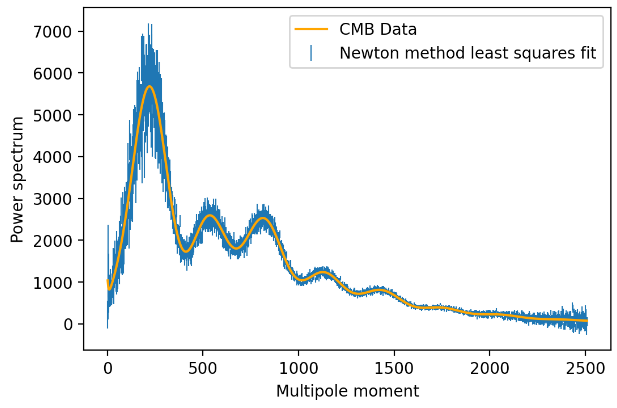
For part b), Newton's method was used to fit the model, using an initial guess of the second set of parameters from part a). In order to run Newton's method, code from Assignment 4 was adapted. First, the parameter gradient function was adapted to use the function `get_spectrum` from the test script:

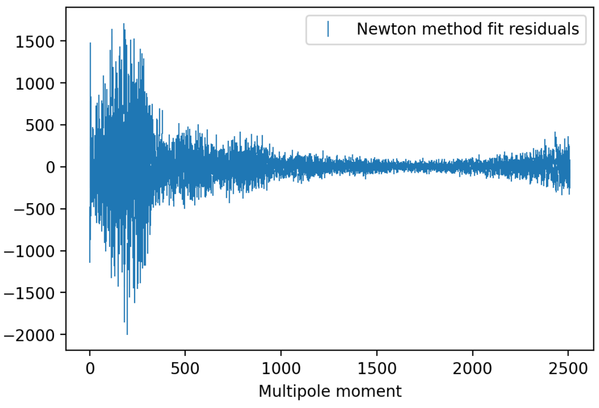
```python
def CMBmodel_param_grad(m,y_data):  # Get parameter gradient of model numerically
    n_data=len(y_data)
    n_m=len(m)
    A=np.empty([n_data,n_m])
    for i in range(n_m):
        dm_i=m[i]/10**8  # Params have orders of magnitudes differences, set dx proportional
        m_c=m.copy()
        m_c[i]=m[i]+dm_i
        y2=get_spectrum(m_c)[:n_data]
        m_c[i]=m[i]-dm_i
        y1=get_spectrum(m_c)[:n_data]
        A[:,i]=(y2-y1)/(2*dm_i)  # Central difference
    return A
```

With this gradient, Newton's method is now able to be computed. Below is code for Newton's minimization method, which also computes the error of the parameters from the covariance matrix:

```python
def CMBmodel_newton_min(m0,y_data,errs,n):
    m=m0.copy()
    ndata=len(y_data)
    N=np.diag(errs**2)  # Errors no longer assumed constant, must include N
    N_inv=np.linalg.inv(N)
    for i in tqdm(range(n)):
        pred=get_spectrum(m)[:ndata]
        grad=CMBmodel_param_grad(m,y_data)[:ndata]
        resid=y_data-pred
        lhs=grad.T@N_inv@grad
        rhs=grad.T@N_inv@resid
        cov=np.linalg.inv(lhs)
        dm=cov@rhs
        m=m+dm
    m_err=np.sqrt(np.diag(cov))
    return m,m_err,cov  # Return covariance matrix for MCMC
```

Here are the plots for the fit and its residuals:





Here are the computed parameters, their corresponding errors and the χ^2 for the fit:

$$H_0 = 6.691641056739895532e+01 \pm 7.994041237217667240e-01$$

$$\Omega_b h^2 = 2.203310594192030986e-02 \pm 3.828039229055306853e-05$$

$$\Omega_c h^2 = 1.202718845129031877e-01 \pm 2.093914075277517398e-03$$

$$\tau = 6.507785511749636376e-02 \pm 3.551722490712985042e-02$$

$$A_s = 2.139696810788384209e-09 \pm 1.435509543793212017e-10$$

$$n_s = 9.651411296248182392e-01 \pm 6.973237142678239200e-03$$

$$\chi^2 = 2584.4382166398823$$

c)

