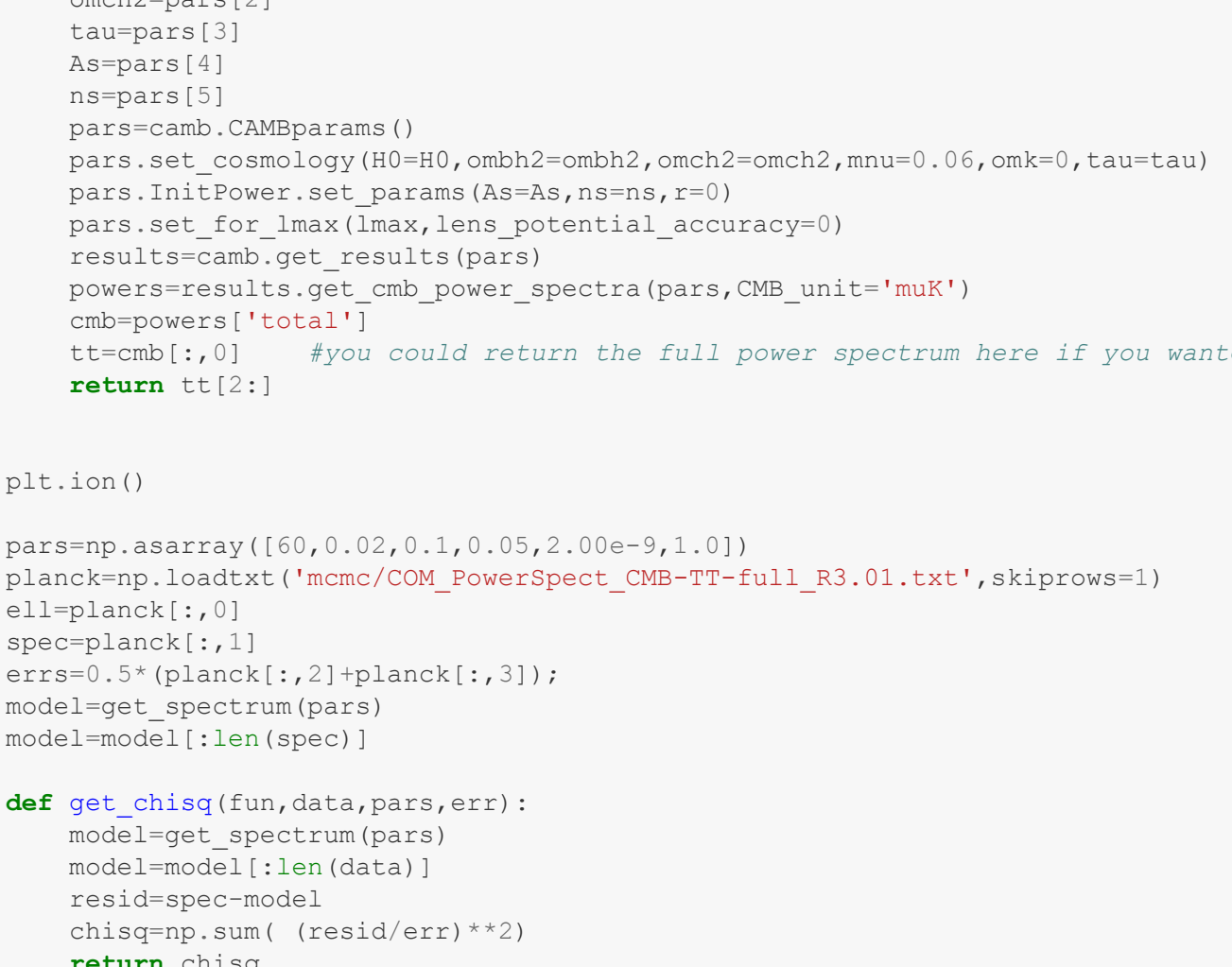
```
In [1]: %matplotlib notebook
        import numpy as np
        import matplotlib.pyplot as plt
        import camb
        from camb import model, initialpower
        from tqdm.notebook import tqdm
        import corner
        import sys, platform, os
        import time
        from IPython.display import display, Math
        camb.__version__,os.path.dirname(camb.__file__)
```

```
Out[1]: ('1.3.6', 'C:\\Users\\miara\\anaconda3\\lib\\site-packages\\camb')
```

```
In [2]: data = np.loadtxt('mcmc/COM_PowerSpect_CMB-TT-full_R3.01.txt')
```

```
In [3]: l = data[:,0]
        var = data[:,1]
        varerr = 1/2*(data[:,2]+data[:,3]) #error on datapoints is taken to be the average of the upper and low
        er errorbars
        N = np.diag(varerr**2) #N matrix
        Ninv = np.linalg.inv(N)
```

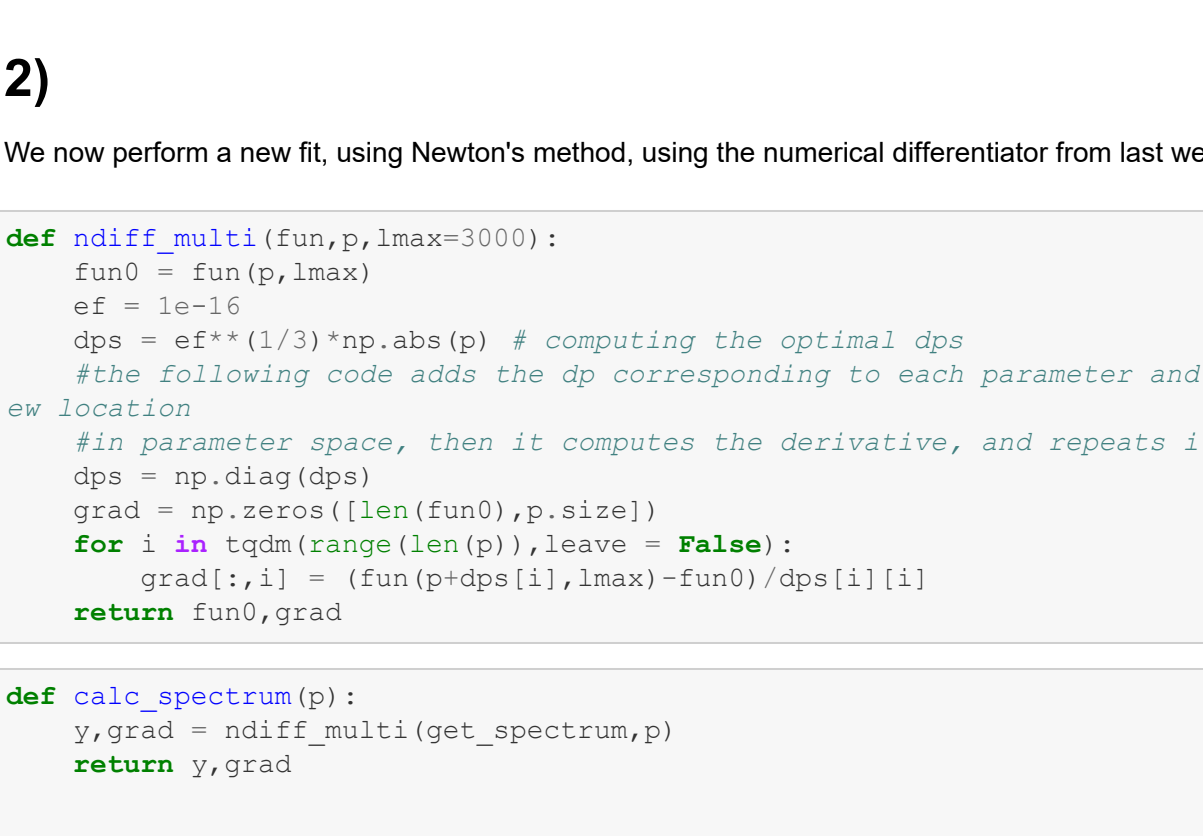
```
In [4]: #taking a look at the data

        fig, ax = plt.subplots()
        ax.plot(l,var,'.')
        ax.errorbar(l,var,varerr,color = 'k',alpha = 0.5)
        ax.set_xlabel(r'Multipole $\ell$')
        ax.set_ylabel('Variance')
        fig.tight_layout()
```



