Problem Set 5 - Matias Castro Tapia

In [1]:

```
import numpy as np
import camb
from matplotlib import pyplot as plt
import time
```

```
C:\Users\Odette\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarni
ng: A NumPy version >=1.16.5 and <1.23.0 is required for this version of Sci
Py (detected version 1.23.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

First, I tried the code planck_likelihood.py provided in the mcmc folder. I copied the code below.

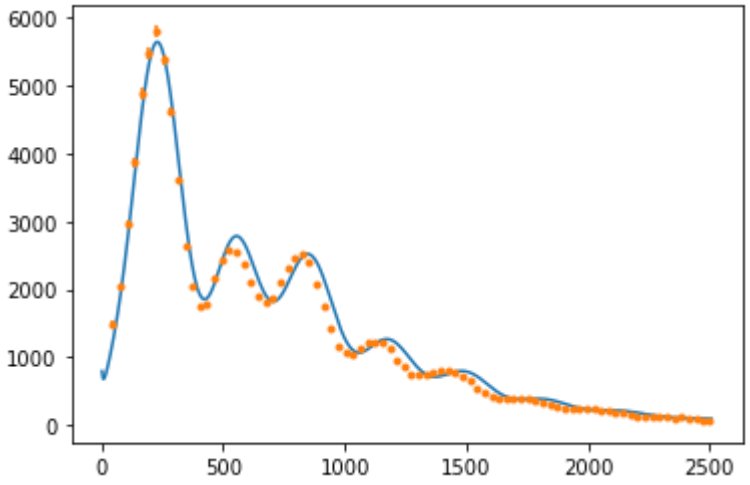
In [2]:

```
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 E
    return tt[2:]


#plt.ion()

pars=np.asarray([60,0.02,0.1,0.05,2.00e-9,1.0])
planck=np.loadtxt('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)]
resid=spec-model
chisq=np.sum( (resid/errs)**2)
print("chisq is ",chisq," for ",len(resid)-len(pars)," degrees of freedom.")
#read in a binned version of the Planck PS for plotting purposes
planck_binned=np.loadtxt('COM_PowerSpect_CMB-TT-binned_R3.01.txt',skiprows=1)
errs_binned=0.5*(planck_binned[:,2]+planck_binned[:,3]);
#plt.clf()
plt.plot(ell,model)
plt.errorbar(planck_binned[:,0],planck_binned[:,1],errs_binned,fmt='.')
plt.show()
```

```
chisq is  15267.937150261656  for  2501  degrees of freedom.
```



Then, I tried obtaining the model using the set of parameters [69,0.022,0.12,0.06,2.10e-9,0.95].

In [3]:

```
pars=np.asarray([69,0.022,0.12,0.06,2.10e-9,0.95])
model=get_spectrum(pars)
model=model[:len(spec)]
resid=spec-model
chisq=np.sum( (resid/errs)**2)
print("chisq is ",chisq," for ",len(resid)-len(pars)," degrees of freedom.")
```

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

In [4]:

```
np.sqrt(2*2501),2501+np.sqrt(2*2501),2501+11*np.sqrt(2*2501)
```

Out[4]:

```
(70.7248188403477, 2571.724818840348, 3278.9730072438247)
```

I obtained $\chi^2 = 3272.2053559202204$. Since, there are 2501 degrees of freedom and the variance of the χ^2 is 2 times the degrees of freedom, then $\sigma_{\chi^2} = 70.72481$. The χ^2 obtained with this set of parameters is not good enough since this is very out from the expected (mean) value of χ^2, which should be about the value of degrees of freedom. However, this value is much better than the 15267.937150261656 obtained for the trial in planck_likelihood.py. Thus, this new set of parameters is more likely than the first se since is closer to the n degrees of freedom.

I defined the routine ev_planck to obtain the model spectrum using the get_spectrum routine from planck_likelihood.py. ev_planck returns the model with the length of the data in the first column of COM_PowerSpect_CMB-TT-full_R3.01.txt

I also defined the routine deriv to obtain the numerical partial derivatives of a function G as follows:

$$\frac{\partial G}{\partial m_i} = \frac{G(x, y, m_1, \ldots m_i + h_i, \ldots, m_n) - G(x, y, m_1, \ldots m_i - h_i, \ldots, m_n)}{2h_i}$$

This beause all the parameters have different orders of magnitude. hh is an array with different scales of h to estimate every numerical partial derivative using a h smaller than the order of magnitude that we already used for each parameter.

In [5]:

```
def ev_planck(p):
    model=get_spectrum(p)
    model=model[:len(spec)]
    return model
```

In [6]:

```
def deriv(fun,p,h):
    y=fun(p)
    derivs=[]
    for i in range(len(p)):
        pl=p.copy()
        pm=p.copy()
        pl[i]=p[i]+h[i]
        pm[i]=p[i]-h[i]
        derivs.append((fun(pl)-fun(pm))/(2*h[i]))
    return y,np.array(derivs)
```

In [54]:

```
hh=np.array([1,0.001,0.01,0.01,1e-11,0.05])/1000
```

I defined Ninv to be N^{-1} of the data, so every element of the diagonal is $1/\sigma_i^2$ (and 0 for any other element out of the diagonal due to we are considering uncorrelated) with σ_i the error obtained from the data as $(\sigma_{i+} + \sigma_{i-})/2$.

In [55]:

```
Ninv=np.eye(len(errs))*(1/errs**2)
```

Then, I used the Newton's method to find a root of $\nabla\chi^2$ for the set of parameters m. Then, the iterative method can be applied using a Taylor expansion for $A(m) = A(m_0) + A'(m_0)\delta m$, and considering the residual $r = d - A(m)$, the new definition for $\chi^2 = (r - A'(m_0))^T N^{-1}(r - A'(m_0))$. And asuming that δm is very small we want to solve $\nabla\chi^2 = -A'^T(m_0)N^{-1}(r - A'(m_0)\delta m) = 0$. Thus, $\delta m = (A'^T(m_0)N^{-1}A'(m_0))^{-1}A'^T(m_0)N^{-1}r$, and the iteration for finding a root of $\nabla\chi^2$ is $m_{n+1} = m_n + \delta m$.

I also used the QR decomposition for $A'(m_0)$, then the estimation of δm is:

$$\delta m = R^{-1}(Q^T N^{-1} Q)^{-1} Q^T N^{-1} r$$

The parameters and errors are reported in planck_fit_params.txt.

In [115]:

