Assignment 5

1) The initial parameters gave a χ^2 of 15267 for 2501 degrees of freedom. The χ^2 should approximate the number of degrees of freedom. It is not a good fit.

Using the new params, we get a χ^2 of 3272 for 2501 degrees of freedom. It is a much beter fit. (pset5_1.py)

2) Using Newton's method we got: (pset5_2.py)

$$H_0 = 67.34 \pm 3.93 \times 10^{-5}$$

$$\Omega_b h^2 = 2.25 \pm 1.14 \times 10^{-8}$$

$$\Omega_c h^2 = 1.19 \times 10^{-1} \pm 2.60 \times 10^{-8}$$

$$\tau = 1.27 \times 10^{-1} \pm 2.60 \times 10^{-8}$$

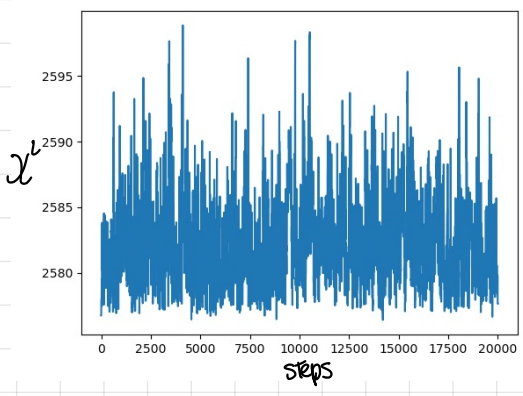
$$A_s = 2.43 \times 10^{-9} \pm 2.04 \times 10^{-10}$$

$$n_s = 9.72 \times 10^{-1} \pm 2.76 \times 10^{-8}$$

With a χ^2 of 2576

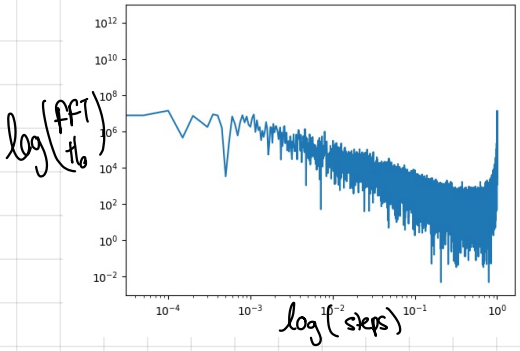
3) We compute the MCMC chain in pset5_3.py

We get a chain for χ^2:

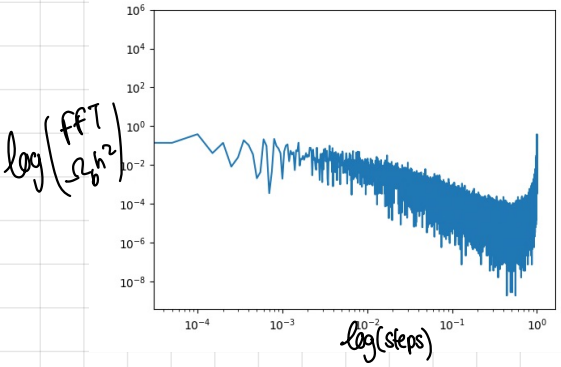


A chain on which we computed the fast fourier transform for :

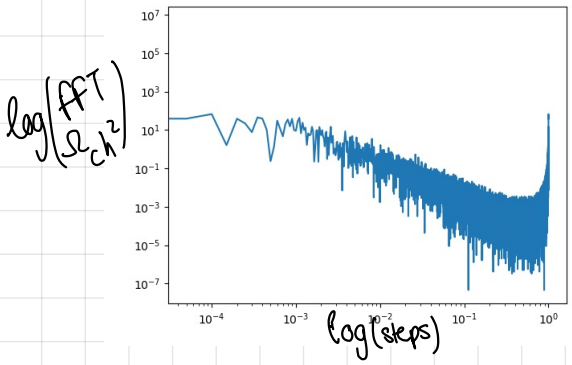
H_0 :