1)

First, we use the function from the test script to compute a "test fit" to see that everything is working, and we can also compute the χ^2 value of this fit.

```
In [6]: def get_spectrum(pars,lmax=3000):
            #print('pars are ',pars)
            H0=pars[0]
            ombh2=pars[1]
            omch2=pars[2]
            tau=pars[3]
            As=pars[4]
            ns=pars[5]
            pars=camb.CAMBparams()
            pars.set_cosmology(H0=H0,ombh2=ombh2,omch2=omch2,mnu=0.06,omk=0,tau=tau)
            pars.InitPower.set_params(As=As,ns=ns,r=0)
            pars.set_for_lmax(lmax,lens_potential_accuracy=0)
            results=camb.get_results(pars)
            powers=results.get_cmb_power_spectra(pars,CMB_unit='muK')
            cmb=powers['total']
            tt=cmb[:,0]    #you could return the full power spectrum here if you wanted to do say EE
            return tt[2:]


        plt.ion()

        pars=np.asarray([60,0.02,0.1,0.05,2.00e-9,1.0])
        planck=np.loadtxt('mcmc/COM_PowerSpect_CMB-TT-full_R3.01.txt',skiprows=1)
        ell=planck[:,0]
        spec=planck[:,1]
        errs=0.5*(planck[:,2]+planck[:,3]);
        model=get_spectrum(pars)
        model=model[:len(spec)]

        def get_chisq(fun,data,pars,err):
            model=get_spectrum(pars)
            model=model[:len(data)]
            resid=spec-model
            chisq=np.sum( (resid/err)**2)
            return chisq

        chisq = get_chisq(get_spectrum,spec,pars,varerr)

        print("chisq is ",chisq," for ",len(spec)-len(pars)," degrees of freedom.")
        #read in a binned version of the Planck PS for plotting purposes
        planck_binned=np.loadtxt('mcmc/COM_PowerSpect_CMB-TT-binned_R3.01.txt',skiprows=1)
        errs_binned=0.5*(planck_binned[:,2]+planck_binned[:,3]);
        fig,ax = plt.subplots()
        ax.plot(ell,model)
        ax.errorbar(planck_binned[:,0],planck_binned[:,1],errs_binned,fmt='.')

        chisq is  15267.937150261654  for  2501  degrees of freedom.
```



```
Out[6]: <ErrorbarContainer object of 3 artists>
```

We see that the value of χ^2 is around 15000 for 2501 degrees of freedom, which indicates that these parameters are not a good fit for the data we have. If the fit was good, χ^2 would be on the order of 2500. Now, let us compute the value of χ^2 for the parameters suggested in the assignment.

```
In [7]: pars =  np.asarray([69, 0.022, 0.12,0.06, 2.1e-9, 0.95])
        chisq = get_chisq(get_spectrum,spec,pars,varerr)
        print("chisq is ",chisq," for ",len(spec)-len(pars)," degrees of freedom.")

        chisq is  3272.2053559202204  for  2501  degrees of freedom.
```

This fit is slightly better, but still not such a good fit.

2)

We now perform a new fit, using Newton's method, using the numerical differentiator from last week's problem set.

```
In [8]: def ndiff_multi(fun,p,lmax=3000):
            fun0 = fun(p,lmax)
            ef = 1e-16
            dps = ef**(1/3)*np.abs(p) # computing the optimal dps
            #the following code adds the dp corresponding to each parameter and evaluates the function at the n
        ew location
            #in parameter space, then it computes the derivative, and repeats it for all parameters in fun
            dps = np.diag(dps)
            grad = np.zeros([len(fun0),p.size])
            for i in tqdm(range(len(p)),leave = False):
                grad[:,i] = (fun(p+dps[i],lmax)-fun0)/dps[i][i]
            return fun0,grad
```

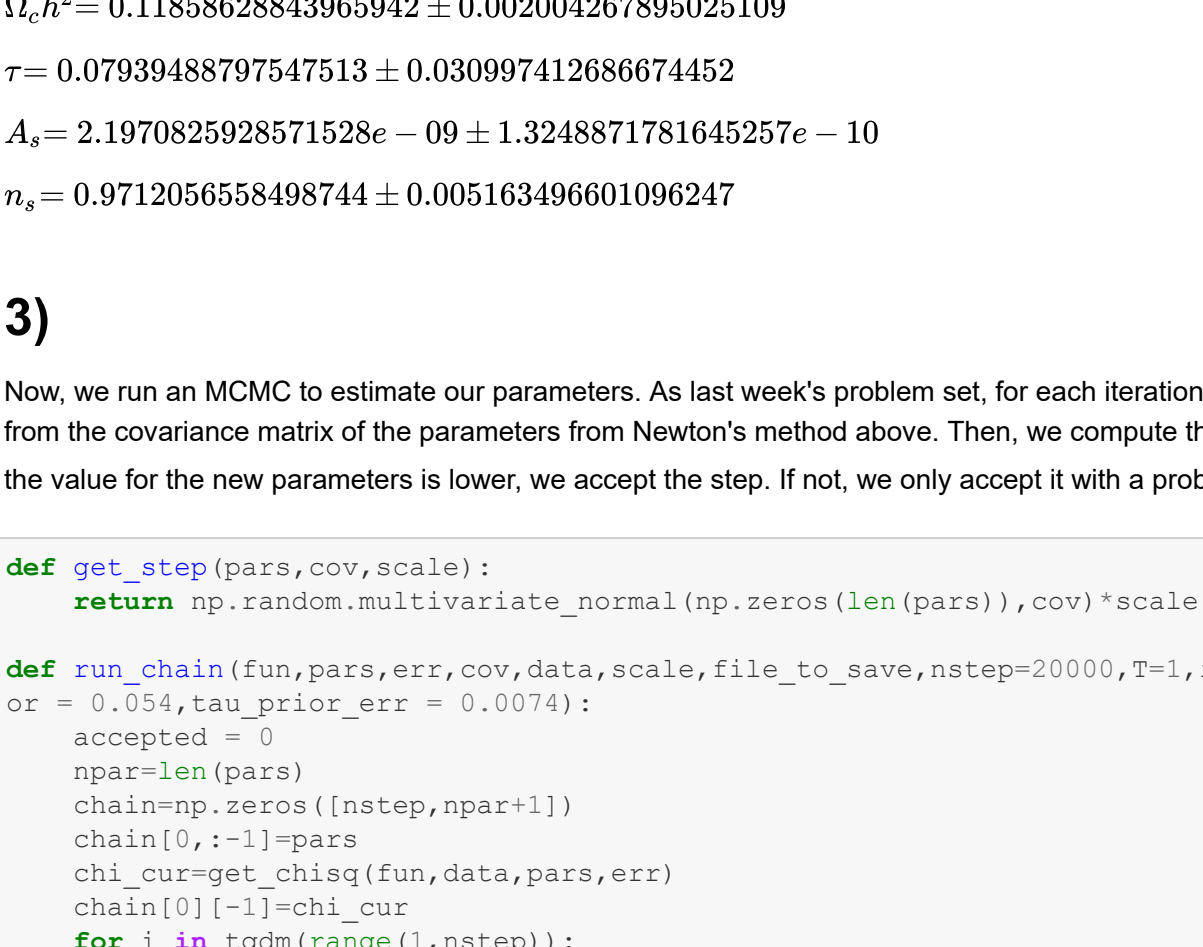
```
In [9]: def calc_spectrum(p):
            y,grad = ndiff_multi(get_spectrum,p)
            return y,grad


        iter = 5
        p0=np.asarray([69, 0.022, 0.12,0.06, 2.1e-9, 0.95])
        p=p0.copy()
        newt_fit = np.zeros([iter+1,p.size+1])
        newt_fit[0][-1] = get_chisq(get_spectrum,spec,p0,varerr)
        newt_fit[0][:-1] = p0

        for i in tqdm(range(iter)):
            pred,grad=calc_spectrum(p)
            pred,grad = pred[:len(spec)],grad[:len(spec)]
            resid=spec-pred
            r=resid.T
            lhs=grad.T@Ninv@grad
            rhs=grad.T@Ninv@r
            cov = np.linalg.inv(lhs)
            dp=cov@rhs
            for ii in range(p.size):
                p[ii]=p[ii]+dp[ii]
            chisq=np.sum((resid/varerr)**2)
            newt_fit[i+1][-1] = chisq
            newt_fit[i+1][:-1] = p

        np.savetxt('planck fit params.txt',newt_fit)
```