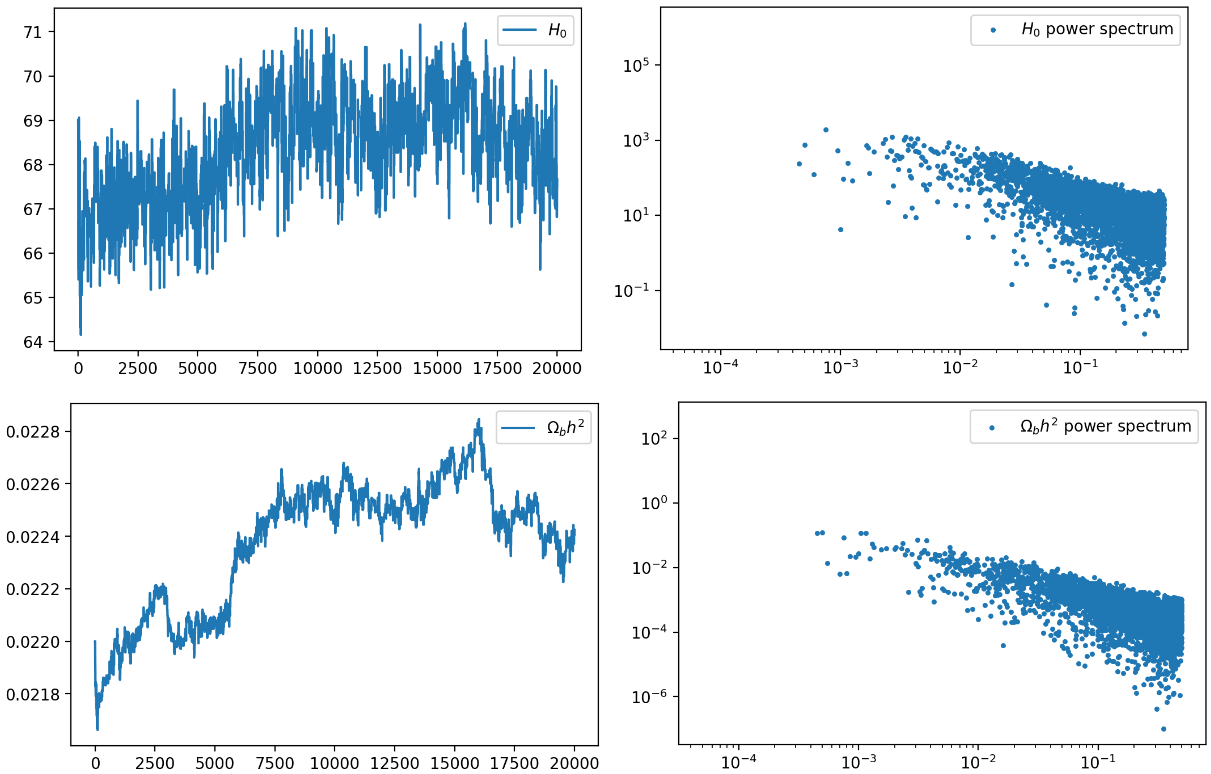
For the MCMC fit, the important part to figure out is the trial step. In this case, the covariance matrix from the last iteration of the Newton method fit was used, then was factored using Cholesky decomposition and multiplied by a random vector whose components are random between 0 and 1. This was the same code used in class:

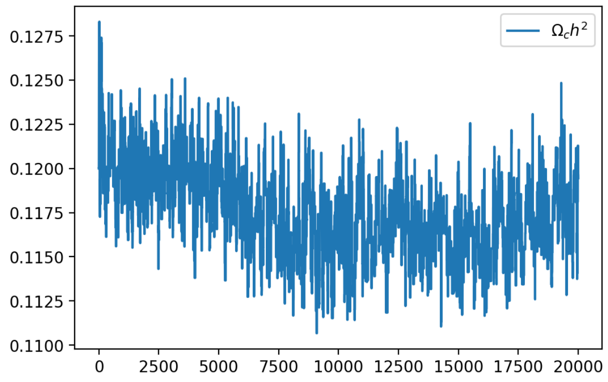
```
def get_step(trial_step):
    if len(trial_step.shape)==1:
        return np.random.randn(len(trial_step))*trial_step
    else:
        L=np.linalg.cholesky(trial_step)
        return L@np.random.randn(trial_step.shape[0])
```

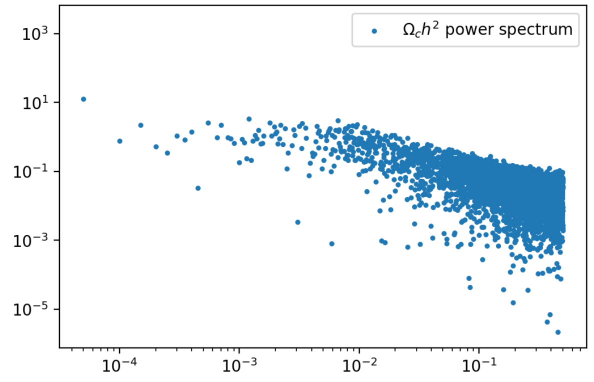
Next is the code for the MCMC chain (next page):