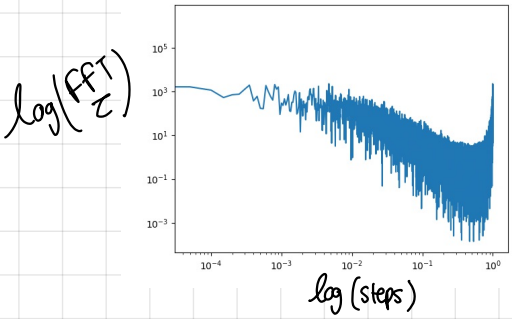
$\Omega_b h^2$:



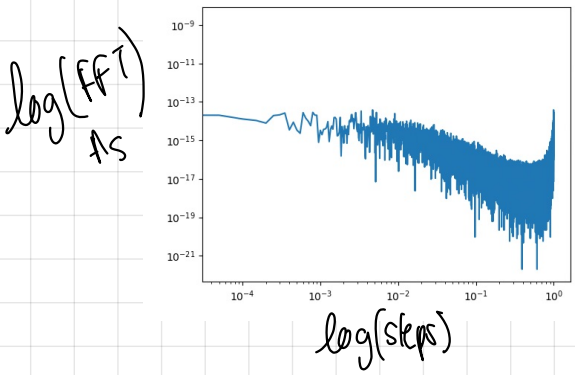
$\Omega_c h^2$:



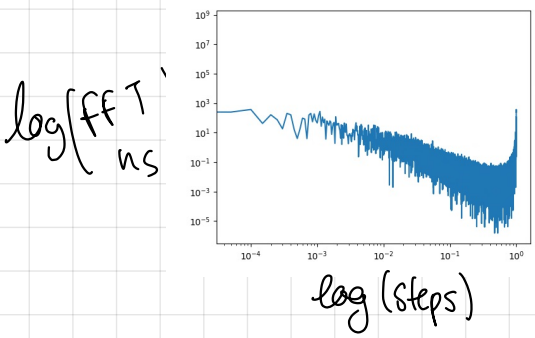
τ :



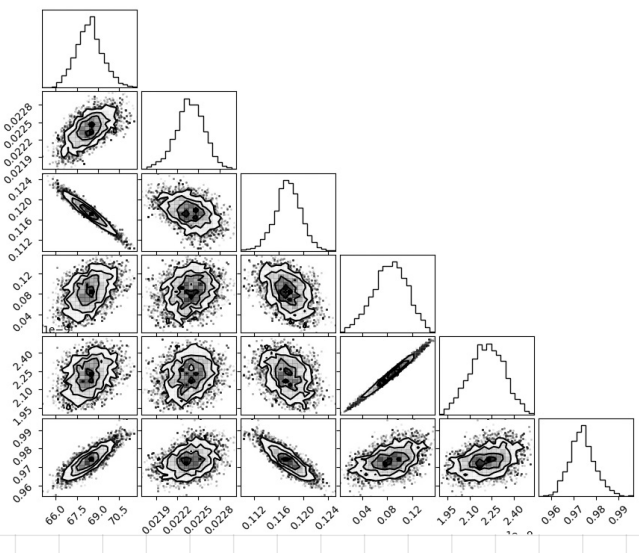
A_S :



n_s



we computed a corner plot based on the chain



Those graphs show a constant behavior (about the same distance from the mean) furthermore, the corner plot shows a rather correlated data.

We use as a tool for convergence test, the gelman rubin scatters method. We get a really low result for all data which proves it converged.

4) In this last question, we compared, to our 3) chain that we cleaned using the importance sampling method, a chain with a fixed tau. We use the process chain method to compare both chains

We get

3 4

- fractions of samples on param 0 more than 5 is 0.0 0.0005
- fractions of samples on param 1 more than 5 is 0.0 0.0
- fractions of samples on param 2 more than 5 is 0.0 0.0
- fractions of samples on param 3 more than 5 is 0.0 0.0
- fractions of samples on param 4 more than 5 is 0.0002 0.8224
- fractions of samples on param 5 more than 5 is 0.0 0.00165

This shows that fixing the tau gave a much better convergence. The percentage of data greater than 5σ is greater for our previous chain. These percentages are our error bars 5σ equivalent. furthermore, during chain 3's run an error would often come up. It would say:

Warning: xe at redshift zero is < 1
Check input parameters an Reionization_xe function in the Reionization module

This suggest that this fixed tau gave a much better constraint Reionization ensuring a beter chain.