We can verify that Newton's method has converged by looking at the difference between the last 2 χ^2 values:

```
In [10]: np.abs(newt_fit[-1][-1]-newt_fit[-2][-1])

Out[10]: 0.002059310299500794
```

Because this value is <<1, we can be confident that Newton's method has converged. We can now look at our improved fit.

```
In [11]: fig,ax = plt.subplots()
         ax.plot(ell,model,label = 'Fit at starting parameters')
         ax.errorbar(planck_binned[:,0],planck_binned[:,1],errs_binned,fmt='.',label = 'Data')
         model2=get_spectrum(p)
         model2=model2[:len(spec)]
         ax.plot(ell,model2,label = 'Fit at Newton\'s method parameters')
         ax.legend()
         noise = np.mean(np.abs(model2-spec))
         chisq = get_chisq(get_spectrum,spec,p,varerr)
         print("chisq is ",chisq," for ",len(spec)-len(pars)," degrees of freedom.")
```




```
chisq is  2576.3052119680515  for  2501  degrees of freedom.
```

```
In [12]: p_err = np.sqrt(np.diag(cov))
         params = [r'$H_0$',r'$\Omega_bh^2$',r'$\Omega_ch^2$',r'$\tau$',r'$A_s$',r'$n_s$']
         print('Best fit parameters using Newton\'s method :\n')
         for i in range(len(p)):
             display(Math(params[i]+'$ = {} \pm {}$'.format(p[i],p_err[i])))

         Best fit parameters using Newton's method :
```

$H_0 = 67.81986034221636 \pm 0.8093944887984065$

$\Omega_bh^2 = 0.02229227267997955 \pm 0.0001629673725677291$

$\Omega_ch^2 = 0.11858628843965942 \pm 0.002004267895025109$

$\tau = 0.07939488797547513 \pm 0.030997412686674452$

$A_s = 2.1970825928571528e-09 \pm 1.3248871781645257e-10$

$n_s = 0.9712056558498744 \pm 0.005163496601096247$

3)

Now, we run an MCMC to estimate our parameters. As last week's problem set, for each iteration of the parameters, we generate a trial step from the covariance matrix of the parameters from Newton's method above. Then, we compute the value of χ^2 for these new parameters. If the value for the new parameters is lower, we accept the step. If not, we only accept it with a probability $e^{-\frac{1}{2}\delta\chi^2}$.

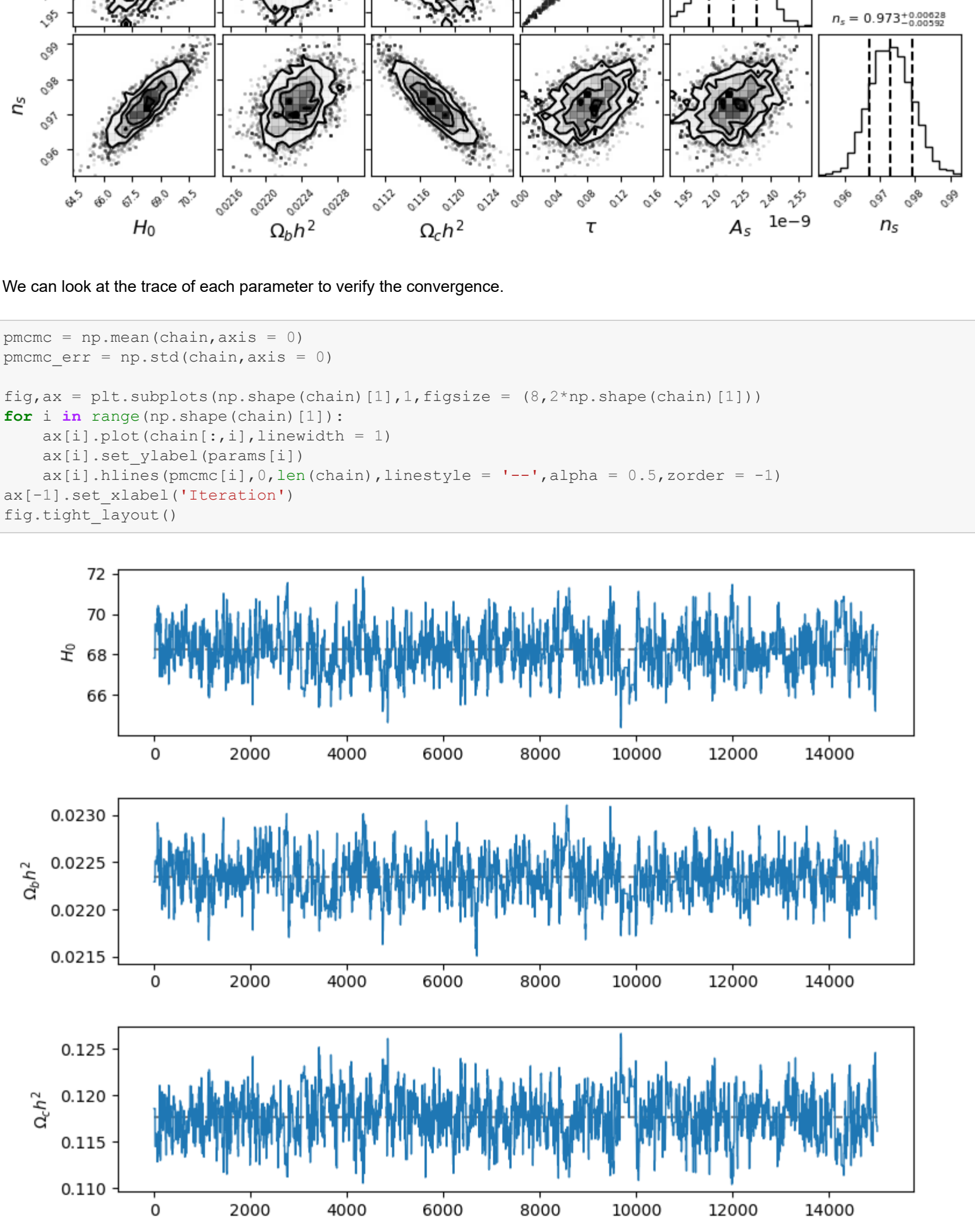
```
In [13]: def get_step(pars,cov,scale):
             return np.random.multivariate_normal(np.zeros(len(pars)),cov)*scale

         def run_chain(fun,pars,err,cov,data,scale,file_to_save,nstep=20000,T=1,include_tauprior = False,tau_pri
         or = 0.054,tau_prior_err = 0.0074):
             accepted = 0
             npar=len(pars)
             chain=np.zeros([nstep,npar+1])
             chain[0,:-1]=pars
             chi_cur=get_chisq(fun,data,pars,err)
             chain[0][-1]=chi_cur
             for i in tqdm(range(1,nstep)):
                 pp=pars+get_step(pars,cov,scale)
                 new_chisq=get_chisq(fun,data,pp,err)
                 if include_tauprior:
                     new_chisq+=(pp[3]-tau_prior)**2/tau_prior_err**2
                 accept_prob=np.exp(-0.5*(new_chisq-chi_cur)/T)
                 if np.random.rand(1)<accept_prob:
                     accepted+=1
                     pars=pp
                     chi_cur=new_chisq
                 chain[i,:-1]=pars
                 chain[i][-1]=chi_cur
                 path = file_to_save
                 np.savetxt(path,chain)
             accrate = accepted/nstep
             chisq = chain[:,-1]
             chain = chain[:,:-1]
             return chain,chisq,accrate
```

```
In [ ]: nstep = 15000
        chain,chisq,accrate = run_chain(get_spectrum,p,varerr,cov,spec,scale = 1,file_to_save = 'Chains/planck_
        chain_{}.txt'.format(nstep),nstep = nstep)
```