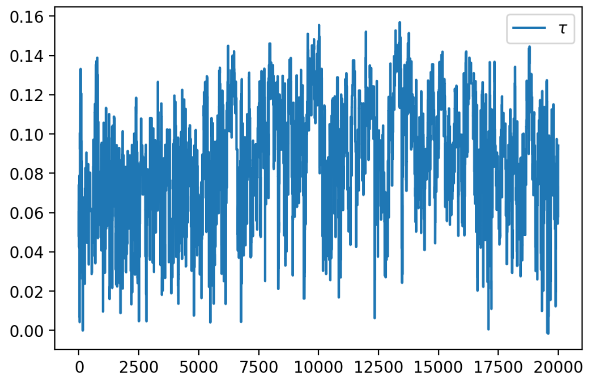
```
def MCMC_chain(y_data,errs,m0,trial_step,nstep=10000,T=1):
    nparam=len(m0)
    chain=np.zeros([nstep,nparam])
    chisq=np.zeros(nstep)
    chain[0,:]=m0
    cur_chisq,resid=model_chisq(m0,errs,y_data)
    chisq[0]=cur_chisq
    m=m0
    for i in tqdm(range(1,nstep)):
        dm=get_step(trial_step)
        trial_m=np.matrix(m+dm)
        new_chisq,resid=model_chisq(trial_m.T,errs,y_data)
        accept_prob=np.exp(-0.5*(new_chisq-cur_chisq)/T)
        if np.random.rand(1)<accept_prob:
            m=trial_m
            cur_chisq,resid=(new_chisq,resid)
        chain[i,:]=np.ravel(m)
        chisq[i]=cur_chisq
    return chain,chisq
```

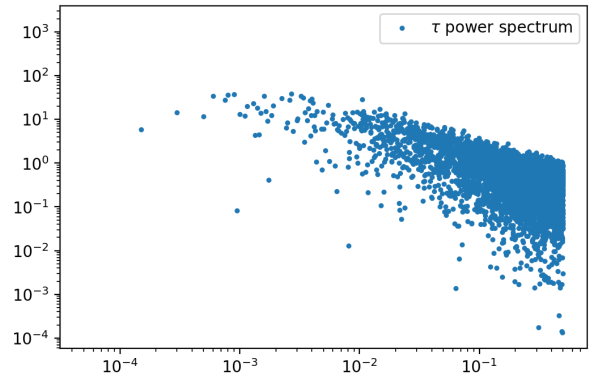
Computing the errors as the average of the 1σ upper and lower bounds, the chain was computed over 20000 steps. The starting point used was the same starting point as Newton's method. Below are the plots of the parameters in the chain and their associated Fourier transformed power spectrum:

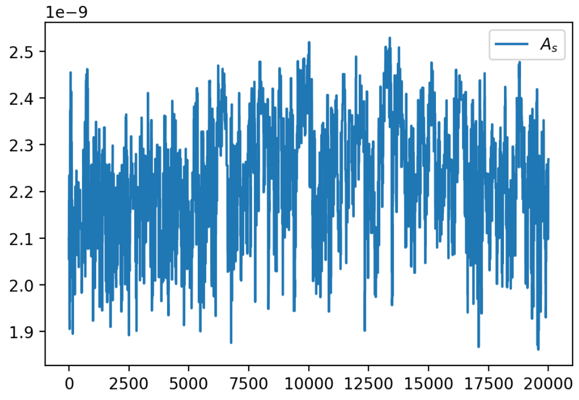


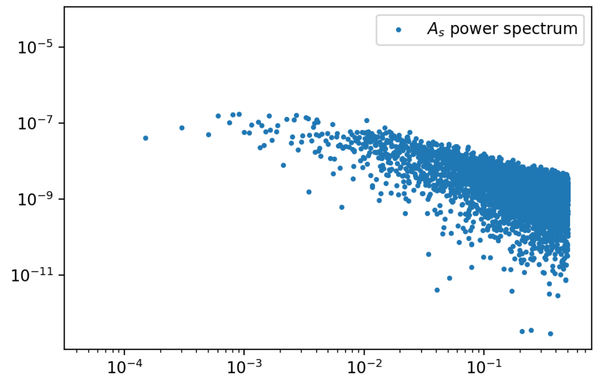

$\Omega_c h^2$

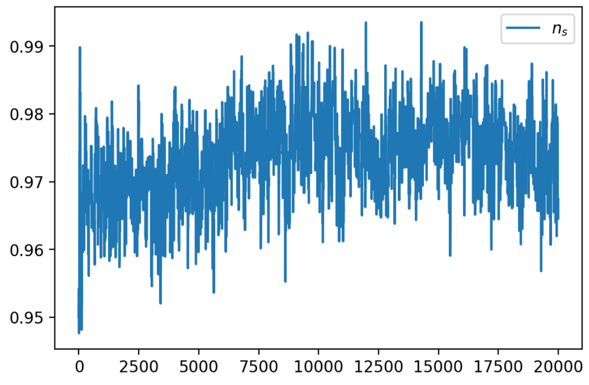

$\Omega_c h^2$ power spectrum

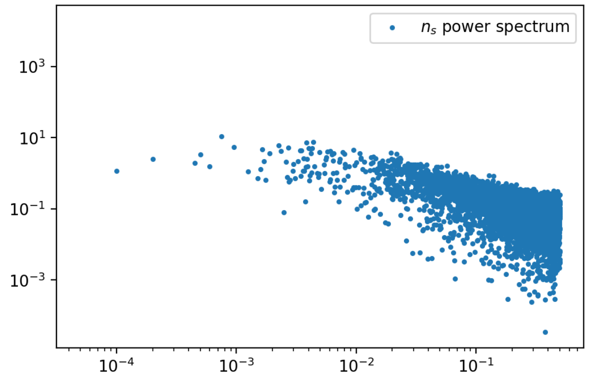

τ


τ power spectrum


1e−9
A_s

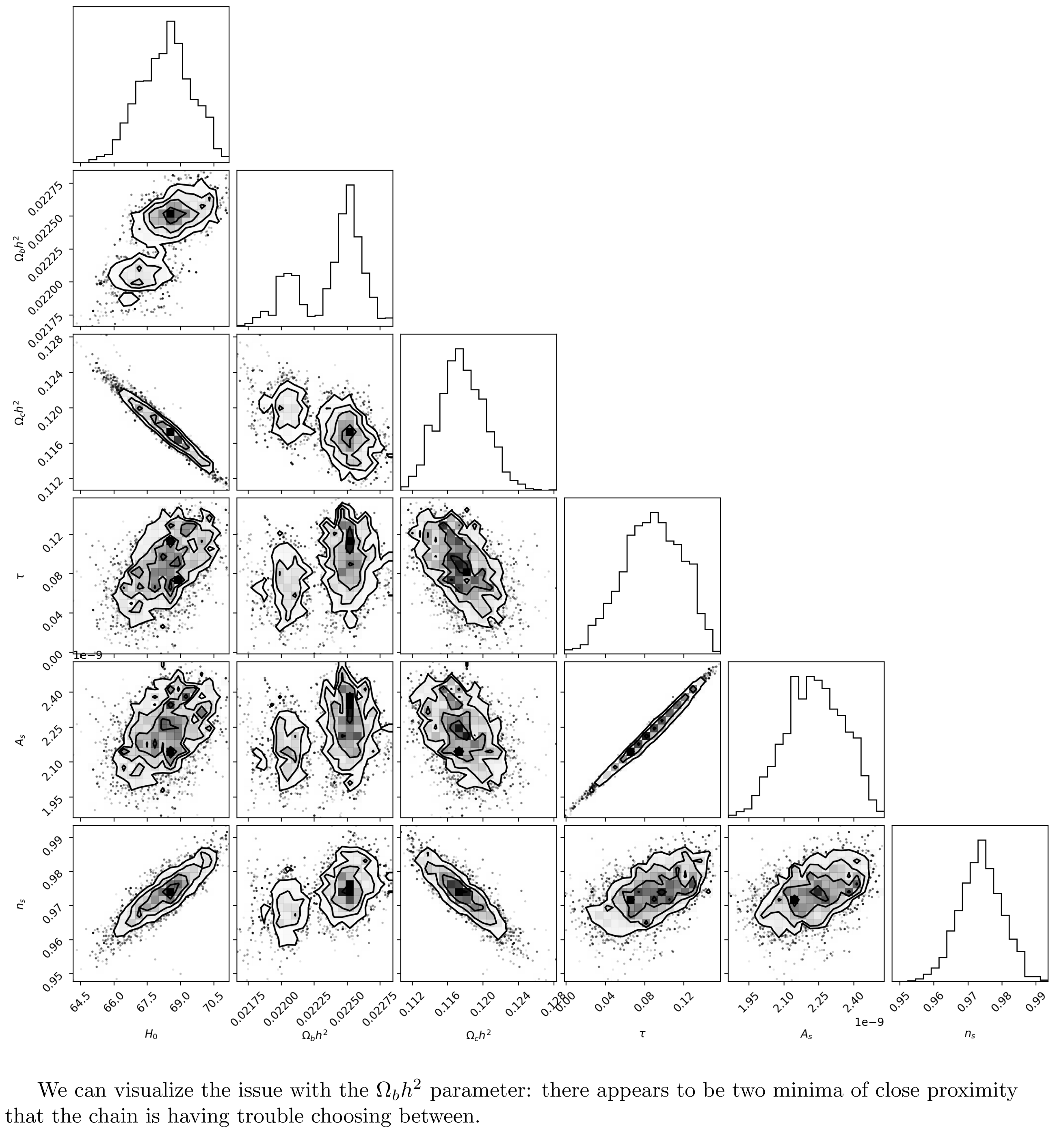

A_s power spectrum


n_s


n_s power spectrum

As can be seen in the power spectrum of each parameter, there is a slight bend and levelling off of the lower frequencies, besides the baryon density parameter $\Omega_b h^2$. Therefore all the parameters besides this one show some degree of convergence.

Using a Python module named `corner.py`, the covariances of each parameter can be plotted:



We can visualize the issue with the $\Omega_b h^2$ parameter: there appears to be two minima of close proximity that the chain is having trouble choosing between.

To find the values for $\Omega_b h^2$ and $\Omega_c h^2$, the mean and standard deviation of their chains were taken. This gave the values:

$$H_0 = 6.832306979302595096e+01 \pm 1.160482230718751451e+00$$
$$\Omega_b h^2 = 2.237641191021686946e-02 \pm 2.435043884764399728e-04$$
$$\Omega_c h^2 = 1.175631073056536968e-01 \pm 2.578813538508824017e-03$$
$$\tau = 8.725414744841814008e-02 \pm 2.979736648981657493e-02$$
$$A_s = 2.230156561272920595e-09 \pm 1.247155332669573012e-10$$
$$n_s = 9.736675070016088673e-01 \pm 6.310940039157754422e-03$$