Assignment5

October 29, 2022

Question 1

[ ]:
```
#Code computing the change of pars manually from [60,0.02,0.1,0.05,2.00e-9,1.0]␣
↪and [69, 0.022, 0.12,0.06, 2.1e-9, 0.95]

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

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]) #test chisq is  15267.
↪937968194292  for  2501  degrees of freedom.
pars=np.asarray([69, 0.022, 0.12,0.06, 2.1e-9, 0.95]) #q1 chisq is  3272.
↪203604462886  for  2501  degrees of freedom.
planck=np.loadtxt('Assignment5/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('Assignment5/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()
```

Question 2

[ ]:
```
#Code to compute newton method

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

newtom_params_txt_file = "planck_fit_params.txt"

def get_spectrum(pars,lmax=3000):
    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]
    return tt[2:]
def num_derivs(fun,pars,dp,x):
    A=np.empty([len(x),len(pars)])
    for i in range(len(pars)):
        pp=pars.copy()
        pp[i]=pars[i]+dp[i]
```

```
        y_right=fun(pp)
        y_right=y_right[:len(spec)]
        pp[i]=pars[i]-dp[i]
        y_left=fun(pp)
        y_left=y_left[:len(spec)]
        A[:,i]=(y_right-y_left)/(2*dp[i])
    return A
def newton(fun,pars,dp,x,y,niter=3):
    errs=0.5*(planck[:,2]+planck[:,3])
    err_diag = np.asarray(errs**2)
    N = np.zeros((len(x),len(x)))
    np.fill_diagonal(N, err_diag)
    for i in range(niter):
        pred=fun(pars)
        pred=pred[:len(spec)]
        r=y-pred
        A=num_derivs(fun,pars,dp,x)
        lhs=A.T@np.linalg.inv(N)@A
        rhs=A.T@np.linalg.inv(N)@r
        step=np.linalg.inv(lhs)@rhs
        pars=pars+step
    return pars,np.linalg.inv(lhs)
#Initial pars
pars=np.asarray([69, 0.022, 0.12,0.06, 2.1e-9, 0.95])

#DATA
planck=np.loadtxt('Assignment5/COM_PowerSpect_CMB-TT-full_R3.01.txt',skiprows=1)
ell=planck[:,0] #x
spec=planck[:,1] #y

#Setup
dx = 1e-8
dp = pars*dx
fitp,curve=newton(get_spectrum,pars,dp,ell,spec)

#Errors on param
error_fitp=np.diag(np.linalg.cholesky(curve))
errs=0.5*(planck[:,2]+planck[:,3])
model=get_spectrum(fitp)
model=model[:len(spec)]
resid=spec-model
chisq=np.sum((resid/errs)**2)
print("chsiq is " + str(chisq))

#Document editing planck_fit_params.txt
f = open("Assignment5/planck_fit_params.txt", "a")
line = "Hubble constant is " + str(fitp[0]) + " ± " + str(error_fitp[0]) + "\n"
```

```
f.write(line)
line = "Baryon density is " + str(fitp[1]) + " ± " + str(error_fitp[1]) +"\n"
f.write(line)
line = "Dark matter density is " + str(fitp[2]) + " ± " + str(error_fitp[2])␣
↪+"\n"
f.write(line)
line = "Optical depth is " + str(fitp[3]) + " ± " + str(error_fitp[2])+ "\n"
f.write(line)
line = "Primordial amplitude of the spectrum is " + str(fitp[4]) + " ± " +␣
↪str(error_fitp[4])+ "\n"
f.write(line)
line = "Primordial tilt of the spectrum is " + str(fitp[5]) + " ± " +␣
↪str(error_fitp[5])
f.write(line)
```

planck_fit_params.txt * Hubble constant is 67.34446649082994 ± 3.929618194466233e-05 * Baryon density is 0.0224748518522885 ± 1.1372956877108872e-08 * Dark matter density is 0.11938840491314752 ± 2.604464866254484e-08 * Optical depth is 0.12650918521699278 ± 2.604464866254484e-08 * Primordial amplitude of the spectrum is 2.4307447362719345e-09 ± 2.0408733057146e-16 * Primordial tilt of the spectrum is 0.972377519638394 ± 2.7563004980729678e-08