Once our chain has run, we can load it up and look at it:

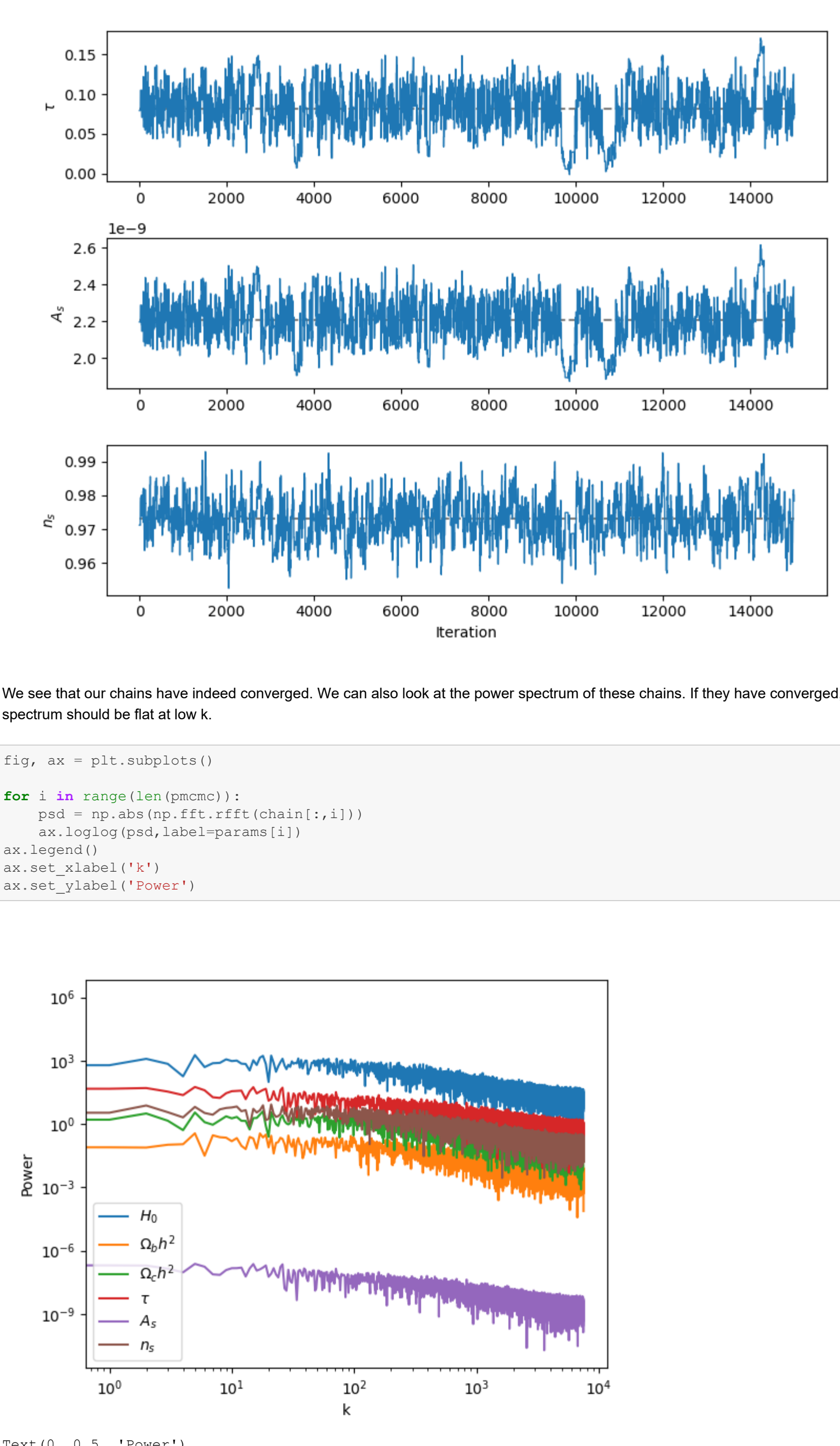
```
In [18]: chainload = np.loadtxt('Chains/planck_chain_{}.txt'.format(15000))
         chain = chainload[:,:-1]
```

```
In [19]: fig = corner.corner(chain,labels = params,label_kwargs=dict(fontsize=12),
                             quantiles = [0.16,.5,.84],show_titles=True,title_fmt='.3g',title_kwargs = dict(font
         size = 8))
         fig.set_size_inches((8.5,8.5))
         for ax in fig.get_axes():
             ax.tick_params(axis='both', labelsize=6)
```