```
p=pars.copy()
for j in range(20):
    y,der=deriv(ev_planck,p,hh)
    r=(spec-y)
    chisq=np.sum(((spec-y)**2)/(errs**2))
    A_m=np.ones([r.size,6])
    for i in range(6):
        A_m[:,i]=der[i]
    Q,R=np.linalg.qr(A_m)

    dp=np.linalg.inv(R)@np.linalg.inv(Q.T@Ninv@Q)@Q.T@Ninv@r
    for i in range(p.size):
        p[i]=p[i]+dp[i]
    errp=np.sqrt(np.diag(np.linalg.inv(A_m.T@Ninv@A_m)))
    Nm=np.linalg.inv(A_m.T@Ninv@A_m)
    with open('planck_fit_params.txt', 'a') as f:
        for k in p:
            f.write(str(k)+' ')
        for l in errp[0:-1]:
            f.write(str(l)+' ')
        f.write(str(errp[-1])+'\n')
    print(p,chisq,dp)
print(np.linalg.inv(A_m.T@Ninv@A_m))
```

```
[6.82344625e+01 2.23628848e-02 1.17685431e-01 8.47240930e-02
 2.21639573e-09 9.73001434e-01] 2576.1575893366007 [-7.78757823e-02 -8.909
98570e-06  1.78189838e-04 -1.51397965e-03
 -5.63615040e-12 -3.98935207e-04]
[6.82417834e+01 2.23638766e-02 1.17669376e-01 8.51834096e-02
 2.21823759e-09 9.73042689e-01] 2576.1534424325637 [ 7.32091837e-03  9.917
86495e-07 -1.60551498e-05  4.59316626e-04
  1.84185289e-12  4.12544725e-05]
[6.83082327e+01 2.23711937e-02 1.17516035e-01 8.61657615e-02
 2.22175922e-09 9.73382339e-01] 2576.152266607657 [ 6.64492395e-02  7.3170
9934e-06 -1.53341020e-04  9.82351857e-04
  3.52163544e-12  3.39650539e-04]
[6.82319965e+01 2.23625372e-02 1.17690899e-01 8.47627691e-02
 2.21631560e-09 9.72989540e-01] 2576.1571267456293 [-7.62362002e-02 -8.656
49511e-06  1.74864489e-04 -1.40299237e-03
 -5.44362589e-12 -3.92799237e-04]
[6.82422258e+01 2.23639924e-02 1.17668562e-01 8.51812869e-02
 2.21822450e-09 9.73044407e-01] 2576.1561373882696 [ 1.02293476e-02  1.455
22087e-06 -2.23369562e-05  4.18517747e-04
```

In [116]:

```
p,errp
```

Out[116]:

```
(array([6.82416236e+01, 2.23638984e-02, 1.17669875e-01, 8.51333156e-02,
        2.21802047e-09, 9.73041130e-01]),
 array([1.18632544e+00, 2.28910732e-04, 2.65264455e-03, 3.41070114e-02,
        1.43163834e-10, 6.56624303e-03]))
```

After 20 iterations the change in the parameters is about $m_i \times 10^{-3}$ and about 0.001 in the χ^2, then, the parameters obtained were $H_0 = 6.82416236e+01$, $\Omega_b h^2 = 2.23638984e-02$, $\Omega_c h^2 = 1.17669875e-01$, $\tau = 8.51333156e-02$, $A_s = 2.21802047e-09$, $n_s = 9.73041130e-01$.

And the errors: $\sigma_{H_0} = 1.18632544e+00$, $\sigma_{\Omega_b h^2} = 2.28910732e-04$, $\sigma_{\Omega_c h^2} = 2.65264455e-03$, $\sigma_\tau = 3.41070114e-02$, $\sigma_{A_s} = 1.43163834e-10$, $\sigma_{n_s} = 6.56624303e-03$.

In [58]:

```
Nm
```

Out[58]:

```
array([[ 1.40736806e+00,  1.91980999e-04, -3.07098491e-03,
         2.24717010e-02,  8.31440168e-11,  6.89892617e-03],
       [ 1.91980999e-04,  5.24001233e-08, -3.52274631e-07,
         2.98035612e-06,  1.14679314e-14,  7.42832963e-07],
       [-3.07098491e-03, -3.52274631e-07,  7.03652313e-06,
        -4.93716268e-05, -1.80350373e-13, -1.56028162e-05],
       [ 2.24717010e-02,  2.98035612e-06, -4.93716268e-05,
         1.16328823e-03,  4.86410039e-12,  1.24144088e-04],
       [ 8.31440168e-11,  1.14679314e-14, -1.80350373e-13,
         4.86410039e-12,  2.04958833e-20,  4.56870493e-13],
       [ 6.89892617e-03,  7.42832963e-07, -1.56028162e-05,
         1.24144088e-04,  4.56870493e-13,  4.31155476e-05]])
```

The error in the parameters are obtained from the curvature matrix $N_m = (A' N^{-1} A')^{-1}$.

I defined the routine chi2 to calculate the χ^2 and the noise routine to calculate the shift of the parameters. I used np.random.multivariate_normal and N_m to generate some parameters shifted proportionally to their covariance matrix in the below description of the MCMC routine.

In [59]:

```
def chi2(theta,d):
    yy=ev_planck(theta)
    return (d-yy)@(Ninv@(d-yy))
```

In [60]:

```
def noise(Nm):
    return(np.random.multivariate_normal([0]*len(Nm),Nm))
```

In [61]:

```
noise(Nm)
```

Out[61]:

```
array([-1.47734368e+00, -2.62899969e-05,  3.92482603e-03, -3.92610312e-02,
       -1.47506102e-10, -7.12626926e-03])
```

I defined the routine run_mcmc to run a chain. The routine computes the χ^2 for the starting guess, then shifts the parameters using the noise routine and computes the χ^2 for the new set of parameters. The routine computes a probability $e^{-0.5\Delta\chi^2}$ (with $\Delta\chi^2 = \Delta\chi^2_n - \Delta\chi^2_{n-1}$) and accepts and adds the parameters to the chain just if the probability is greater than a random value between 0 and 1. If the new χ^2 is lower than the old, the 'probability' obtained will be always greater than 1 and the new set of parameters will be added to the chain. If the new set of parameters is not accepted, then the new position in the chain will be equal to the previous one. The routine iterates and compares with the previous step for nsteps and finally returns the parameters in the chain along with the χ^2 for every case. After some trials I ran the MCMC for 25000 steps. Also, an optimal

scale for shifting the parameters and finding an apparent convergence of the chain was noise(Nm) ×[0.8,0.8,0.8,0.45,0.45,0.8]. This was obtained trying 1000 steps multiple times, printing the acceptance ratio in the chain, and looking how the parameters change in every step. An optimal acceptance ratio was about 28%. The starting guess was the optimal values obtained from the Newton's method.

In [31]:

```
def run_mcmc(data,start_pos,nstep,Nm,scale=None):
    nparam=start_pos.size
    params=np.zeros([nstep,nparam+1])
    params[0,0:-1]=start_pos
    cur_chisq=chi2(start_pos,data)
    params[0,-1]=cur_chisq
    cur_pos=start_pos.copy()
    if scale[0]==None:
        scale=np.ones(nparam)
    naccept=0
    accept_rate=np.zeros(nstep)
    for i in range(1,nstep):
        new_pos=cur_pos+noise(Nm)*scale
        new_chisq=chi2(new_pos,data)
        print('Step:',i)
        delt=new_chisq-cur_chisq
        prob=np.exp(-0.5*delt)
        print('prob='+str(prob))
        if np.random.rand()<prob:
                accept=True
        else:
                accept=False
        if accept:
            cur_pos=new_pos
            cur_chisq=new_chisq
            naccept += 1

        accept_rate[i] = naccept / i
        print('acc rate:'+str(accept_rate[i]))
        params[i,0:-1]=cur_pos
        params[i,-1]=cur_chisq
        print(params[i])
        #with open('planck_chain.txt', 'a') as f:
         #   for j in params[i,0:-1]:
          #      f.write(str(j)+' ')
           # f.write(str(params[i,-1])+'\n')
    return params
```

In [101]:

```
parrs=np.array([6.82416236e+01,2.23638984e-02,1.17669875e-01,8.51333156e-02,2.21802047e-09,
```

In [105]:

```
#chain=run_mcmc(spec,parrs,400,Nm,np.array([0.7,0.7,0.7,0.3,0.3,0.7]))#np.array([1.1,1.1,1.
#chain=run_mcmc(spec,parrs,1000,Nm,np.array([0.7,0.7,0.7,0.4,0.4,0.7]))
chaint=run_mcmc(spec,parrs,1000,Nm,np.array([0.8,0.8,0.8,0.45,0.45,0.8]))
```

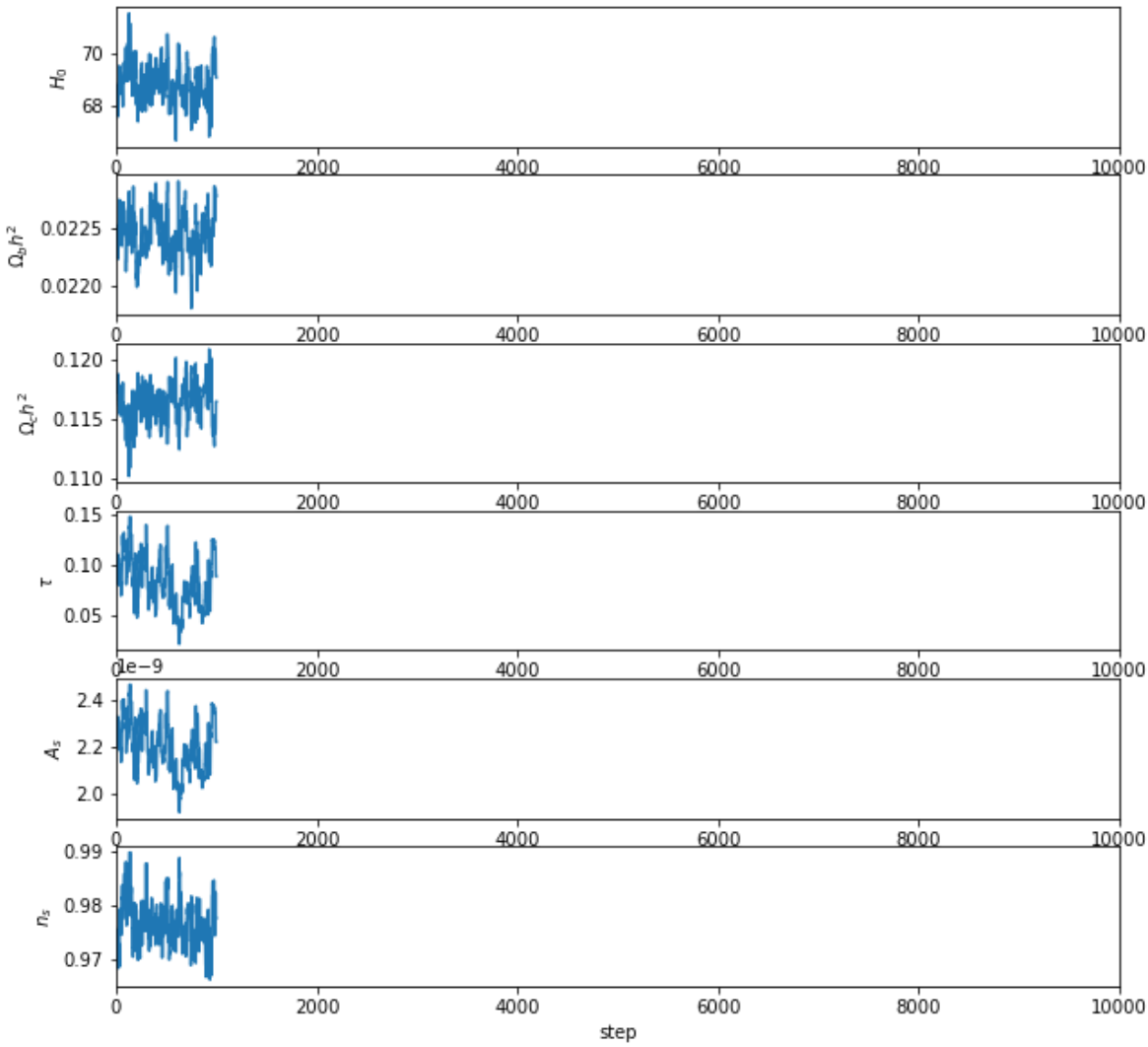
```
[6.77454958e+01 2.24252629e-02 1.18684906e-01 8.67328565e-02
 2.22675690e-09 9.72100455e-01 2.57873155e+03]
Step: 922
prob=1.1216160701772566
acc rate:0.2841648590021692
[6.81466066e+01 2.24171256e-02 1.17798130e-01 1.04431331e-01
 2.30191290e-09 9.75517345e-01 2.57850201e+03]
Step: 923
prob=0.21554228888369706
acc rate:0.28494041170097506
[6.83691988e+01 2.25050668e-02 1.17050966e-01 9.57758700e-02
 2.26187133e-09 9.75587541e-01 2.58157120e+03]
Step: 924
prob=0.4971905327027223
acc rate:0.2857142857142857
[6.77750426e+01 2.22660106e-02 1.18125128e-01 8.43924937e-02
 2.21514671e-09 9.75939688e-01 2.58296877e+03]
Step: 925
prob=0.08028997630927433
acc rate:0.28540540540540543
```

In [109]:

```
f,ax=plt.subplots(6,1,figsize=(10,10))
par=['$H_{0}$','$\Omega_{b}h^{2}$','$\Omega_{c}h^{2}$',r'$\tau$','$A_{s}$','$n_{s}$']
for i in range(0,6):
    ax[i].plot([chaint[j][i] for j in range(0,len(chaint))],'-')
    ax[i].set_ylabel(par[i])
    ax[i].set_xlim(0,10000)
plt.xlabel('step')
```

Out[109]:

```
Text(0.5, 0, 'step')
```



Below I modified the run_mcmc routine to not print every step, but to write the χ^2 value and the parameters of each step on the file planck_chain.txt. Once modified I ran the 25000 steps.

In [46]:

```
def run_mcmc(data,start_pos,nstep,Nm,scale=None):
    nparam=start_pos.size
    params=np.zeros([nstep,nparam+1])
    params[0,0:-1]=start_pos
    cur_chisq=chi2(start_pos,data)
    params[0,-1]=cur_chisq
    cur_pos=start_pos.copy()
    if scale[0]==None:
        scale=np.ones(nparam)
    naccept=0
    accept_rate=np.zeros(nstep)
    for i in range(1,nstep):
        new_pos=cur_pos+noise(Nm)*scale
        new_chisq=chi2(new_pos,data)
        #print('Step:',i)
        delt=new_chisq-cur_chisq
        prob=np.exp(-0.5*delt)
        #print('prob='+str(prob))
        if np.random.rand()<prob:
                accept=True
        else:
                accept=False
        if accept:
            cur_pos=new_pos
            cur_chisq=new_chisq
            naccept += 1

        accept_rate[i] = naccept / i
        #print('acc rate:'+str(accept_rate[i]))
        params[i,0:-1]=cur_pos
        params[i,-1]=cur_chisq
        #print(params[i])
        with open('planck_chain.txt', 'a') as f:
            f.write(str(cur_chisq)+' ')
            for j in cur_pos[0:-1]:
                f.write(str(j)+' ')
            f.write(str(cur_pos[-1])+'\n')
    return params
```

In [47]:

```
chain=run_mcmc(spec,parrs,25000,Nm,np.array([0.8,0.8,0.8,0.45,0.45,0.8]))
```

Reading the file with the chain saved.

In [13]:

```
fl=open('planck_chain.txt')
chain=np.array([(x.strip()).split(' ') for x in fl.readlines()],dtype='float64')
fl.close()
```

In [14]:

```
chi2m=np.array([chain[j][0] for j in range(0,len(chain))])
H0=np.array([chain[j][1] for j in range(0,len(chain))])
Ob=np.array([chain[j][2] for j in range(0,len(chain))])
Oc=np.array([chain[j][3] for j in range(0,len(chain))])
tau=np.array([chain[j][4] for j in range(0,len(chain))])
As=np.array([chain[j][5] for j in range(0,len(chain))])
ns=np.array([chain[j][6] for j in range(0,len(chain))])
```

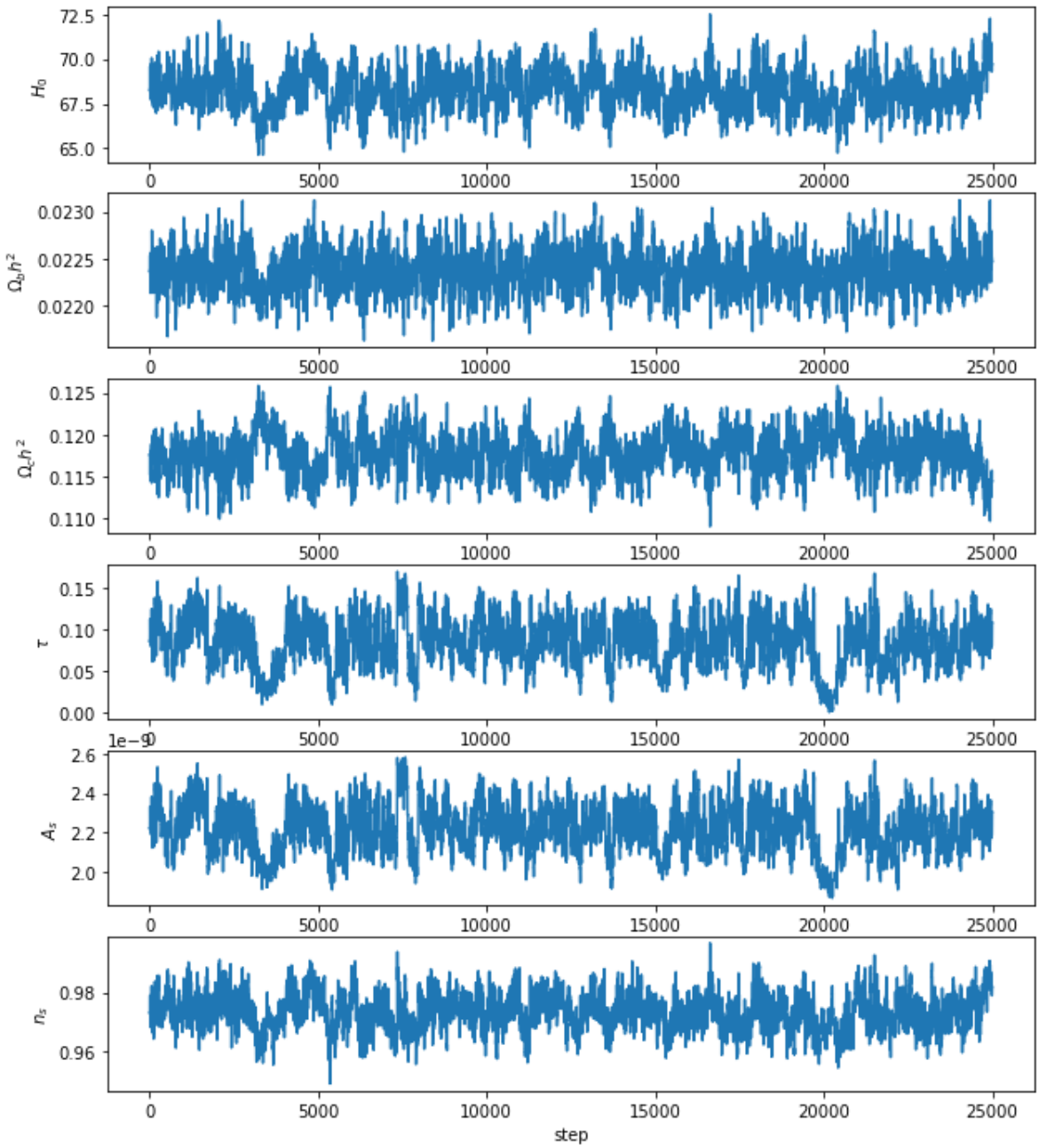
Below we can see that the plots of parameter vs steps almost look like white noise, this means that the parameters have forgotten their initial values, and then the chain should have converged.

In [15]:

```
f,ax=plt.subplots(6,1,figsize=(10,12))
par=[r'$H_{0}$',r'$\Omega_{b}h^{2}$',r'$\Omega_{c}h^{2}$',r'$\tau$',r'$A_{s}$',r'$n_{s}$']
for i in range(1,7):
    ax[i-1].plot([chain[j][i] for j in range(0,len(chain))],'-')
    ax[i-1].set_ylabel(par[i-1])
    #ax[i-1].set_xlim(0,10000)
plt.xlabel('step')
```

Out[15]:

```
Text(0.5, 0, 'step')
```



Now, using the corner package we can plot the posterior distributions for the parameters along with the scatter of the parameters. Thus, we can also see the correlation between the parameters.

In [16]:

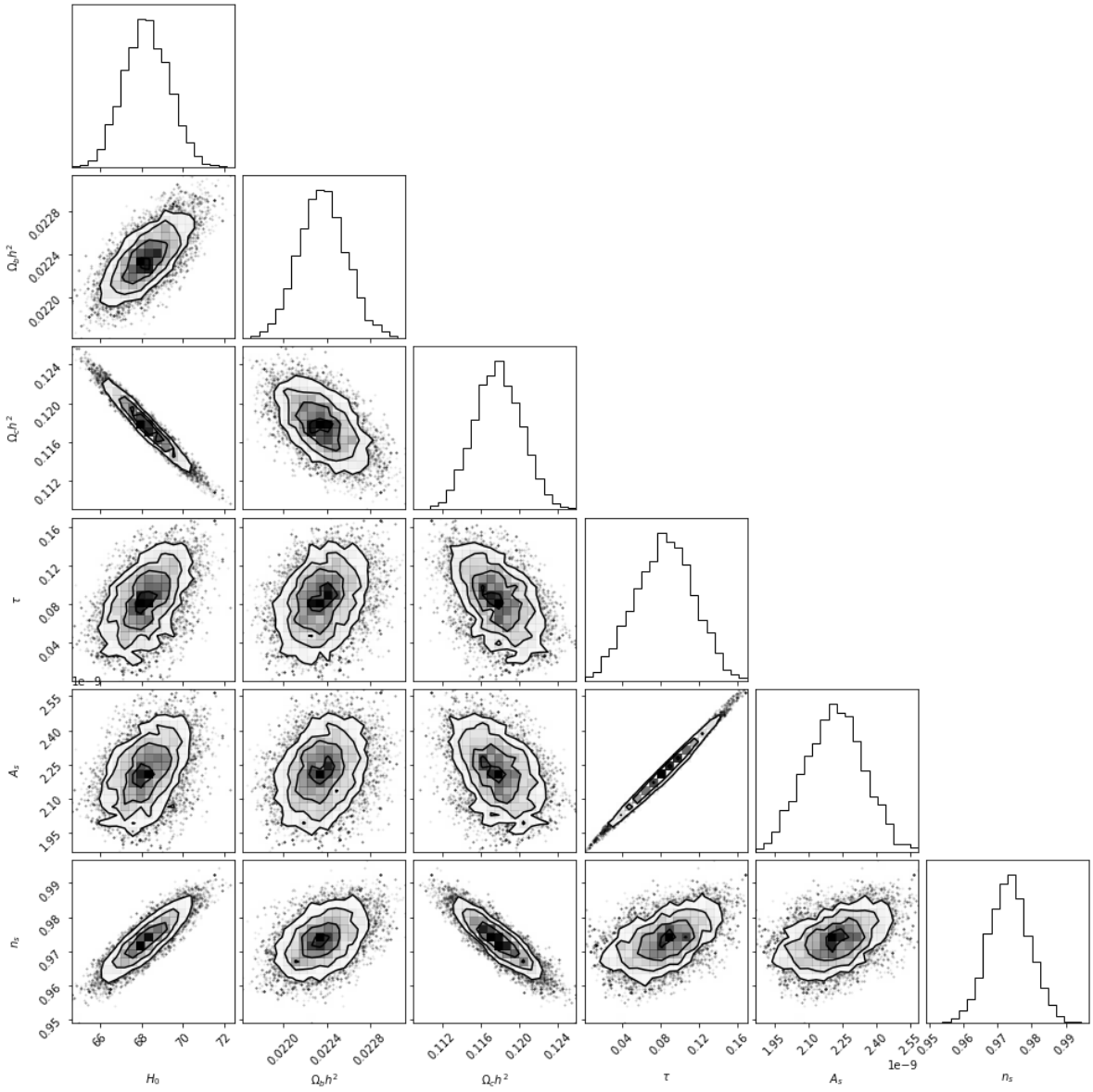
```
samples=np.array([H0,Ob,Oc,tau,As,ns])
```

In [17]:

```
import corner
```

In [18]:

```
figure = corner.corner(samples.T,labels=par)
```



And the estimation of the parameters with their uncertainties are the following:

In [121]:

```
parr=[r'H_0',r'Omega_b h^2',r'Omega_c h^2',r'tau',r'A_s',r'n_s']
for i in range(len(samples)):
    print(r'The estimation for '+parr[i]+'='+str(np.mean(samples[i]))+
          r' with error:'+str(np.std(samples[i])))
```