$$\chi^2 = 2577.46786559916$$

From these values, the baryon density, dark matter density and dark energy density parameters Ω_b, Ω_c and Ω_Λ can be calculated from the parameters:

$$\Omega_b = \frac{100^2}{H_0^2}\Omega_b h^2\,,\ \Omega_c = \frac{100^2}{H_0^2}\Omega_c h^2$$
$$\Omega_\Lambda = 1 - \Omega_b - \Omega_c$$

Calculating these values from the fit parameters and propagating their errors, the values obtained are:

$$\Omega_b = 0.0479353282443152 \pm 0.00044049338488250756$$
$$\Omega_c = 0.2518467286323539 \pm 0.0011481207059277339$$
$$\Omega_\Lambda = 0.7002179431233309 \pm 0.0012297217480004357$$

d)

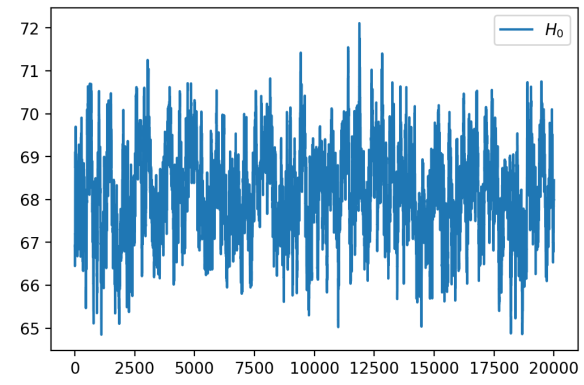
In order to calculate new parameters from the chain using the constraint for the new value of τ, the chain was importance sampled. Using a weight function that was a Gaussian with mean and standard deviation given by τ and its error, a weighted average was performed on each of the chain variables. Here is the code for this (adapted from class code):

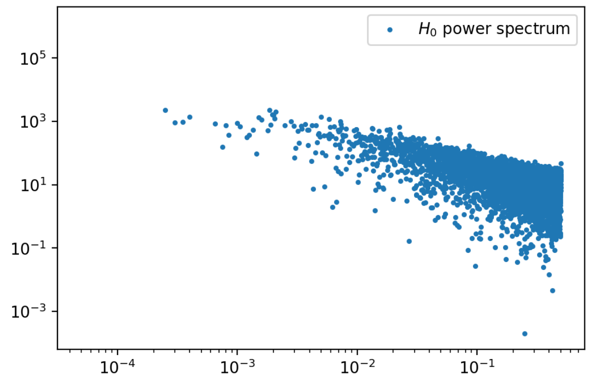
```
def importance_sample(chain,wt):
    npar=len(chain[0,:])
    tot=np.zeros(npar)
    totsqr=np.zeros(npar)
    for i in range(npar):
        tot[i]=np.sum(wt*chain[:,i])
        totsqr[i]=np.sum(wt*chain[:,i]**2)
    mean=tot/np.sum(wt)
    meansqr=totsqr/np.sum(wt)
    var=meansqr-mean**2
    return mean,np.sqrt(var)
```