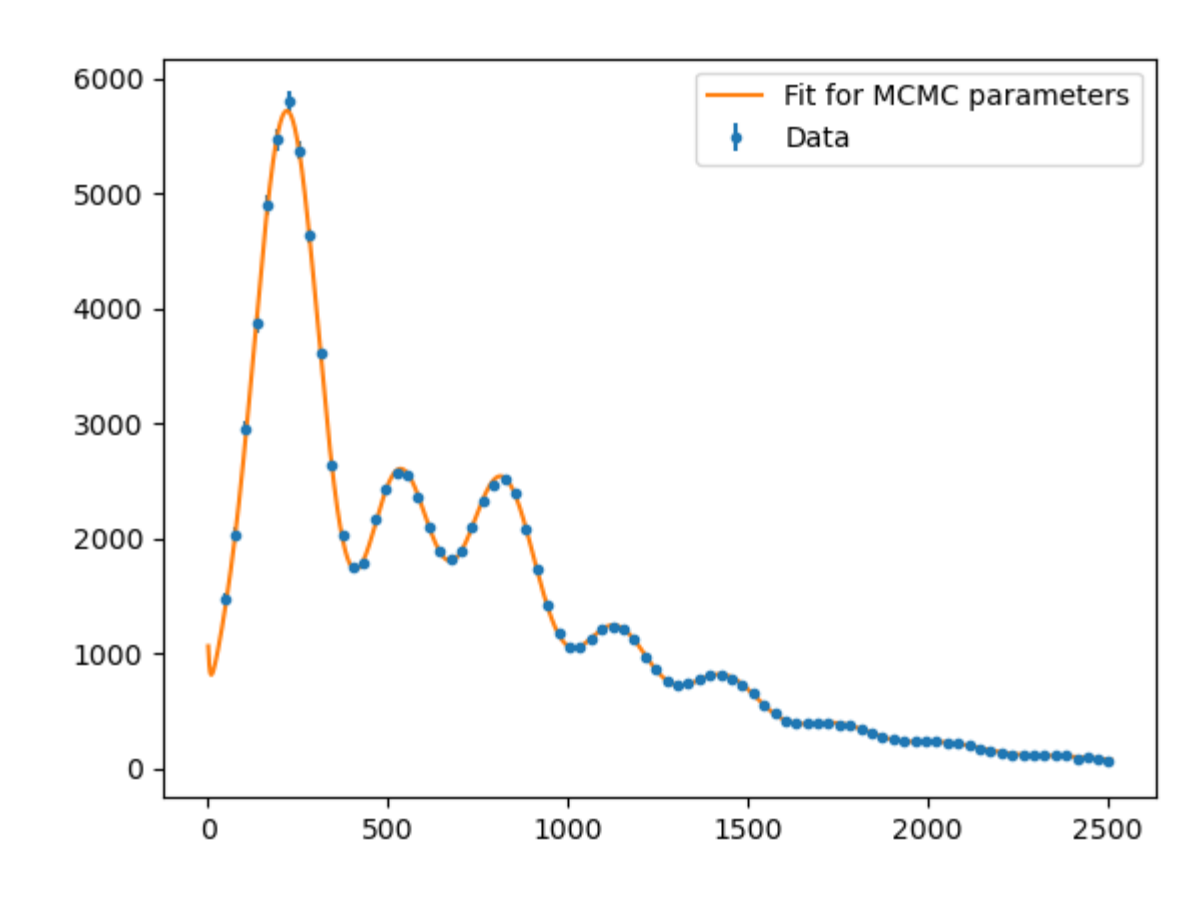
We can look at the trace of each parameter to verify the convergence.

```
In [20]: pmcmc = np.mean(chain,axis = 0)
         pmcmc_err = np.std(chain,axis = 0)

         fig,ax = plt.subplots(np.shape(chain)[1],1,figsize = (8,2*np.shape(chain)[1]))
         for i in range(np.shape(chain)[1]):
             ax[i].plot(chain[:,i],linewidth = 1)
             ax[i].set_ylabel(params[i])
             ax[i].hlines(pmcmc[i],0,len(chain),linestyle = '--',alpha = 0.5,zorder = -1)
         ax[-1].set_xlabel('Iteration')
         fig.tight_layout()
```



We see that the chains have indeed converged. We can also look at the power spectrum of these chains. If they have converged, the spectrum should be flat at low k.

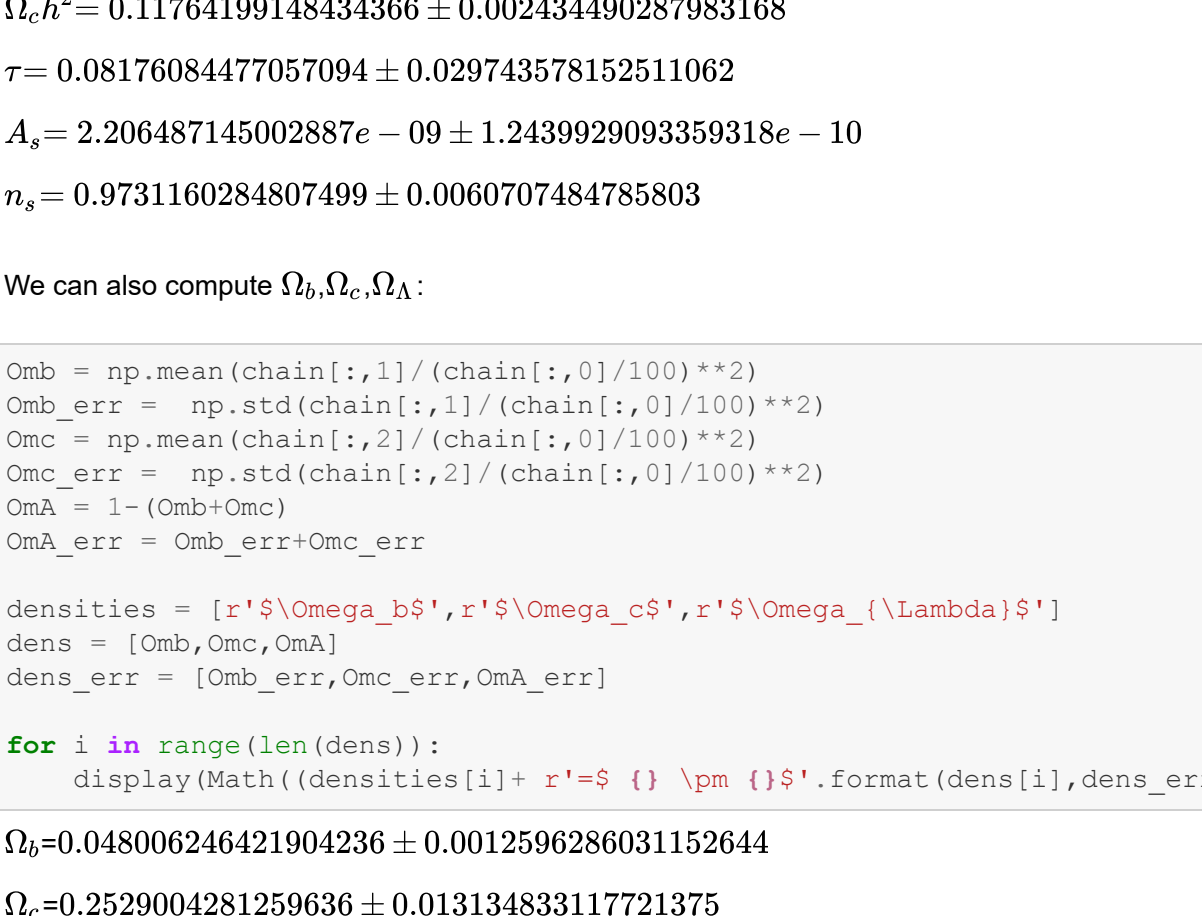
```
In [23]: fig, ax = plt.subplots()
         for i in range(len(pmcmc)):
             psd = np.abs(np.fft.rfft(chain[:,i]))
             ax.loglog(psd,label=params[i])
         ax.legend()
         ax.set_xlabel('k')
         ax.set_ylabel('Power')
```



```
Out[23]: Text(0, 0.5, 'Power')
```

We see that the power spectra for each parameter looks like what we expect for converged chains. Let us now look at the actual fit.

```
In [24]: fig,ax = plt.subplots()
         ax.errorbar(planck_binned[:,0],planck_binned[:,1],errs_binned,fmt='.',label = 'Data')
         model2=get_spectrum(pmcmc)
         model2=model2[:len(spec)]
         ax.plot(ell,model2,label = 'Fit for MCMC parameters')
         ax.legend()
         chisq = get_chisq(get_spectrum,spec,pmcmc,varerr)
         print("chisq is ",chisq," for ",len(spec)-len(pars)," degrees of freedom.")
```




```
chisq is  2577.4758700939155  for  2501  degrees of freedom.
```

It looks good, however, the chi square is pretty much the same as for Newton's method.

```
In [25]: print('Best fit parameters using an MCMC:\n')
         for i in range(len(pmcmc)):
             display(Math(params[i]+'$ = {} \pm {}$'.format(pmcmc[i],pmcmc_err[i])))

         Best fit parameters using an MCMC:
```