Question 3:

[ ]:
```
#Code to compute MCMC

import numpy as np
from matplotlib import pyplot as plt
import camb
import corner

def get_spectrum(pars,lmax=3000):
    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]
    return tt[2:]
def num_derivs(fun,pars,dp,x):
```

```
    A=np.empty([len(x),len(pars)])
    for i in range(len(pars)):
        pp=pars.copy()
        pp[i]=pars[i]+dp[i]
        y_right=fun(pp)
        y_right=y_right[:len(spec)]
        pp[i]=pars[i]-dp[i]
        y_left=fun(pp)
        y_left=y_left[:len(spec)]
        A[:,i]=(y_right-y_left)/(2*dp[i])
    return A
def newton(fun,pars,dp,x,y,niter=3):
    errs=0.5*(planck[:,2]+planck[:,3])
    err_diag = np.asarray(errs**2)
    N = np.zeros((len(x),len(x)))
    np.fill_diagonal(N, err_diag)
    for i in range(niter):
        pred=fun(pars)
        pred=pred[:len(spec)]
        r=y-pred
        A=num_derivs(fun,pars,dp,x)
        lhs=A.T@np.linalg.inv(N)@A
        rhs=A.T@np.linalg.inv(N)@r
        step=np.linalg.inv(lhs)@rhs
        pars=pars+step
    return pars,np.linalg.inv(lhs)
def run_chain(fun,pars,trial_step,data,nstep=20000,T=1):
    npar=len(pars)
    chain=np.zeros([nstep,npar])
    chisq=np.zeros(nstep)
    chain[0,:]=pars
    chi_cur=fun(pars,data)
    chisq[0]=chi_cur
    for i in range(1,nstep):
        pp=pars+get_step(trial_step)
        new_chisq=fun(pp,data)
        accept_prob=np.exp(-0.5*(new_chisq-chi_cur)/T)
        if np.random.rand(1)<accept_prob:
            pars=pp
            chi_cur=new_chisq
        chain[i,:]=pars
        chisq[i]=chi_cur
    return chain,chisq
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])
def process_chain(chain,chisq,T=1.0):
    dchi=chisq-np.min(chisq)
    #density in chain is exp(-0.5*chi^2/T), but
    #we wanted it to be exp(-0.5*chi^2)
    #so, we want to downweight by ratio, which is
    #exp(-0.5*chi^2*(1-1/T)).  We'll calculate the mean
    #and standard deviation of the chain, but will also
    #return the weights so you could calculate whatever you want

    wt=np.exp(-0.5*dchi*(1-1/T)) #the magic line that importance samples

    #calculate the weighted sum of the chain and the chain squared
    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
def spectrum_chisq(pars,data):
    x=data['x']
    y=data['y']
    errs=data['errs']

    model=get_spectrum(pars)
    model=model[:len(spec)]
    resid=spec-model
    chisq=np.sum((resid/errs)**2)
    return chisq

#Initial pars
pars=np.asarray([69, 0.022, 0.12,0.06, 2.1e-9, 0.95])

#DATA
planck=np.loadtxt('Assignment5/COM_PowerSpect_CMB-TT-full_R3.01.txt',skiprows=1)
ell=planck[:,0] #x
spec=planck[:,1] #y

#Setup
```

```
dx = 1e-8
dp = pars*dx
fitp,curve=newton(get_spectrum,pars,dp,ell,spec)

#Errors on pars
error_fitp=np.diag(np.linalg.cholesky(curve))
errs=0.5*(planck[:,2]+planck[:,3])
model=get_spectrum(fitp)
model=model[:len(spec)]
resid=spec-model
chisq=np.sum((resid/errs)**2)

data={}
data['x']=ell
data['y']=spec
data['errs']=errs

#Run MCMC chain
chain,chivec=run_chain(spectrum_chisq,fitp,curve,data)
steps = np.linspace(0,20000, 20000)
f = open("Assignment5/chain_4.txt", "a")
for line in chain:
    f.write(str(line) + "\n")
f.close()
f = open("Assignment5/chain_chi_4.txt", "a")
for line in chivec:
    f.write(str(line)+ "\n")
f.close()
f = open("Assignment5/pars_chain_4.txt", "a")
f.write(str(pars))
f.close()
f = open("Assignment5/errs_chain_4.txt", "a")
f.write(str(errs))
f.close()

#Chi v step graph showing the chain
plt.plot(steps, chivec)

#Corner of the distribution
corner.corner(chain)

#Log(FFT) plot
for i in range (len(chain.T)):
    plt.loglog(np.linspace(0,1,len(chain.T[i])),np.abs(np.fft.fft(chain.
 ↪T[i]))**2)
plt.show()
```

Output analysis

[ ]:
```
#Code that takes a bad chain format and formats it to the expected output
#It also performs the plot chain[i] v steps for each params

import numpy as np
import matplotlib.pyplot as plt

f = open("Assignment5/chain_3.txt", "r")
chain = []
lines = f.readlines()
d = open("Assignment5/chain_chi_3.txt", "r")
lines_d = d.readlines()
chain_chi = []

#FORMATTING
for i in range(len(lines_d)):
    chain_chi.append(float(lines_d[i]))

count_chain = 0
for r in range(0,len(lines),2):
    line1 = lines[r][1:].split(" ")
    line2 = lines[r+1][1:-2].split(" ")
    line1.append(line2[0])
    line1.append(line2[1])
    line = []
    line.append(chain_chi[count_chain])
    count_chain = count_chain + 1
    for j in line1:
        line = np.append(line,float(j))
    chain.append(np.array(line))
chain = np.array(chain)
steps = np.linspace(0,20000, 20000)


re = open("Assignment5/planck chain.txt", "a")

#FILE FILLING
for i in range(len(chain)):
    line = ""
    for j in range(len(chain[i])):
        line = line + " " + str(chain[i][j]) + " "
    line = line + str("\n")
    re.write(line)

#PLOTS
for i in range(len(chain.T)):
    plt.plot(steps, chain.T[i])
```

Question 4:

[ ]:
```
#Code to MCMC with fixed tau
import numpy as np
from matplotlib import pyplot as plt
import camb
import corner


def get_spectrum(pars,lmax=3000):
    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]
    return tt[2:]
def num_derivs(fun,pars,dp,x):
    A=np.empty([len(x),len(pars)])
    for i in range(len(pars)):
        pp=pars.copy()
        pp[i]=pars[i]+dp[i]
        y_right=fun(pp)
        y_right=y_right[:len(spec)]
        pp[i]=pars[i]-dp[i]
        y_left=fun(pp)
        y_left=y_left[:len(spec)]
        A[:,i]=(y_right-y_left)/(2*dp[i])
    return A
def newton(fun,pars,dp,x,y,niter=3):
    errs=0.5*(planck[:,2]+planck[:,3])
    err_diag = np.asarray(errs**2)
    N = np.zeros((len(x),len(x)))
    np.fill_diagonal(N, err_diag)
    for i in range(niter):
        pred=fun(pars)
        pred=pred[:len(spec)]
        r=y-pred
        A=num_derivs(fun,pars,dp,x)
        lhs=A.T@np.linalg.inv(N)@A
        rhs=A.T@np.linalg.inv(N)@r
```

```
        step=np.linalg.inv(lhs)@rhs
        pars=pars+step
    return pars,np.linalg.inv(lhs)
def run_chain(fun,pars,trial_step,data,nstep=20000,T=1):
    npar=len(pars)
    chain=np.zeros([nstep,npar])
    chisq=np.zeros(nstep)
    chain[0,:]=pars
    chi_cur=fun(pars,data)
    chisq[0]=chi_cur
    for i in range(1,nstep):
        pp = []
        for j in range(0,3):
            pp.append(pars[j]+get_step(trial_step)[j])
        pp.append(pars[3])
        for j in range(4,len(pars)):
            pp.append(pars[j]+get_step(trial_step)[j])
        new_chisq=fun(pp,data)
        new_chisq=fun(pp,data)
        accept_prob=np.exp(-0.5*(new_chisq-chi_cur)/T)
        if np.random.rand(1)<accept_prob:
            pars=pp
            chi_cur=new_chisq
        chain[i,:]=pars
        chisq[i]=chi_cur
    return chain,chisq
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])
def process_chain(chain,chisq,T=1.0):
    dchi=chisq-np.min(chisq)
    #density in chain is exp(-0.5*chi^2/T), but
    #we wanted it to be exp(-0.5*chi^2)
    #so, we want to downweight by ratio, which is
    #exp(-0.5*chi^2*(1-1/T)).  We'll calculate the mean
    #and standard deviation of the chain, but will also
    #return the weights so you could calculate whatever you want

    wt=np.exp(-0.5*dchi*(1-1/T)) #the magic line that importance samples

    #calculate the weighted sum of the chain and the chain squared
    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
def spectrum_chisq(pars,data):
    x=data['x']
    y=data['y']
    errs=data['errs']

    model=get_spectrum(pars)
    model=model[:len(spec)]
    resid=spec-model
    chisq=np.sum((resid/errs)**2)
    return chisq

#Initial pars
pars=np.asarray([69, 0.022, 0.12,0.06, 2.1e-9, 0.95])

#DATA
planck=np.loadtxt('Assignment5/COM_PowerSpect_CMB-TT-full_R3.01.txt',skiprows=1)
ell=planck[:,0] #x
spec=planck[:,1] #y

#Setup
dx = 1e-8
dp = pars*dx
fitp,curve=newton(get_spectrum,pars,dp,ell,spec)

error_fitp=np.diag(np.linalg.cholesky(curve))
errs=0.5*(planck[:,2]+planck[:,3])
fitp[3] = pars[3]
model=get_spectrum(fitp)
model=model[:len(spec)]
resid=spec-model
chisq=np.sum((resid/errs)**2)
print("chsiq is " + str(chisq))


data={}
data['x']=ell
data['y']=spec
```

```
data['errs']=errs

chain,chivec=run_chain(spectrum_chisq,fitp,curve,data)
print(errs)
steps = np.linspace(0,20000, 20000)
f = open("Assignment5/fixed_chain_4.txt", "a")
for line in chain:
    f.write(str(line) + "\n")
f.close()
f = open("Assignment5/fixed_chain_chi_4.txt", "a")
for line in chivec:
    f.write(str(line)+ "\n")
f.close()
f = open("Assignment5/fixed_pars_chain_4.txt", "a")
f.write(str(pars))
f.close()
f = open("Assignment5/fixed_errs_chain_4.txt", "a")
f.write(str(errs))
f.close()

#Chi v step graph showing the chain
plt.plot(steps, chivec)

#Corner of the distribution
corner.corner(chain)

#Log(FFT) plot
for i in range (len(chain.T)):
    plt.loglog(np.linspace(0,1,len(chain.T[i])),np.abs(np.fft.fft(chain.
 ↪T[i]))**2)
plt.show()
```

Output analysis

[ ]:
```
#Code to format correctly the pars data
##it also compares the chain we had in 3 to the chain 4
import numpy as np
import matplotlib.pyplot as plt

f1 = open("Assignment5/OUTPUT_FIXED_CHAIN_4/fixed_chain_4.txt", "r")
f2 = open("Assignment5/OUTPUT_CHAIN_3/chain_3.txt", "r")
chain1 = []
lines_f1 = f1.readlines()
chain2 = []
lines_f2 = f2.readlines()
d1 = open("Assignment5/OUTPUT_FIXED_CHAIN_4/fixed_chain_chi_4.txt", "r")
lines_d1 = d1.readlines()
```

```
chain_chi1 = []
d2 = open("Assignment5/OUTPUT_CHAIN_3/chain_chi_3.txt", "r")
lines_d2 = d2.readlines()
chain_chi2 = []

for i in range(len(lines_d1)):
    chain_chi1.append(float(lines_d1[i]))
for i in range(len(lines_d2)):
    chain_chi2.append(float(lines_d2[i]))

def chain_creat(lines_f1,chain1):
    count_chain = 0
    for r in range(0,len(lines_f1),2):
        line1 = lines_f1[r][1:].split(" ")
        line2 = lines_f1[r+1][1:-2].split(" ")
        line1.append(line2[0])
        line1.append(line2[1])
        line = []
        count_chain = count_chain + 1
        for j in line1:
            line = np.append(line,float(j))
        chain1.append(np.array(line))
    chain1 = np.array(chain1)
    return chain1
chain1 = chain_creat(lines_f1,chain1)
chain2 = chain_creat(lines_f2,chain2)

#Text editting
re = open("Assignment5/planck_chain_tauprior.txt", "a")

for i in range(len(chain1)):
    line = ""
    for j in range(len(chain1[i])):
        line = line + " " + str(chain1[i][j]) + " "
    line = line + str("\n")
    re.write(line)


#Plot chi question 4 v steps
steps = np.linspace(0,20000, 20000)
plt.plot(steps, chain_chi1)

#Plot chain[i] question 4 v steps
for i in range(len(chain1.T)):
    plt.plot(steps, chain1.T[i])
plt.show()
```

```
#Comparison
def process_chain(chain,chisq,T=1.0):
    dchi=chisq-np.min(chisq)
    #density in chain is exp(-0.5*chi^2/T), but
    #we wanted it to be exp(-0.5*chi^2)
    #so, we want to downweight by ratio, which is
    #exp(-0.5*chi^2*(1-1/T)).  We'll calculate the mean
    #and standard deviation of the chain, but will also
    #return the weights so you could calculate whatever you want

    wt=np.exp(-0.5*dchi*(1-1/T)) #the magic line that importance samples

    #calculate the weighted sum of the chain and the chain squared
    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
mean,errs,wts=process_chain(chain1,chain_chi1)
mean2,errs2,wts2=process_chain(chain2,chain_chi2)
nsig=5
npar=chain1.shape[1]
for i in range(npar):
    t1=mean[i]+errs[i]*nsig
    t2=mean[i]-errs[i]*nsig
    frac=(np.sum(chain1[:,i]>t1)+np.sum(chain1[:,i]<t2))/chain1.shape[0]
    frac2=(np.sum(chain2[:,i]>t1)+np.sum(chain2[:,i]<t2))/chain2.shape[0]
    print('fractions of samples on param ',i,' more than ',nsig,' is␣
 ↪',frac,frac2)
```

Gelman Rubin test

[ ]:
```
#Code that gets the chain with chi in the first collumn
#It is performs gelman_rubin test on the columns of the chain
import numpy as np
f = open("/Users/louis/Desktop/McGill/FALL 2022/PHYS 512/Assignment/5/
 ↪phys512-2022/Assignment5/planck_chain_tauprior.txt", "r")
def remove_values_from_list(the_list, val):
   return [value for value in the_list if value != val]
```

```
lines = f.readlines()
chain = []
for line in lines:
    l = line.split(" ")
    l = remove_values_from_list(l, "")
    l = l[1:7]
    my_line = []
    for i in range(0,6):
        my_line.append(float(l[i]))
    chain.append(np.array(my_line))
chain = np.array(chain)


nchain = 6
means=np.zeros([nchain,chain.shape[1]])
scats=np.zeros([nchain,chain.shape[1]])
for i in range(nchain):
    means[i,:]=np.mean(chain[i:],axis=0)
    scats[i,:]=np.std(chain[i:],axis=0)
#scatter of means
mean_scat=np.std(means,axis=0)
gelman_rubin=mean_scat/np.mean(scats,axis=0)
print('gelman_rubin scatters are ',gelman_rubin)
```

The first column is the chain with fixed params in 4 the other is the chain in 3 * fractions of samples on param 0 more than 5 is 0.0 0.0005 * fractions of samples on param 1 more than 5 is 0.0 0.0 * fractions of samples on param 2 more than 5 is 0.0 0.0 * fractions of samples on param 3 more than 5 is 0.0 0.0 * fractions of samples on param 4 more than 5 is 0.0002 0.8224 * fractions of samples on param 5 more than 5 is 0.0 0.00165