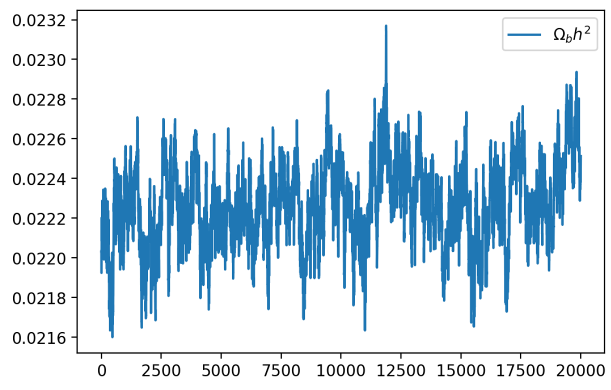
The resulting parameters from this importance sampling are:

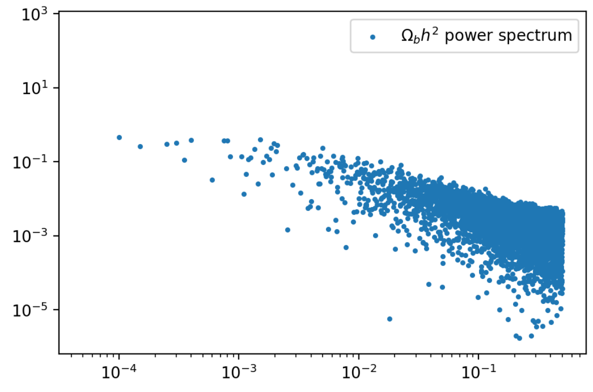
$$H_0 = 6.761291334694415411e+01 \pm 1.045054060035916610e+00$$
$$\Omega_b h^2 = 2.227179933977105342e-02 \pm 2.684773911646924183e-04$$
$$\Omega_c h^2 = 1.190990622298652851e-01 \pm 2.252784472509883514e-03$$
$$\tau = 5.586628496098173385e-02 \pm 7.476387498915549636e-03$$
$$A_s = 2.099975974929071309e-09 \pm 3.200842004626447686e-11$$
$$n_s = 9.698367736185231625e-01 \pm 5.462034102804874149e-03$$
$$\chi^2 = 2577.086764399311$$

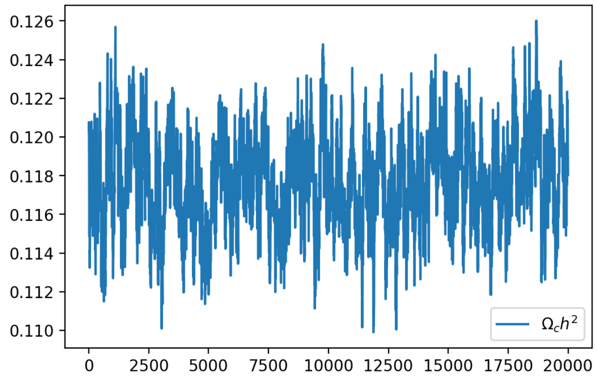
To include the new constraint on the parameter τ for a new MCMC chain, the covariance matrix used to generate steps from was recalculated using these new parameters from the importance sampling. The chain was run with 20000 steps and began from the results of the Newton fit. Here are the plots of the parameter chains and their associated power spectrum:

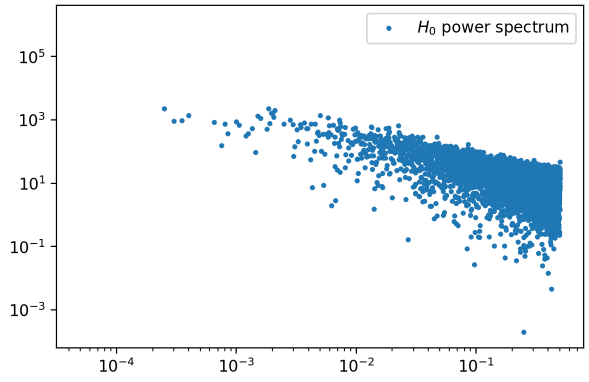


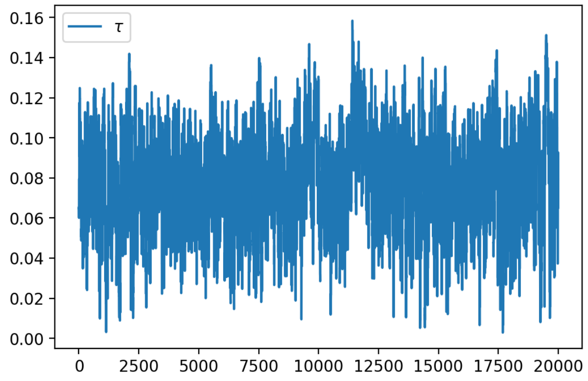


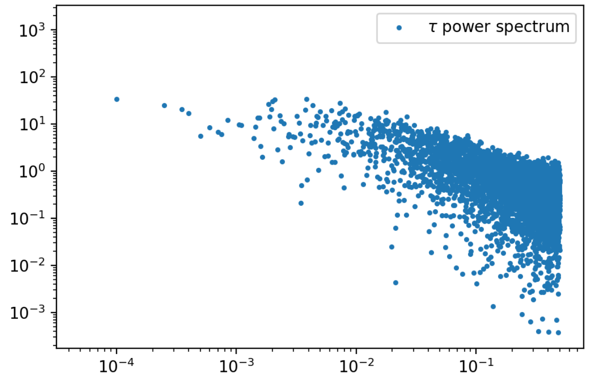


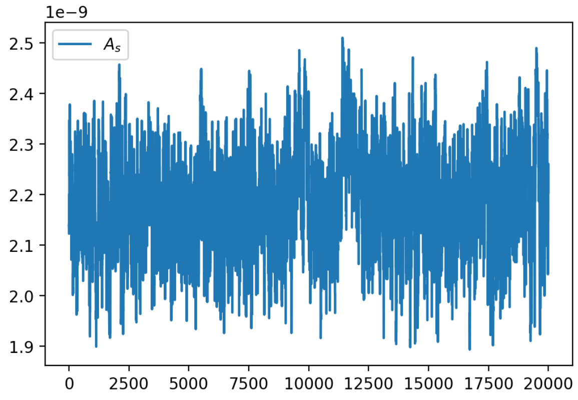


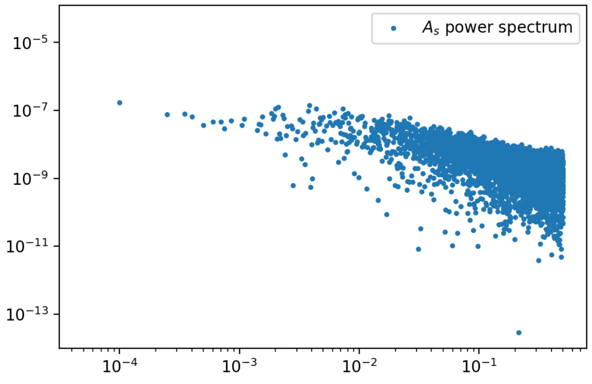

$\Omega_c h^2$

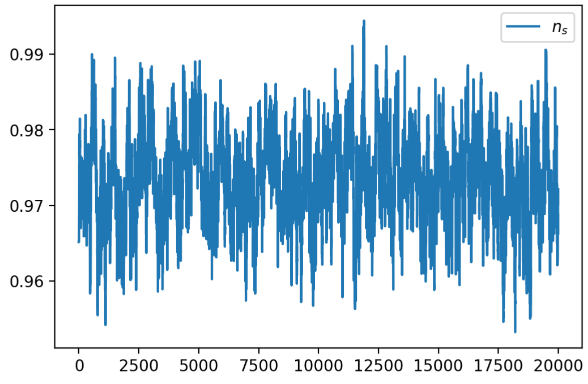

H_0 power spectrum

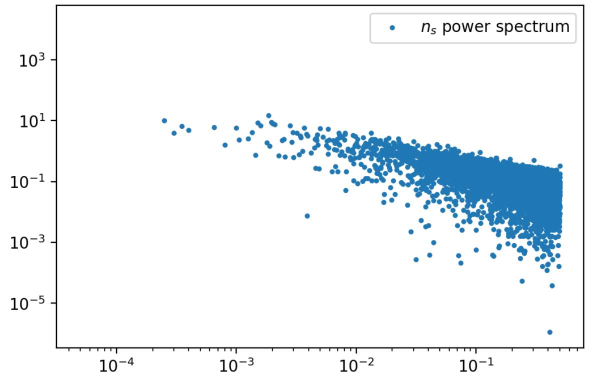

τ


τ power spectrum

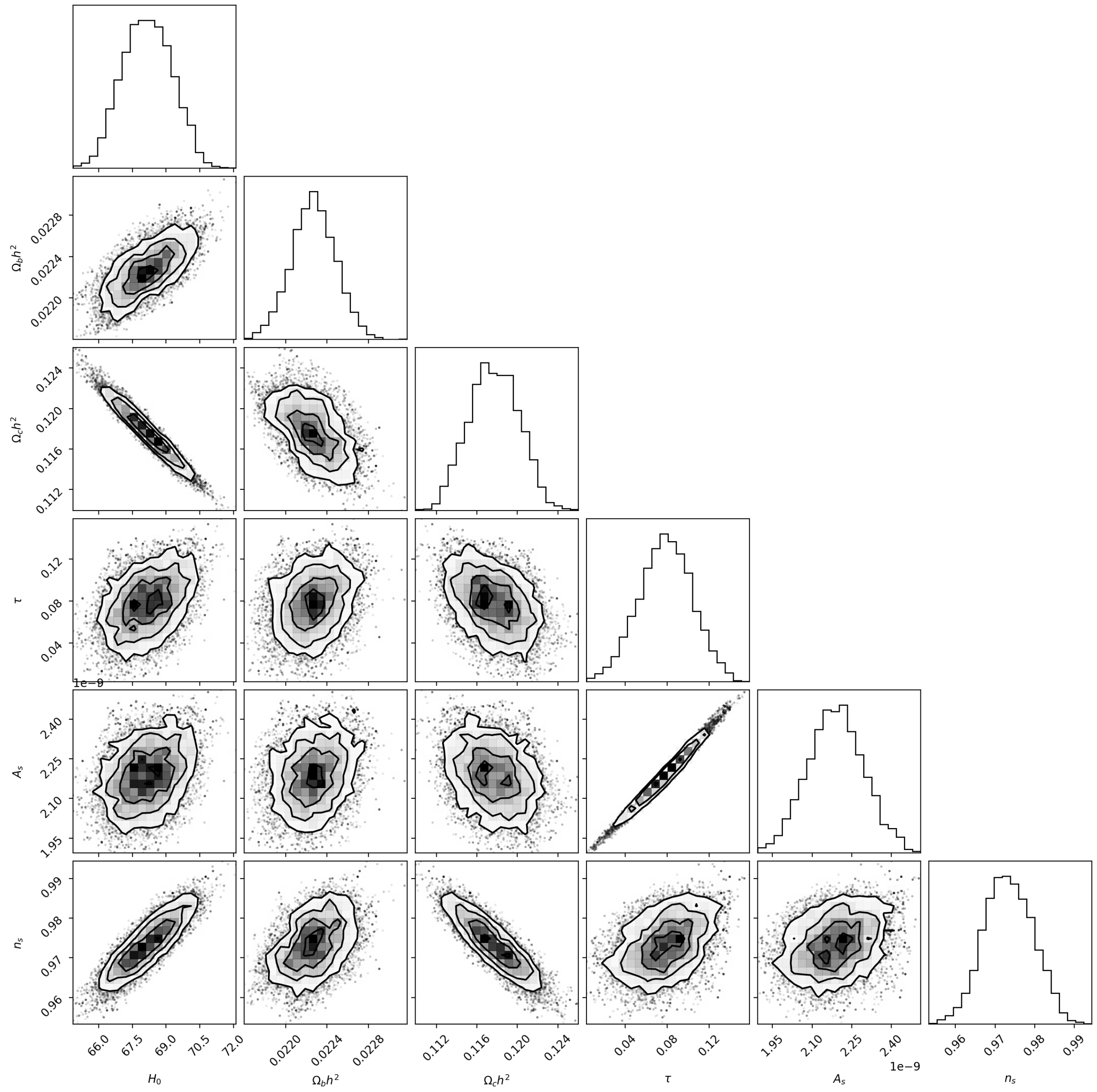

1e−9
A_s


A_s power spectrum


n_s


n_s power spectrum

This time, each power spectrum clearly has a bend with the lower frequencies levelling off, indicating better convergence (with the convergence of the parameter $\Omega_b h^2$ improving a lot). Here are the plots of the parameter covariances using `corner.py`:



These plots also show better convergence of the chain: the parameter distributions are much smoother, with a more well defined center.

The resulting means and errors of each parameter chain are:

$$
\begin{gathered}
H_0 = 6.832306979302595096e+01 \pm 1.160482230718751451e+00 \\
\Omega_b h^2 = 2.237641191021686946e-02 \pm 2.435043884764399728e-04 \\
\Omega_c h^2 = 1.175631073056536968e-01 \pm 2.578813538508824017e-03 \\
\tau = 8.725414744841814008e-02 \pm 2.979736648981657493e-02 \\
A_s = 2.230156561272920595e-09 \pm 1.247155332669573012e-10 \\
n_s = 9.736675070016088673e-01 \pm 6.310940039157754422e-03 \\
\chi^2 = 2576.956858582058
\end{gathered}
$$

Using this new covariance matrix improved the convergence of the chain quite a bit and also slightly lowered the χ^2. The importance sampling did give a better value for τ, with the new chain diverging quite a bit from the value of 0.0540 ± 0.0074.