$H_0 = 68.25077453291213 \pm 1.0813838968974272$

$\Omega_bh^2 = 0.02234985492258494 \pm 0.0002164684677261199$

$\Omega_ch^2 = 0.11764199148434366 \pm 0.002434490287983168$

$\tau = 0.08176084477057094 \pm 0.029743578152511062$

$A_s = 2.206487145002887e-09 \pm 1.2439929093359318e-10$

$n_s = 0.9731160284807499 \pm 0.0060707484785803$

We can also compute $\Omega_b, \Omega_c, \Omega_\Lambda$:

```
In [26]: Omb = np.mean(chain[:,1]/(chain[:,0]/100)**2)
         Omb_err =  np.std(chain[:,1]/(chain[:,0]/100)**2)
         Omc = np.mean(chain[:,2]/(chain[:,0]/100)**2)
         Omc_err =  np.std(chain[:,2]/(chain[:,0]/100)**2)
         OmA = 1-(Omb+Omc)
         OmA_err = Omb_err+Omc_err

         densities = [r'$\Omega_b$',r'$\Omega_c$',r'$\Omega_{\Lambda}$']
         dens = [Omb,Omc,OmA]
         dens_err = [Omb_err,Omc_err,OmA_err]

         for i in range(len(dens)):
             display(Math(densities[i]+r'$ = {} \pm {}$'.format(dens[i],dens_err[i])))
```

$\Omega_b = 0.048006246421904236 \pm 0.0012596286031152644$

$\Omega_c = 0.2529004281259636 \pm 0.013134633117721375$

$\Omega_\Lambda = 0.6990933254521322 \pm 0.0143946172083664$

4)

We now add the constraint from the polarization data that the distribution of τ should have a mean of 0.054 and a standard deviation of 0.0074. First, we will importance sample our previous chain with that constraint. This effectively comes back to assigning a weight on each iterations of parameters of the chain such that this weight is proportional to how close the parameter τ from that iteration was to the prior. Quantitatively, each weight is $e^{-\frac{1}{2}\delta\tau^2/\sigma_\tau^2}$. This gives us a new set of parameters that agree with the prior knowledge on τ.

```
In [27]: def process_chain(chain,chisq,T=1.0,tau_prior = 0.054,tau_err = 0.0074):
             dtau = chain[:,3]-tau_prior
             wt=np.exp(-0.5*dtau**2/tau_err**2)
             npar=chain.shape[1]
             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)
             #divide by sum or weights
             mean=tot/np.sum(wt)
             meansqr=totsqr/np.sum(wt)

             #variance is <x^2>-<x>^2
             var=meansqr-mean**2
             return mean,np.sqrt(var),wt
```

```
In [28]: chisq = chainload[:,-1]
         print('Best fit parameters with the importance sampled MCMC:\n')
         pimpsamp,pimpsamp_err,w = process_chain(chain,chisq)
         for i in range(len(pimpsamp)):
             display(Math(params[i]+'$ = {} \pm {}$'.format(pimpsamp[i],pimpsamp_err[i])))

         Best fit parameters with the importance sampled MCMC:
```

$H_0 = 67.87073884097363 \pm 0.9609301260723686$

$\Omega_bh^2 = 0.022299806667003968 \pm 0.0002061890583405401$

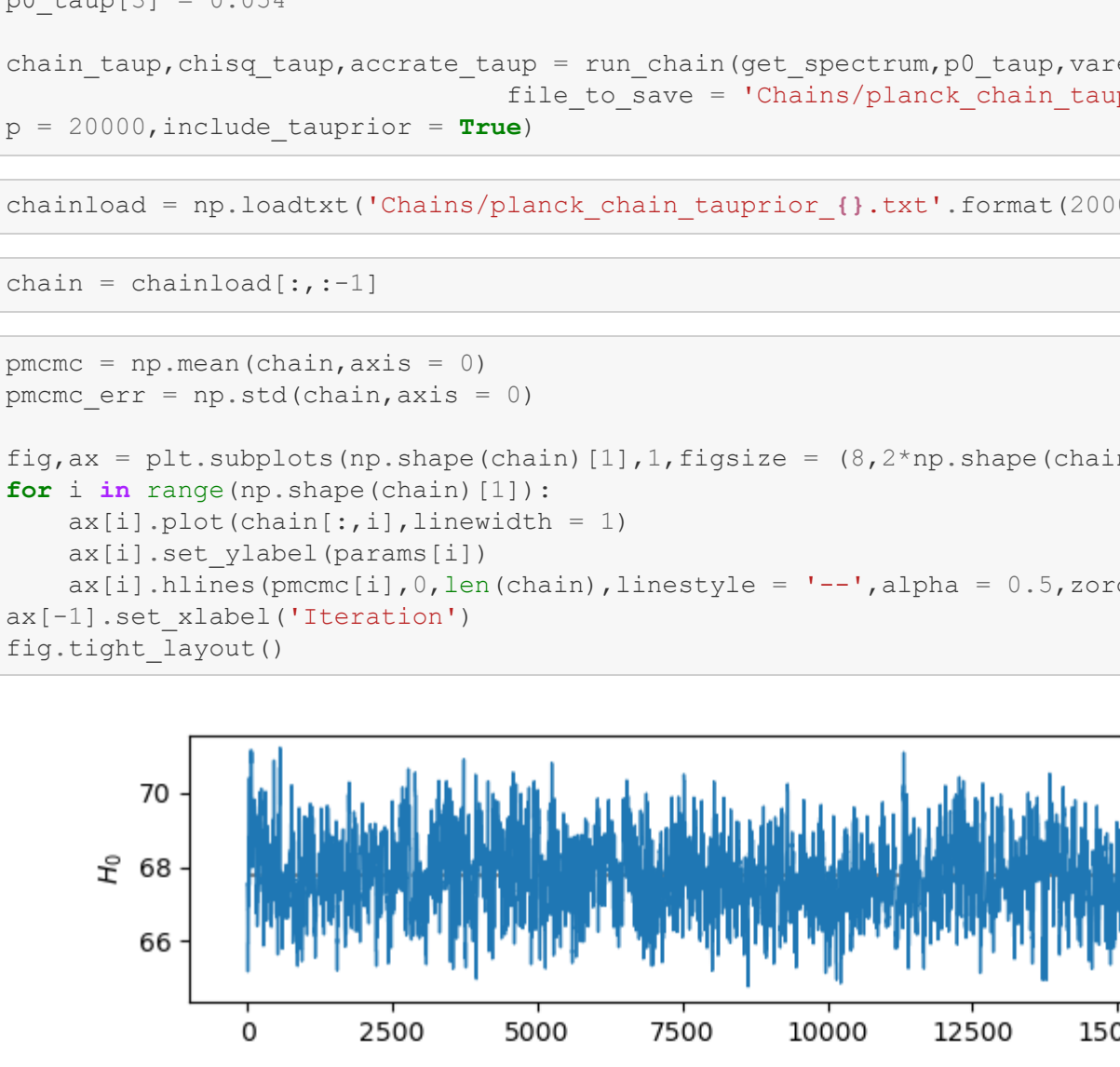
$\Omega_ch^2 = 0.11844156523227146 \pm 0.002192271844630678$

$\tau = 0.05589224036403911 \pm 0.00712548428933894$

$A_s = 2.096977296342167e-09 \pm 3.0497072584057284e-11$

$n_s = 0.9706665036744093 \pm 0.005625002213305259$

```
In [29]: fig,ax = plt.subplots()
         ax.errorbar(planck_binned[:,0],planck_binned[:,1],errs_binned,fmt='.',label = 'Data')
         model2=get_spectrum(pimpsamp)
         model2=model2[:len(spec)]
         ax.plot(ell,model2,label = 'Fit at MCMC method parameters')
         ax.legend()
         chisq = get_chisq(get_spectrum,spec,pimpsamp,varerr)
         print("chisq is ",chisq," for ",len(spec)-len(pars)," degrees of freedom.")
```




```
chisq is  2577.0796104116343  for  2501  degrees of freedom.
```

Again, the fit looks sensible, but now the other parameters have "ajusted" such that χ^2 remains pretty much unchanged, but our parameter τ now agrees with it's constraints from the polarization data. We can run a new chain using this constraint to compare the resulting parameters to those of the importance sampled chain. We will do so by slightly modifying our likelihood function, where as before, it was defined as $e^{-\frac{1}{2}\delta\chi^2}$, it is now $e^{-\frac{1}{2}\delta\tau^2/\sigma_\tau^2} \cdot e^{-\frac{1}{2}\delta\chi^2}$. However, we will now compute our covariance matrix from the weighted chain such that we obtain a faster convergence. We will also start our guess of τ at the value of the prior. As we did for the previous chain, we look at the traces of the parameters to verify convergence, we look at the corner plots because they're cool, and we look at the value of our parameters.