```
The estimation for H_0=68.23136560956435 with error:1.0973647667228792
The estimation for Omega_b h^2=0.02235838108619997 with error:0.000219506716
08967346
The estimation for Omega_c h^2=0.1177340217127854 with error:0.0024278126238
312155
The estimation for tau=0.08356992190162157 with error:0.02998685615091737
The estimation for A_s=2.2145843746879562e-09 with error:1.2611974772802786e
-10
The estimation for n_s=0.9734267864131818 with error:0.005991863070678401
```

Using the formulation for a spatially flat universe, we will have that $\Omega_\Lambda = 1 - (\Omega_c h^2 + \Omega_b h^2)\frac{100^2}{H_0^2}$. And the error can be estimated using the partial derivatives of Ω_Λ respect to the parameters obtained from the MCMC $\Delta\Omega_\Lambda = \sqrt{(\frac{\partial\Omega_\Lambda}{\partial\Omega_c h^2})^2(\Delta\Omega_c h^2)^2 + (\frac{\partial\Omega_\Lambda}{\partial\Omega_b h^2})^2(\Delta\Omega_b h^2)^2 + (\frac{\partial\Omega_\Lambda}{\partial H_0})^2(\Delta H_0)^2}$ (considering $\Delta m = \sigma_m$ for each parameter).

In [70]:

```
Ol=1-(np.mean(Ob)+np.mean(Oc))*(100**2)/(np.mean(H0)**2)
```

In [71]:

```
sOl=(100**2)*np.sqrt(((np.std(Ob)**2)+(np.std(Oc)**2))/(np.mean(H0)**4)+
                ((np.std(H0)**2)*2*((np.mean(Ob)+np.mean(Oc))**2))/(np.mean(H0)**6))
```

In [72]:

```
Ol,sOl,Ol+sOl
```

Out[72]:

```
(0.6990831845226845, 0.008617536185061251, 0.7077007207077457)
```

The estimation for the dark energy was $\Omega_\Lambda = 0.699 \pm 0.0086$

Now, we can define weights to do importance sampling using the prior distribution for the optical depth $\tau = 0.0540 \pm 0.0074$. If we consider that the posterior distributions obtained using MCMC are Gaussians and the new prior is also a Gaussian (G), we can define the weigths:

$$w_i = \frac{G(\tau_i|0.0540, 0.0074)}{G(\tau_i|\bar{\tau}_{MCMC}, \sigma_{\tau MCMC})}$$

. Then, the estimation of each parameter in the chain using the importance sampling with the weights is $< m >= \frac{\sum_i w_i m_i}{\sum_i w_i}$. Also $< m^2 >= \frac{\sum_i w_i m_i^2}{\sum_i w_i}$, and then, $\sigma_m = \sqrt{< m^2 > - < m >^2}$.

In [73]:

```
ptau_old=np.exp(-((tau-np.mean(tau))/np.std(tau))**2)
```

In [74]:

```
ptau_new=np.exp(-((tau-0.0540)/0.0074)**2)
```

In [80]:

```
wi=ptau_new/ptau_old
```

In [87]:

```
means=(samples@wi)/sum(wi)
```

In [88]:

```
meanssq=((samples**2)@wi)/sum(wi)
```

In [89]:

```
errsi=np.sqrt(meanssq-means**2)
```

The estimation of the parameters using importance sampling are the following:

In [122]:

```
for i in range(len(samples)):
    print(r'The estimation for '+parr[i]+'='+str(means[i])+
          r' with error:'+str(errsi[i]))
```

```
The estimation for H_0=67.73133129924331 with error:1.0520689969219863
The estimation for Omega_b h^2=0.022288421089291775 with error:0.00021299605
62668949
The estimation for Omega_c h^2=0.11879595144153415 with error:0.002379746856
3476515
The estimation for tau=0.05288371159474349 with error:0.005340468268458898
The estimation for A_s=2.0856005001653924e-09 with error:2.416685352403217e-
11
The estimation for n_s=0.9705314751136545 with error:0.005791619596947485
```

We can restimate the covariance matrix using $N_m = (A'(< m >)N^{-1}A'(< m >))^{-1}$ for $< m >$ from the importance sampling that I just did.

In [95]:

```
y,der=deriv(ev_planck,means,hh)
A_m=np.ones([y.size,6])
for i in range(6):
    A_m[:,i]=der[i]
```

In [97]:

```
Nmm=np.linalg.inv(A_m.T@Ninv@A_m)
```

In [98]:

```
Nmm
```

Out[98]:

```
array([[ 1.37789509e+00,  1.92985369e-04, -3.03110424e-03,
         2.33610676e-02,  8.23786070e-11,  6.53112379e-03],
       [ 1.92985369e-04,  5.28631944e-08, -3.58200828e-07,
         3.40690959e-06,  1.25621624e-14,  7.18407451e-07],
       [-3.03110424e-03, -3.58200828e-07,  7.00491063e-06,
        -5.11083453e-05, -1.77648786e-13, -1.49137543e-05],
       [ 2.33610676e-02,  3.40690959e-06, -5.11083453e-05,
         1.36728653e-03,  5.43008831e-12,  1.16804645e-04],
       [ 8.23786070e-11,  1.25621624e-14, -1.77648786e-13,
         5.43008831e-12,  2.17085416e-20,  4.04178701e-13],
       [ 6.53112379e-03,  7.18407451e-07, -1.49137543e-05,
         1.16804645e-04,  4.04178701e-13,  4.00938917e-05]])
```

Below I defined a routine named prior to evaluate the trial step value of τ in the Gaussian distribution $G(\tau_i|0.0540, 0.0074)$

In [38]:

```
def prior(theta):
        return np.exp(-((theta[3]-0.0540)/0.0074)**2)
```

I redefined the routine to run a chain as run_mcmc_p, this routine includes the prior information defined in the routine prior. Thus, the new probability in each step k is now $e^{-0.5\Delta\chi^2}\frac{prior(t_k)}{prior(t_{k-1})}$.

In [99]:

```
def run_mcmc_p(data,start_pos,nstep,Nm,scale=None):
    nparam=start_pos.size
    params=np.zeros([nstep,nparam+1])
    params[0,0:-1]=start_pos
    cur_chisq=chi2(start_pos,data)
    params[0,-1]=cur_chisq
    cur_pos=start_pos.copy()
    if scale[0]==None:
        scale=np.ones(nparam)
    naccept=0
    accept_rate=np.zeros(nstep)
    for i in range(1,nstep):
        new_pos=cur_pos+noise(Nm)*scale
        new_chisq=chi2(new_pos,data)
        #print('Step:',i)
        delt=new_chisq-cur_chisq
        prob=prior(new_pos)*np.exp(-0.5*delt)/prior(cur_pos)
        #print('prob='+str(prob))
        if np.random.rand()<prob:
                accept=True
        else:
                accept=False
        if accept:
            cur_pos=new_pos
            cur_chisq=new_chisq
            naccept += 1

        accept_rate[i] = naccept / i
        #print('acc rate:'+str(accept_rate[i]))
        params[i,0:-1]=cur_pos
        params[i,-1]=cur_chisq
        #print(params[i])
        with open('planck_chain_tauprior.txt', 'a') as f:
            f.write(str(cur_chisq)+' ')
            for j in cur_pos[0:-1]:
                f.write(str(j)+' ')
            f.write(str(cur_pos[-1])+'\n')
    return params
```

In [102]:

```
chain1=run_mcmc_p(spec,parrs,25000,Nmm,np.array([0.7,0.7,0.7,0.4,0.4,0.7]))
```

In [103]:

```
fl1=open('planck_chain_tauprior.txt')
chain1=np.array([(x.strip()).split(' ') for x in fl1.readlines()],dtype='float64')
fl1.close()
```

In [104]:

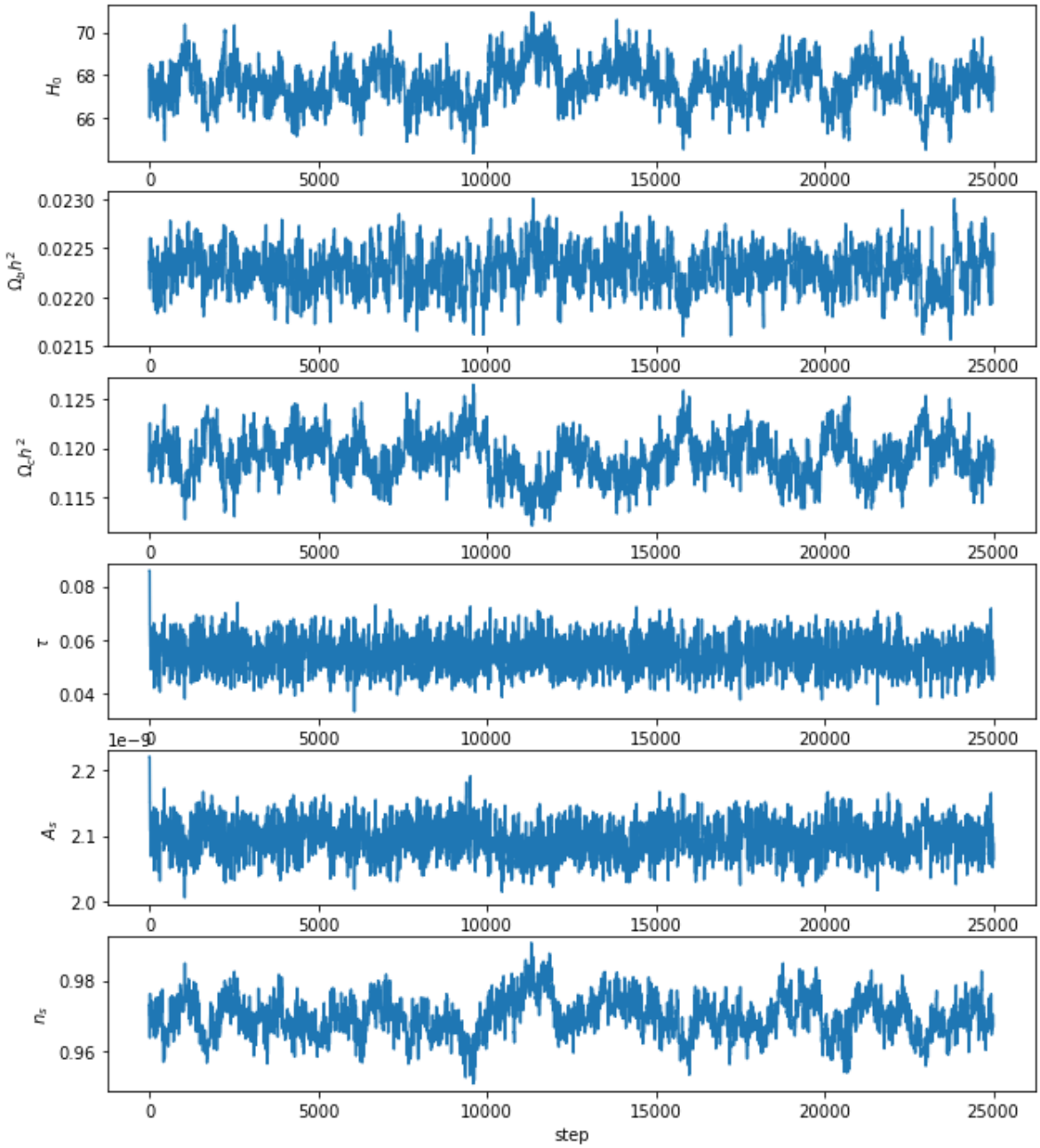
```
chi21=np.array([chain1[j][0] for j in range(0,len(chain1))])
H01=np.array([chain1[j][1] for j in range(0,len(chain1))])
Ob1=np.array([chain1[j][2] for j in range(0,len(chain1))])
Oc1=np.array([chain1[j][3] for j in range(0,len(chain1))])
tau1=np.array([chain1[j][4] for j in range(0,len(chain1))])
As1=np.array([chain1[j][5] for j in range(0,len(chain1))])
ns1=np.array([chain1[j][6] for j in range(0,len(chain1))])
```

In [107]:

```
f,ax=plt.subplots(6,1,figsize=(10,12))
par=['$H_{0}$','$\Omega_{b}h^{2}$','$\Omega_{c}h^{2}$',r'$\tau$','$A_{s}$','$n_{s}$']
for i in range(1,7):
    ax[i-1].plot([chain1[j][i] for j in range(0,len(chain1))],'-')
    ax[i-1].set_ylabel(par[i-1])
    #ax[i-1].set_xlim(0,25000)
plt.xlabel('step')
```

Out[107]:

```
Text(0.5, 0, 'step')
```

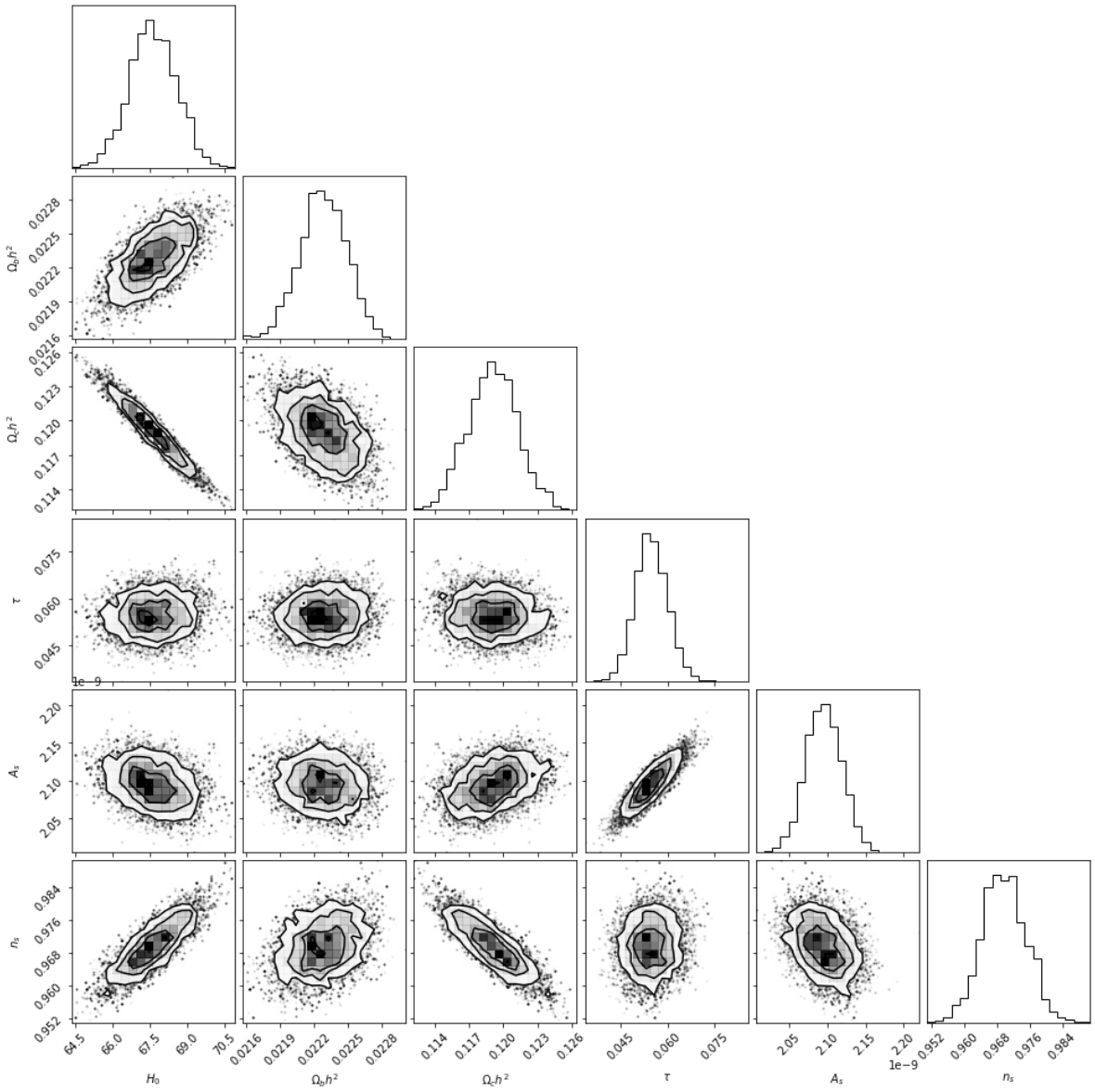


In [108]:

```
samples1=np.array([H01,Ob1,Oc1,tau1,As1,ns1])
```

In [109]:

```
figure1 = corner.corner(samples1.T,labels=par)
```



The new set of parameters using the prior information is:

In [118]:

```
for i in range(len(samples1)):
    print(r'The estimation for '+parr[i]+'='+str(np.mean(samples1[i]))+
          r' with error:'+str(np.std(samples1[i])))
```

```
The estimation for H_0=67.61572079584533 with error:0.9543111784957445
The estimation for Omega_b h^2=0.022275331251134352 with error:0.00020756084
149584837
The estimation for Omega_c h^2=0.11905167357470824 with error:0.002189197371
554165
The estimation for tau=0.05471969559559261 with error:0.00510230826913746
The estimation for A_s=2.094890407757391e-09 with error:2.352825656303212e-1
1
The estimation for n_s=0.9696847297738862 with error:0.0054369523058007485
```

The absolute differences from what we obtained using importance sampling are:

In [119]:

```
for i in range(len(samples1)):
    print(r'The absolute difference for '+parr[i]+'='+
          str(np.abs(np.mean(samples1[i])-means[i])))
```

```
The absolute difference for H_0=0.11561050339797418
The absolute difference for Omega_b h^2=1.308983815742279e-05
The absolute difference for Omega_c h^2=0.00025572213317408277
The absolute difference for tau=0.0018359840008491182
The absolute difference for A_s=9.289907591998539e-12
The absolute difference for n_s=0.0008467453397682512
```

We can see that the differences from what we obtained using the prior information for MCMC and the importance sampling are of the order of magnitude $< \sigma_m$ (less than the estimated error) for each parameter m. Therefore, if we already ran our chain but we know new prior information, using importance sampling could be a good approximation of what we will be obtaining if we run a new chain including the prior information.

We can compare what we obtained between the two chains to see that this difference is larger than the difference of comparing the estimations from the chain with prior information and the estimations from the importance sampling.

In [120]:

```
for i in range(len(samples1)):
    print(r'The absolute difference for '+parr[i]+'='+
          str(np.abs(np.mean(samples1[i])-np.abs(np.mean(samples[i])))))
```

```
The absolute difference for H_0=0.6156448137190154
The absolute difference for Omega_b h^2=8.304983506561905e-05
The absolute difference for Omega_c h^2=0.0013176518619228417
The absolute difference for tau=0.028850226306028966
The absolute difference for A_s=1.1969396693056532e-10
The absolute difference for n_s=0.00374205663929561
```