PHYS-512: Problem Set 5

Simon Briesenick

October 31, 2022

1. With the initial set of parameters, χ^2 results in roughly `chisq = 15268`. For 2501 degrees of freedom, I would expect a good fit model to result in $\chi^2 = 2501 \pm 71$ (or in general χ^2 should be at $n \pm \sqrt{2n}$). Clearly, χ^2 lies outside these bounds. With the new set of parameters, χ^2 is calculated to be `chisq = 3272.2053559202122`. I would again conclude that the fit is not acceptable, although a lot better.

2. To implement Newton's method for this problem, I left the numeric differentiator from the previous problem set practically unchanged. The key difference is that I implemented a stopping condition for this problem. The stopping condition is triggered whenever the χ^2 value of two successive trial steps is closer then a tolerance factor `tol` that I've fixed to be `0.5`. Otherwise, Newton's method is executed for a number of steps `iterator`. As initial condition `chi` (the list that will store all later computed values for χ^2) is set to hold `4000`.

```python
[TRUNCATED]
iterator = 101
[TRUNCATED]
# starting condition
chi = [4000]
d_chi = []
tol = 0.5
converged = False
Ninv = np.eye(len(spec))/(errs**2)

for i in range(iterator):
    pred, grad = gradient(model_params[i])
    r = spec - pred
    r = np.matrix(r).transpose()
    grad = np.matrix(grad)
    lhs = grad.T*Ninv*grad
    rhs = grad.T*Ninv*r
    u, s, vh = np.linalg.svd(lhs)
    dm = vh.T@np.linalg.inv(np.diag(s))@u.T@rhs
    dm = dm.reshape(6,)
    if i == iterator - 1:
        break
        else:
            model_params[i + 1] = model_params[i] + dm
            new_chi = get_chs(pred, spec)
            chi.append(new_chi)
            converged = np.abs(chi[-2] - chi[-1]) < tol
            # print(f"new chi = {new_chi}")
        if converged:
            print('Newton Method has converged.')
            break
        print(f"Params in current loop: {model_params[i]}\nWith chi sq: {new_chi}.")
```

Doing this results in a χ^2 value of 3107.3948501647533 and parameters (dropped dimensions)

$$H_0 = 67.6 \qquad \Omega_b h^2 = 2.23 \times 10^{-2}$$
$$\Omega_c h^2 = 0.1193 \qquad \tau = 7.3 \times 10^{-2}$$
$$A_s = 2.17 \times 10^{-9} \qquad n_s = 9.69$$

with parameter errors

$$\Delta H_0 = 0.5 \qquad \Delta\Omega_b h^2 = 0.02 \times 10^{-2}$$
$$\Delta\Omega_c h^2 = 0.0008 \qquad \Delta\tau = 3.2 \times 10^{-2}$$
$$\Delta A_s = 0.13 \times 10^{-9} \qquad \Delta n_s = 0.006$$

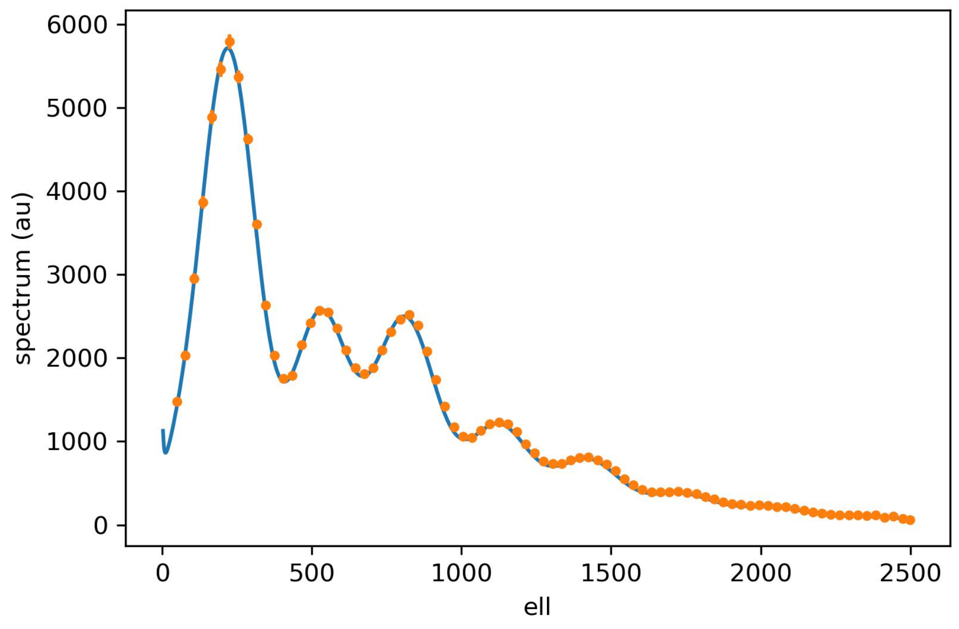

Figure 1: Fit to spectrum using Newton's method.

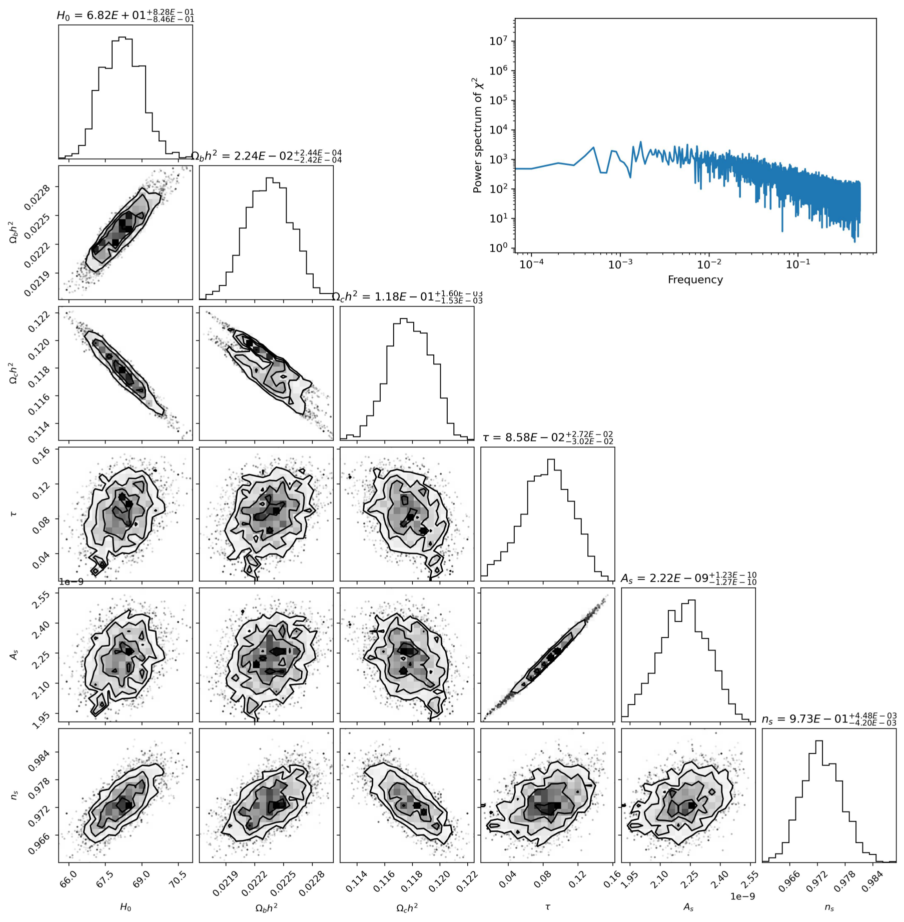

Figure 2: Corner plots for the first run of the MCMC chain with 10,000 steps. The inset shows a double-log plot of the power spectrum for χ^2.

3. With the curvature matrix from the previous problem, I initialized parameters and step sizes in parameter space for the MCMC chain - They are imported with the call to `newton.py` as

```
from newton import curv, best, errs, spec, get_spectrum
```

The MCMC chain is set up to run for `n_steps` number of steps (in this case 10,000). If a step in the chain is accepted, the new parameters and the associated value for χ^2 is saved in separate textfiles by the call

```
np.savetxt('mcmc_params.txt', par)
np.savetxt('chi_square.txt', chi_sq)
```

The results are shown in Fig. 2. One can see that the parameter values are more or less normally distributed. The FFT power spectrum for χ^2 also shows this - The frequency components are more or less constant with a linear decrease at higher frequencies. With $\Omega_b = 4.82 \pm 0.17$ and $\Omega_c = 25.7 \pm 0.96$, I get $\Omega_\Lambda = -29.5 \pm 1.1$
Complete Code:

```
import numpy as np
from newton import curv, best, errs, spec, get_spectrum
```

```python
def get_chi_sq(data, params):
    data = get_spectrum(params)[:len(spec)]
    chisq = np.sum((data - spec)**2/errs**2)
    return chisq

n_steps = 10000
steps_taken = 0
chi_sq = []
par = []
# initialize parameter list by using best fit +/- a little something
# par_errs = np.sqrt(np.diag(np.linalg.inv(curv)))
init_par = np.random.multivariate_normal(best, np.linalg.inv(curv)/2)
# init_par = best + 3.0*np.random.randn(len(best))*par_errs
chi_sq.append(get_chi_sq(spec, init_par))
par.append(init_par)


while steps_taken < n_steps:
    # generate new parameters
    step = np.random.multivariate_normal(np.zeros(best.size),
                np.linalg.inv(curv)/2)
    par_new = par[-1] + step
    # calculate new chi squared
    new_chi = get_chi_sq(spec, par_new)
    delta_chi = new_chi - chi_sq[-1]
    prob_step = np.exp(-0.5*delta_chi)
    # if prob_step is greater then one, chi will have increased
    # and we want to discard that step, otherwise accept with a certain
    # probability:
    if prob_step > np.random.rand(1):
        par.append(par_new)
        chi_sq.append(new_chi)
    else:
        # if step is not taken, copy last entries for params and chi_sq
        par.append(par[-1])
        chi_sq.append(chi_sq[-1])
    steps_taken += 1
    np.savetxt('mcmc_params.txt', par)
    np.savetxt('chi_square.txt', chi_sq)
    if steps_taken % 100 == 0:
        print(f"At {steps_taken/n_steps*100:.2f} percent")
```

4. I first sampled the latter half of my prior MCMC run to get an importance sampled estimate for the

new parameters and the covariance matrix. I calculated the weights according to the rule

$$w = \exp\left(-0.5\delta\chi^2\right) = \exp\left(-0.5 * \left(\frac{\tau - \tau_{\text{pol}}}{\sigma_{\text{pol}}}\right)^2\right) \tag{1}$$

where the subscript pol identifies the polarization data values. The normalized weights are then used to generate a new set of parameters, `sampled_params` by averaging the last `5000` steps. Similarly for the covariance matrix.

```python
import numpy as np

mcmc_params = np.loadtxt('mcmc_params.txt')
tau_pol = 0.054
tau_unc = 0.0074

# first step: importance sampling
# selecting only last 5000 steps
mcmc_params = mcmc_params[5001:,:]
# get tau values
tau = mcmc_params[:,3]
# define delta chi squared
delta_chi = (tau - tau_pol)**2/tau_unc**2
# define weights
w = np.exp(-0.5*delta_chi)
# normalize weights
w = w/np.sum(w)
# importance sample parameters
sampled_params = np.zeros(6)
for j in range(6):
    sampled_params[j] = sum(mcmc_params[:,j]*w)
# get covariance matrix
cov_mat = np.cov(mcmc_params, rowvar=False, aweights=w)
```

I get the following values for the parameters:

$$H_0 = 68.2 \qquad \Omega_b h^2 = 2.22 \times 10^{-2}$$
$$\Omega_c h^2 = 0.176 \qquad \tau = 5.6 \times 10^{-2}$$
$$A_s = 2.09 \times 10^{-9} \qquad n_s = 9.73$$

with parameter errors

$$\Delta H_0 = 0.7 \qquad \Delta\Omega_b h^2 = 0.02 \times 10^{-2}$$
$$\Delta\Omega_c h^2 = 0.001 \qquad \Delta\tau = 0.7 \times 10^{-2}$$
$$\Delta A_s = 0.03 \times 10^{-9} \qquad \Delta n_s = 0.004$$

At this point, I accidentally deleted my previous MCMC values. I am rerunning it right now, but I'm already late with submitting my homework, so I will upload this version first and then upload the rest when I'm done. The strategy for the rest of this questions is to modify the definition of my χ^2 getter function to read

```python
def get_chi_sq(data, params):
    data = get_spectrum(params)[:len(spec)]
    chisq = np.sum((data - spec)**2/errs**2) +
                 (params[3] - tau_pol)**2/(tau_unc)**2
    return chisq
```

and run the chain again with the importance sampled parameters and covariances.