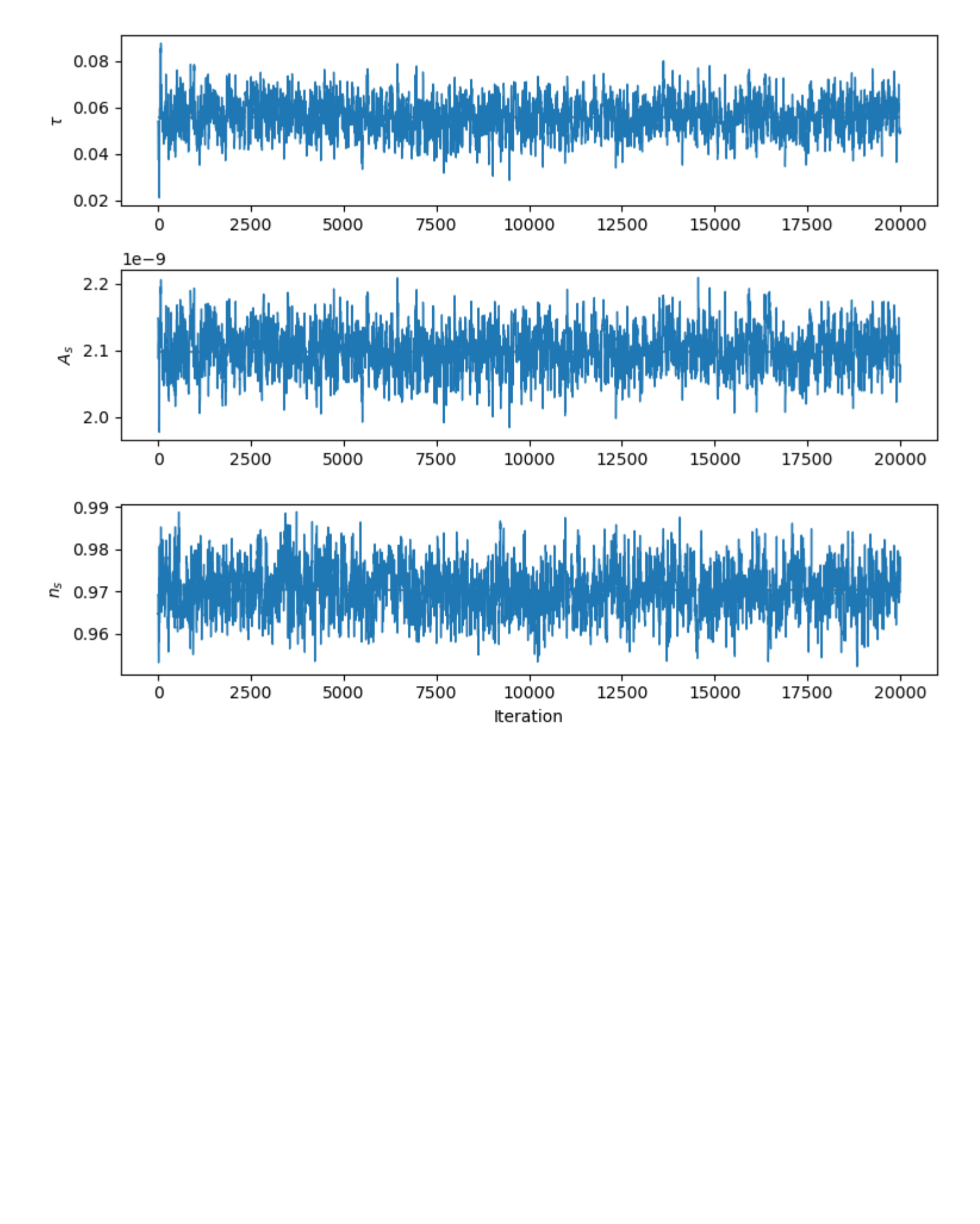
```
In [30]: mcmccov = np.cov(chain.T,aweights=w)
```

```
In [ ]: p0_taup = pmcmc.copy()
        p0_taup[3] = 0.054

        chain_taup,chisq_taup,accrate_taup = run_chain(get_spectrum,p0_taup,varerr,mcmccov,spec,scale = 1,
                                          file_to_save = 'Chains/planck_chain_tauprior_{}.txt'.format(20000),nste
        p = 20000,include_tauprior = True)
```

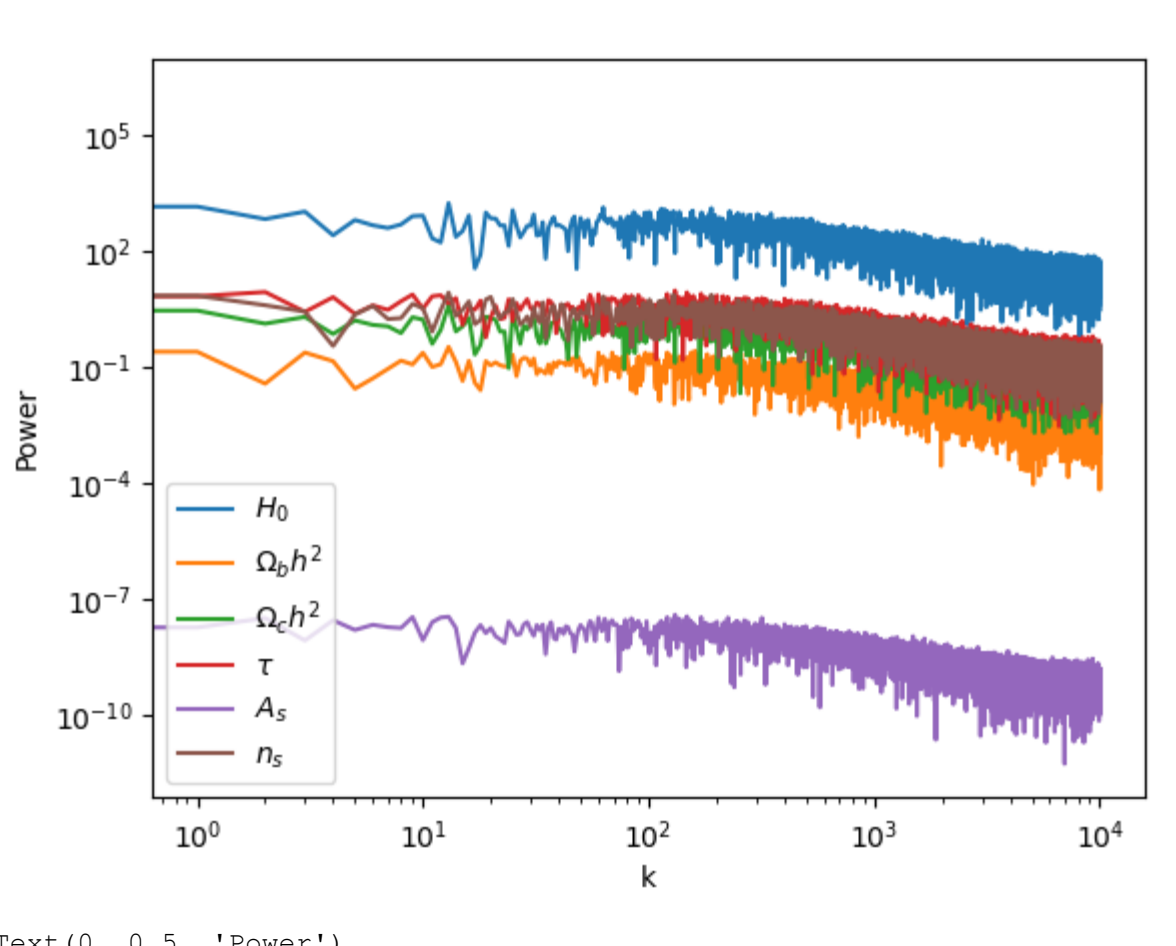
```
In [31]: chainload = np.loadtxt('Chains/planck_chain_tauprior_{}.txt'.format(20000))
```

```
In [32]: chain = chainload[:,:-1]
```

```
In [33]: pmcmc = np.mean(chain,axis = 0)
         pmcmc_err = np.std(chain,axis = 0)

         fig,ax = plt.subplots(np.shape(chain)[1],1,figsize = (8,2*np.shape(chain)[1]))
         for i in range(np.shape(chain)[1]):
             ax[i].plot(chain[:,i],linewidth = 1)
             ax[i].set_ylabel(params[i])
             ax[i].hlines(pmcmc[i],0,len(chain),linestyle = '--',alpha = 0.5,zorder = -1)
         ax[-1].set_xlabel('Iteration')
         fig.tight_layout()
```

