Muath Hamidi | Problem Set 5

In [43]:

```
#================================================
# Course: PHYS 512
# Problem Set: PS5
#================================================
# By: Muath Hamidi
# Email: muath.hamidi@mail.mcgill.ca
# Department of Physics, McGill University
# October 2022

#================================================
# Libraries
#================================================
import numpy as np # For math
import matplotlib.pyplot as plt # For graphs
from scipy.stats import chi2
import camb
import time
```

Problem 1

In [66]:

```
#=====================================================
# Loading Data
#=====================================================
dat = np.loadtxt("COM_PowerSpect_CMB-TT-full_R3.01.txt")

#=====================================================
# Data
#=====================================================
parameters=[60,0.02,0.1,0.05,2.00e-9,1.0] # parameters
planck=np.loadtxt("COM_PowerSpect_CMB-TT-full_R3.01.txt",skiprows=1) # data
ell=planck[:,0] # scale
spec=planck[:,1] # variance
errs=0.5*(planck[:,2]+planck[:,3]) # error

# This part is mostly taken from John's code.
#=====================================================
# Spectrum
#=====================================================
def Spectrum(parameters):
    lmax=3000
    H0=parameters[0] # hubble
    ombh2=parameters[1] # baryon density
    omch2=parameters[2] # dark matter density
    tau=parameters[3] # optical depth
    As=parameters[4] # Primordial amplitude of the spectrum
    ns=parameters[5] # Primordial tilt of the spectrum

    parameters=camb.CAMBparams()
    parameters.set_cosmology(H0=H0,ombh2=ombh2,omch2=omch2,mnu=0.06,omk=0,ta
    parameters.InitPower.set_params(As=As,ns=ns,r=0)
    parameters.set_for_lmax(lmax,lens_potential_accuracy=0)
    results=camb.get_results(parameters)
    powers=results.get_cmb_power_spectra(parameters,CMB_unit='muK')
    cmb=powers['total']
    tt=cmb[:,0]
    return tt[2:]

#=====================================================
# Plot
#=====================================================
model=Spectrum(parameters)
model=model[:len(spec)]
resid=spec-model
chisq=np.sum((resid/errs)**2)
print("chisq is ",chisq," for ",len(resid)-len(parameters)," degrees of free
# 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])

#=====================================================
# Plot
#=====================================================
plt.clf()
plt.plot(ell,model)
plt.errorbar(planck_binned[:,0],planck_binned[:,1],errs_binned,fmt='.')
plt.show()
```

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

The first χ^2 doesn't have a good fit. Let's see the new one.

In [67]:
```
#================================================
# Xi With Different Parameters
#================================================
parameters=[69,0.022,0.12,0.06,2.1e-9,0.95]
model=Spectrum(parameters)
model=model[:len(spec)]
resid=spec-model
chisq2=np.sum((resid/errs)**2)
dof=len(resid)-len(parameters)

print("New Chi is:", chisq2)
print("Mean & Variance = ", dof, " ± ", 2*dof)
```

```
New Chi is: 3272.2053559202204
Mean & Variance =  2501  ±  5002
```

The new χ^2 is closer to the currently-accepted value. It has less than %0.07 difference than your John's value (3270). The new χ^2 is within the expected error. So, it is acceptable.

Problem 2

I will use Newton's method here to find the best-fit parameters.

In [162]:

```
#=====================================================
# Derivative
#=====================================================
def Ndiff(fun,x,dx_ord): # This is the derivative which we did previously.
    order=-3+dx_ord
    dx=10**(order)
    yplus=fun(x+dx)
    yminus=fun(x-dx)
    yplus2=fun(x+2*dx)
    yminus2=fun(x-2*dx)
    F=(8*yplus-yplus2+yminus2-8*yminus)/(12*dx)
    return F

#=====================================================
# Gradient
#=====================================================
def pars_grad(fun,pars): # Gives the derivatives
    H0, ombh2, omch2, tau, As, ns = pars # Parameters
    fun_H02 = lambda H02:fun(pars) # H0
    grad_H0 = Ndiff(fun_H02,H0,dx_ord=+1)
    fun_ombh22 = lambda ombh22:fun(pars) # ombh2
    grad_ombh2 = Ndiff(fun_ombh22,ombh2,dx_ord=-2)
    fun_omch22 = lambda omch22:fun(pars) # omch2
    grad_omch2 = Ndiff(fun_omch22,omch2,dx_ord=-1)
    fun_tau2 = lambda tau2:fun(pars) # tau
    grad_tau = Ndiff(fun_tau2,tau,dx_ord=-2)
    fun_As2 = lambda As2:fun(pars) # As
    grad_As = Ndiff(fun_As2,As,dx_ord=-9)
    fun_ns2 = lambda ns2:fun(pars) # ns
    grad_ns = Ndiff(fun_ns2,ns,dx_ord=0)
    return np.array([grad_H0, grad_ombh2, grad_omch2, grad_tau, grad_As, gra

#=====================================================
# Newton's Method
#=====================================================
def Newton(p,y,iterations): # our Newton's method
    pars = p
    for i in range(iterations):
        pred = Spectrum(pars)[:len(y)]
        grad = pars_grad(Spectrum,pars)[:len(y)]
        r = y - pred
        err = (r**2).sum()
        r = r.T
        lhs=grad.T@grad
        rhs=grad.T@r
        dp=np.linalg.pinv(lhs)@(rhs)
        for jj in range(pars.size):
            pars[jj]=pars[jj]+dp[jj]
    return pars, dp

#=====================================================
# Parameters, Steps, and Errors
#=====================================================
iterations = 200
for i in range(iterations):
    p0 = np.array([69,0.022,0.12,0.06,2.1e-9,0.95]) # initial parameters
```

```
    newtons_pars, step = Newton(p0,spec,iterations) # get the parameters fro
    pred = Spectrum(newtons_pars)[:len(spec)]
    grad = pars_grad(Spectrum,newtons_pars)[:len(spec)]
    Ninv = np.linalg.inv(np.diag(errs))
    lhs = grad.T@Ninv@grad
    pars_errs = np.sqrt(np.diag(np.linalg.inv(lhs))) # Errors
    print("Errors:", pars_errs)
    np.savetxt("planck_fit_params.txt", np.array([newtons_pars, pars_errs]).
```

In [163]:

```
#=================================================
# Parameters' Data
#=================================================
newton = np.loadtxt("planck_fit_params.txt")
newtons_pars = newton[:,0]
pars_errs = newton[:,1]

#=================================================
# Parameters & Errors
#=================================================
print("H0 = {} ± {}".format(newtons_pars[0],pars_errs[0]))
print("Ombh2 = {} ± {}".format(newtons_pars[1],pars_errs[1]))
print("Omch2 = {} ± {}".format(newtons_pars[2],pars_errs[2]))
print("tau = {} ± {}".format(newtons_pars[3],pars_errs[3]))
print("As = {} ± {}".format(newtons_pars[4],pars_errs[4]))
print("ns = {} ± {}".format(newtons_pars[5],pars_errs[5]))
```

```
H0 = 68.3986031556773 ± 0.11784318244725173
Ombh2 = 0.022121744622708352 ± 2.2051083954933798e-05
Omch2 = 0.1180217548210856 ± 0.0002540622178874012
tau = 0.054078619377851016 ± 0.0036313575607223986
As = 2.1040491937315643e-09 ± 1.4638715063801205e-11
ns = 0.9661097310578793 ± 0.0006887883985434955
```

The results took time but seems good. Now, doing the mcmc...

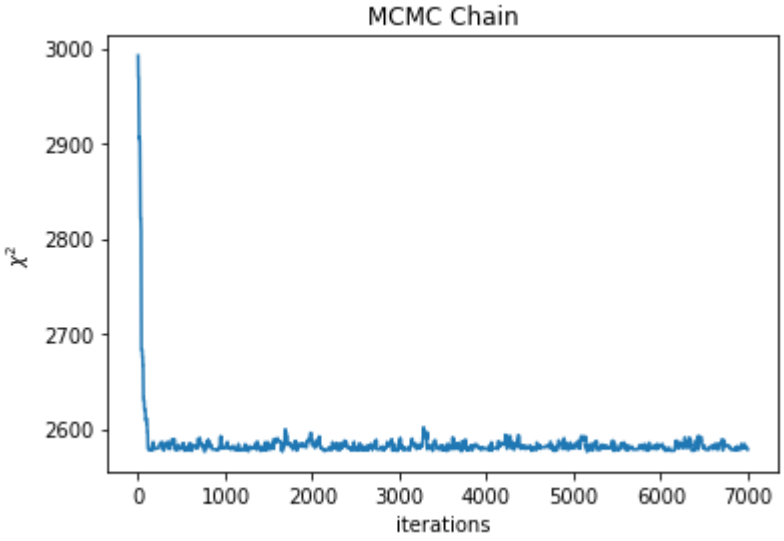
Problem 3

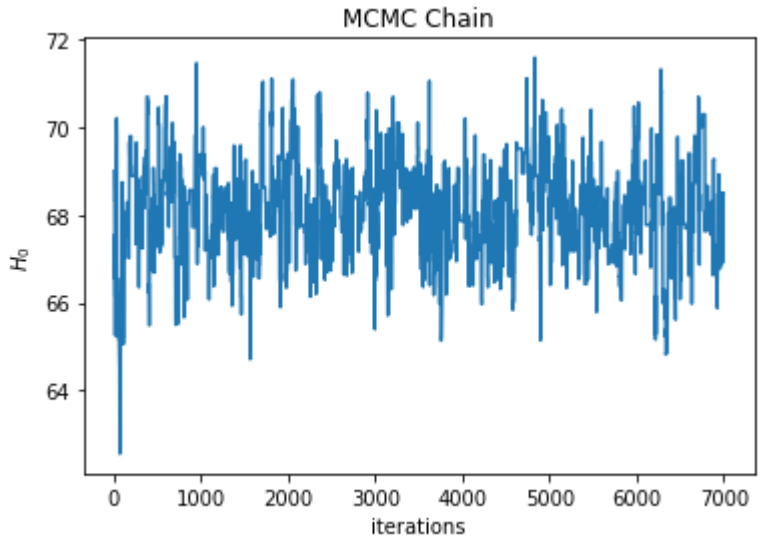
In [48]:

```
#=====================================================
# Chi Square
#=====================================================
def chi2(y,pred,errs): # chi square
    chi2 = np.sum((pred-y)**2/errs**2)
    return chi2

#=====================================================
# MCMC
#=====================================================
def get_step(y,parameters,chisq,cov,errs): # returns the new parameters and
    step = np.random.multivariate_normal(np.zeros(cov.shape[0]),cov) # step
    new_pars = parameters + step # new parameteres
    new_pred = Spectrum(new_pars)[:len(y)] # new predicted data
    new_chisq = chi2(y,new_pred,errs) # new chi square
    acc = np.exp(-0.5*(new_chisq - chisq)) # acceptance limit
    if np.random.rand(1) < acc:
        return new_pars, new_chisq
    else:
        return parameters, chisq


def MCMC_run_chain(y,p0,cov,errs,nstep): # MCMC
    n=p0.size # parameters array size
    pars_chain = np.zeros((nstep,n),dtype=float)
    pars_chain[0,:] = p0
    pred = Spectrum(p0)[:len(y)]
    chisq_chain = np.zeros(nstep,dtype=float) # initializing chi^2 chain
    chisq_chain[0] = chi2(y,pred,errs)

    # Fill the chains
    for i in range(1,nstep):
        parameters = pars_chain[i-1,:]
        chisq = chisq_chain[i-1]
        pars_new, chisq_new = get_step(y,parameters,chisq,cov,errs)
        pars_chain[i,:] = pars_new
        chisq_chain[i] = chisq_new

    return pars_chain, chisq_chain

#=====================================================
# CAMB
#=====================================================
nsteps = 7000
t0 = time.time()
for i in range(nsteps):
    p0 = np.array([69,0.022,0.12,0.054,2.1e-9,0.95])
    pred = Spectrum(p0)[:len(spec)]
    newtons_pars = np.loadtxt("planck_fit_params.txt")[:,0]
    grad = pars_grad(Spectrum,newtons_pars)[:len(spec)]
    Ninv = np.linalg.inv(np.diag(errs**2))
    lhs = grad.T@Ninv@grad
    cov = np.linalg.inv(lhs)
    pars_chain, chisq_chain = MCMC_run_chain(spec,p0,cov,errs,nsteps) # MCMC
    dat = np.zeros((nsteps,7)) # Data
    dat[:,0] = chisq_chain # chisq_chain data
```

```
    dat[:,1:] = pars_chain # pars_chain data
    np.savetxt("planck_chain.txt", dat) # data file
t1 = time.time()

# printing steps number and the time needed
print("Iterations: {} ; Time: {} sec".format(nsteps,t1-t0))


#================================================
# Plot
#================================================
# loading data
MCMC = np.loadtxt("planck_chain.txt")
planck_chain = MCMC[:,1:]
planck_chisq = MCMC[:,0]

x = np.arange(nsteps)

plt.figure()
plt.title("MCMC Chain")
plt.ylabel("$\chi^2$")
plt.xlabel("iterations")
plt.plot(x,planck_chisq)
plt.show()

plt.figure()
plt.title("MCMC Chain")
plt.ylabel("$H_0$")
plt.xlabel("iterations")
plt.plot(x,planck_chain[:,0])
plt.show()
```

```
Iterations: 7000 ; Time: 8718.910218954086 sec
```





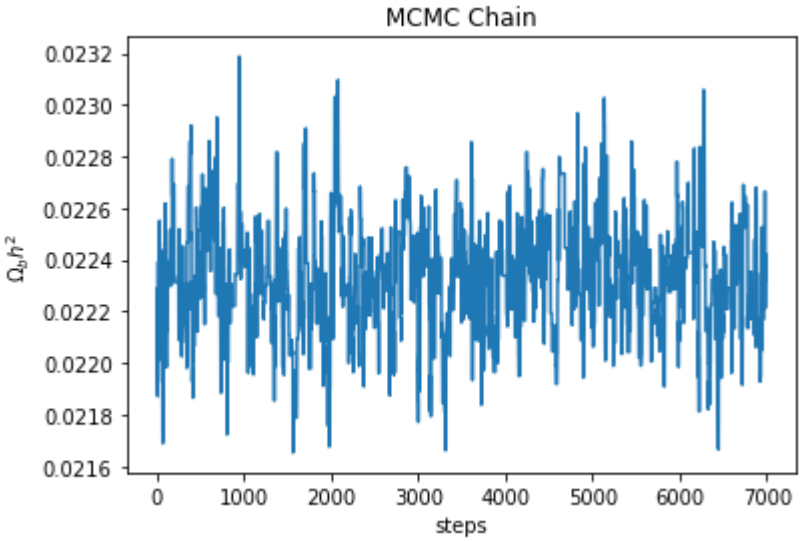
As you see from the both plots, the chains are converged. Now, the mean value of the dark energy...

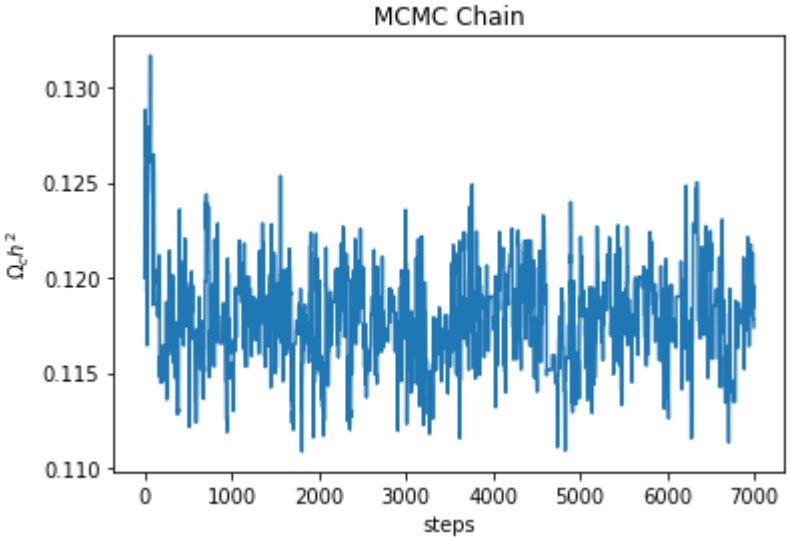
In [120]:

```
#=====================================================
# Parameters & Errors
#=====================================================
Size = 3500
pars = planck_chain[-1,:]
Error = np.std(planck_chain[Size:,:],axis=0) # Error

print("H0 = {} ± {}".format(pars[1],Error[1]))
print("Ombh2 = {} ± {}".format(pars[2],Error[2]))
print("Omch2 = {} ± {}".format(pars[3],Error[3]))
print("tau = {} ± {}".format(pars[4],Error[4]))
print("As = {} ± {}".format(pars[5],Error[5]))
print("ns = {} ± {}".format(pars[6],Error[6]))

#=====================================================
# Plots [omegas]
#=====================================================
plt.figure()
plt.title("MCMC Chain")
plt.plot(x,planck_chain[:,2])
plt.ylabel("$\Omega_b h^2$")
plt.xlabel("steps")
plt.show()

plt.figure()
plt.title("MCMC Chain")
plt.plot(x,planck_chain[:,3])
plt.ylabel("$\Omega_c h^2$")
plt.xlabel("steps")
plt.show()

#=====================================================
# Omgea Lambda
#=====================================================
H0 = pars[1] # H0
H0s = Error[1] # H0 error
ombh = pars[2] # Ω_b h^2
ombhs = Error[2] # Ω_b h^2 error
omch = pars[3] # Ω_c h^2
omchs = Error[3] # Ω_c h^2 error

# Gaussian distribution
H0a = np.random.normal(H0,H0s,nsteps) # H0 array
ombha = np.random.normal(ombh,ombhs,nsteps) # Ω_b h^2 array
omcha = np.random.normal(omch,omchs,nsteps) # Ω_c h^2 array

h0a = H0a / 100
omla = 1 - ombhs / h0a**2 - omchs / h0a**2 # Ω_λ array
oml = np.mean(omls) # Ω_λ
omls = np.std(omls) # Ω_λ error
print("Ω_λ = {} ± {}".format(oml,omls))
```

```
H0 = 68.48767696710387 ± 1.0806722779836055
Ombh2 = 0.02232214872902867 ± 0.00021057935196129075
Omch2 = 0.11739643662712652 ± 0.002420428836466584
tau = 0.10968971598019954 ± 0.024363932690417655
```

```
As = 2.3294809352899043e-09 ± 1.005098018795298e-10
ns = 0.9738071642465274 ± 0.005937240526629451
```





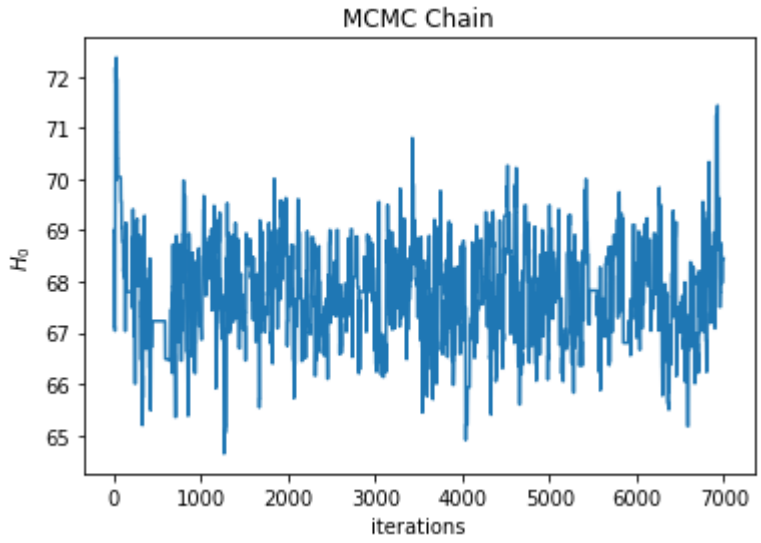

```
Ω_λ = 0.9047732999835447 ± 0.003136793414355047
```

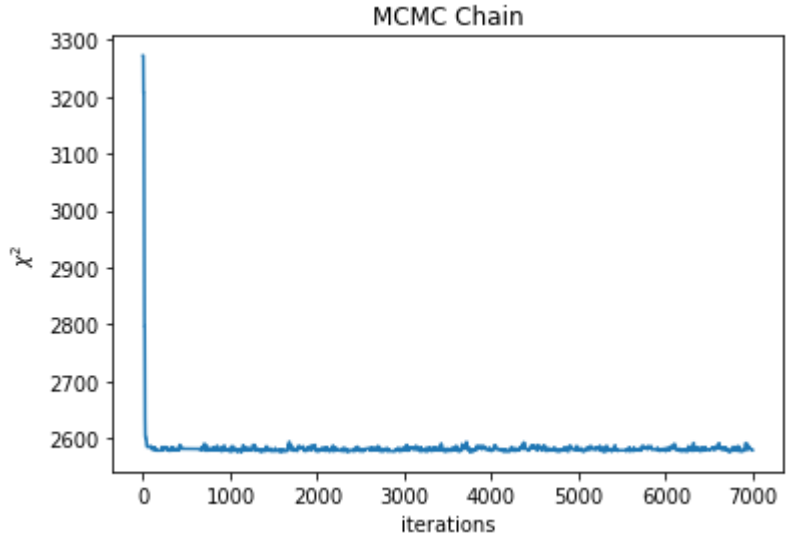
If you sum the three Ω's: $\Omega_{total} = 1.04$, with has 4% error.

Problem 4

In [146]:

```
#=====================================================
# tau & Data
#=====================================================
tau = 0.054 # tau
tau_s = 0.0074 # tau_sigma

dat = np.loadtxt("planck_chain_tauprior.txt") # loading data
tau_chain = dat[:,1:]
tau_chisq = dat[:,0]
tau_pars = tau_chain[-1,:]
tau_pars_s = np.std(tau_chain[Size:,:], axis=0) # Errors

#=====================================================
# Plot
#=====================================================
x = np.arange(nsteps)

plt.figure()
plt.title("MCMC Chain")
plt.plot(x,tau_chain[:,0])
plt.ylabel("$H_0$")
plt.xlabel("iterations")
plt.show()

plt.figure()
plt.title("MCMC Chain")
plt.ylabel("$\chi^2$")
plt.xlabel("iterations")
plt.plot(x,tau_chisq)
plt.show()

#=====================================================
# Parameters & Errors
#=====================================================
print("H0 = {} ± {}".format(tau_pars[0],tau_pars_s[0]))
print("Ombh2 = {} ± {}".format(tau_pars[1],tau_pars_s[1]))
print("Omch2 = {} ± {}".format(tau_pars[2],tau_pars_s[2]))
print("tau = {} ± {}".format(tau_pars[3],tau_pars_s[3]))
print("As = {} ± {}".format(tau_pars[4],tau_pars_s[4]))
print("ns = {} ± {}".format(tau_pars[5],tau_pars_s[5]))

#=====================================================
# Comparison with Problem 3
#=====================================================
print("=====================================================")
print("ΔH0 ± Δσ_H0 = {} ± {}".format(tau_pars[0]-pars[1], tau_pars_s[0]-Erro
print("ΔOmbh2 ± Δσ_Ombh2= {} ± {}".format(tau_pars[1]-pars[2], tau_pars_s[1]
print("ΔOmch2 ± Δσ_Omch2 = {} ± {}".format(tau_pars[2]-pars[3], tau_pars_s[2
print("Δtau ± Δσ_tau = {} ± {}".format(tau_pars[3]-pars[4], tau_pars_s[3]-Er
print("ΔAs ± Δσ_As = {} ± {}".format(tau_pars[4]-pars[5], tau_pars_s[4]-Erro
print("Δns ± Δσ_ns = {} ± {}".format(tau_pars[5]-pars[6], tau_pars_s[5]-Erro
```





H0 = 68.45265160914121 ± 0.9860310267291196
Ombh2 = 0.022329830495642865 ± 0.00020008385780982544
Omch2 = 0.11720945787957257 ± 0.0022175482210855905
tau = 0.051717622575093436 ± 0.007861739788101714

```
As = 2.0688066136158894e-09 ± 3.404911371166441e-11
ns = 0.9761097110577991 ± 0.005232839745511397
==================================================
ΔH0 ± Δσ_H0 = -0.03502535796265249 ± -0.09464125125448597
ΔOmbh2 ± Δσ_Ombh2= 7.681766614194702e-06 ± -1.0495494151465312e-05
ΔOmch2 ± Δσ_Omch2 = -0.0001869787475539536 ± -0.00020288061538099346
Δtau ± Δσ_tau = -0.057972093405106104 ± -0.016502192902315943
ΔAs ± Δσ_As = -2.6067432167401494e-10 ± -6.64606881678654e-11
Δns ± Δσ_ns = 0.002302546811271733 ± -0.0007044007811180539
```

As you can see, the differences between both methods are small, so both are consistant.

In [ ]:

```
1
```