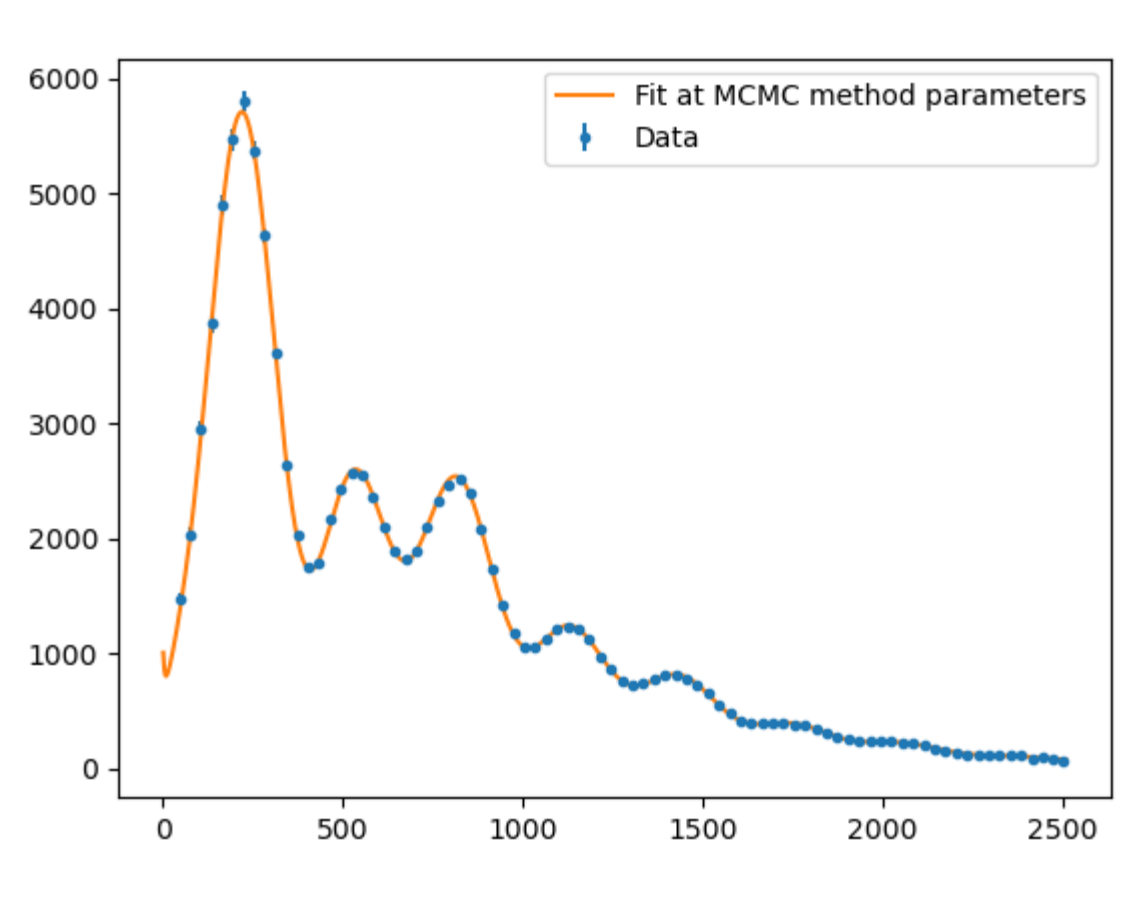


In [34]:
```
fig, ax = plt.subplots()

for i in range(len(pmcmc)):
    psd = np.abs(np.fft.rfft(chain[:,i]))
    ax.loglog(psd,label=params[i])
ax.legend()
ax.set_xlabel('k')
ax.set_ylabel('Power')
```



Out[34]: `Text(0, 0.5, 'Power')`

In [35]:
```
fig,ax = plt.subplots()
ax.errorbar(planck_binned[:,0],planck_binned[:,1],errs_binned,fmt='.',label = 'Data')
model2=get_spectrum(pmcmc)
model2=model2[:len(spec)]
ax.plot(ell,model2,label = 'Fit at MCMC method parameters')
ax.legend()
chisq = get_chisq(get_spectrum,spec,pmcmc,varerr)
print("chisq is ",chisq," for ",len(spec)-len(pars)," degrees of freedom.")
```




```
chisq is  2577.043102636848  for  2501  degrees of freedom.
```

In [36]:
```
print('Best fit parameters for the chain with the constraint:\n')
for i in range(len(pmcmc)):
    display(Math(params[i]+r'$= {} \pm {}$'.format(pmcmc[i],pmcmc_err[i])))

print('\nBest fit parameters for the chain with the constraint:\n')
for i in range(len(pmcmc)):
    display(Math(params[i]+r'$= {} \pm {}$'.format(pimpsamp[i],pimpsamp_err[i])))
```

```
Best fit parameters for the chain with the constraint:
```

$H_0 = 67.76346189604908 \pm 1.0008806776718875$

$\Omega_b h^2 = 0.02229694213976638 \pm 0.00021528509595358883$

$\Omega_c h^2 = 0.11873758371378285 \pm 0.0022660402476238958$

$\tau = 0.055928979114304674 \pm 0.007341093217705456$

$A_s = 2.0984807374930637e - 09 \pm 3.1452296824359454e - 11$

$n_s = 0.9703921091096512 \pm 0.0053907417373109875$

```
Best fit parameters for the chain with the constraint:
```

$H_0 = 67.87073884097363 \pm 0.9609301260723686$

$\Omega_b h^2 = 0.022299806667003968 \pm 0.00020618905834054\01$

$\Omega_c h^2 = 0.11844156523227146 \pm 0.002192271844630678$

$\tau = 0.05589224036403911 \pm 0.00712548428933894$

$A_s = 2.096977296342167e - 09 \pm 3.0497072584057284e - 11$

$n_s = 0.9706665036744093 \pm 0.005625002213305259$

We see that there is no statistical difference between those two sts of parameters. Hence, importance sampling has the same effect as running a whole new chain with a constraint, but the first method takes a few milliseconds to run, and the other takes a couple of hours.

In [ ]: