BayesianOpticalPlant

November 1, 2015

1 Estimating aLIGO calibration parameters with MCMC

In this notebook we demonstrate how to use MCMC to estimate three parameters in the aLIGO DARM loop: the optical gain k, the DARM pole f_0, and a time delay τ. Together they constitute a simple transfer function:

$$P(f) = \frac{k \exp\left(-2\pi \mathrm{i} f \tau\right)}{1 + \mathrm{i} f / f_0}.$$

For the MCMC, we use the `emcee` package. To visualize the resulting posterior, we use the `corner` package.

2 Notebook settings

```
In [30]: %matplotlib inline
         %config InlineBackend.figure_format = 'png'
         from __future__ import division
         from IPython.display import clear_output

         import corner
         import emcee as mc
         import matplotlib as mpl
         import matplotlib.patches as patches
         import matplotlib.pyplot as plt
         import numpy as np
         import os
         import scipy.constants as scc
         import scipy.optimize as opt
         import scipy.integrate as scint
         import scipy.interpolate as sctrp
         import scipy.io as scio
         import scipy.signal as sig
         import scipy.special as scsp
         import sys

         # List of non-awful colors
         cList = [
                  (0, 0.2, 0.9),
                  (1.0, 0, 0),
                  (0, 0.7, 0),
                  (1.0, 0, 0.9),
                  (0.8, 0.8, 0),
                  (0, 0.6, 0.9),
                  (1, 0.5, 0),
```

```
            (0.5, 0.5, 0.5),
            (0.4, 0, 0.5),
            (0, 0, 0),
            (0.5, 0.3, 0),
            (0, 0.3, 0),
           ]

    # Now alter my matplotlib parameters
    mpl.rcParams.update({'axes.color_cycle': cList,
                         'font.family': 'serif',
                         'font.size': 12,
                         'legend.borderpad': 0.1,
                         'legend.fancybox': True,
                         'legend.fontsize': 11,
                         'legend.framealpha': 0.7,
                         'legend.handletextpad': 0.1,
                         'legend.labelspacing': 0.2,
                         'legend.loc': 'best',
                         'lines.linewidth': 1.5,
                         'text.usetex': True,
                         })
    mpl.rc("savefig", dpi=200)
```

3 Measured TF

We load a measured pcal sweep. To estimate the uncertainty on each measurement, we use the usual Bendat & Piersol coherence formula.

```
In [3]: ff, re, im = np.loadtxt('opt_plant_2015-09-10_tf.txt', unpack=1)
        tf = re+1j*im
        tf /= 3.477e-7*1e12 # convert from ct/m to mA/pm
        _, coh = np.loadtxt('opt_plant_2015-09-10_coh.txt', unpack=1)

In [4]: uncs_rel = np.sqrt((1-coh)/(2*coh*5))
        uncs_mag = uncs_rel * np.abs(tf)
        uncs_pha = np.copy(uncs_rel)
```

4 MCMC estimate

With the measurement in hand, we start the MCMC analysis. In the usual fashion, we start by defining a prior $p(k, f_0, \tau)$. We'll choose a prior that is uniform for $k > 0$, $f_0 > 0$, and $\tau > 0$:

$$p(k, f_0, \tau) = \begin{cases} 1 & \text{if } k > 0 \text{ and } f_0 > 0 \text{ and } \tau > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Then we write down the likelihood function for the probability of observing the data $\mathbf{d}$ (i.e., the measured transfer function) given certain values of k, f_0, and τ:

$$p(\mathbf{d} \,|\, k, f_0, \tau) \propto \exp\left[-\sum_i \frac{\left| P_i - \hat{P}_i(k, f_0, \tau) \right|^2}{2\sigma_i^2} \right],$$

where P_i denotes the measurement of $P(f)$ at f_i, σ_i is the measured uncertainty at f_i, and $\hat{P}_i(k, f_0, \tau)$ denotes the value of $P(f_i)$ that one would compute given certain values of k, f_0, and τ. Then the posterior

is

$$p(k, f_0, \tau \,|\mathbf{d}) = \frac{1}{Z}\, p(\mathbf{d} \,|\, k, f_0, \tau)\, p(k, f_0, \tau),$$

where Z is a normalization constant.

```
In [69]: def lnprob(th, ff, tf, uncs):
             tf0 = th[0]/(1+1j*ff/th[1])*np.exp(-2j*np.pi*ff*th[2])
             if th[0]<0 or th[1]<0 or th[2]<0:
                 return -np.inf
             else:
                 return -np.sum(np.abs(tf-tf0)**2/(2*uncs**2))

In [70]: ndim = 3
         nwalkers = 100
         rr = np.transpose(np.vstack([np.random.normal(0, 0.03, nwalkers),
                         np.random.normal(0, 10, nwalkers),
                         np.random.normal(0, 1e-6, nwalkers)]))
         p0 = np.tile(np.array([3.6, 300.0, 50e-6]), (nwalkers,1))+rr

In [71]: samp = mc.EnsembleSampler(nwalkers, ndim, lnprob, args=[ff, tf, uncs_rel])

In [72]: pos, prob, state = samp.run_mcmc(p0, 100)
         samp.reset()

In [73]: samp.run_mcmc(pos, 2000);

In [74]: results = np.copy(samp.flatchain)
         results[:,2] *= 1e6
```

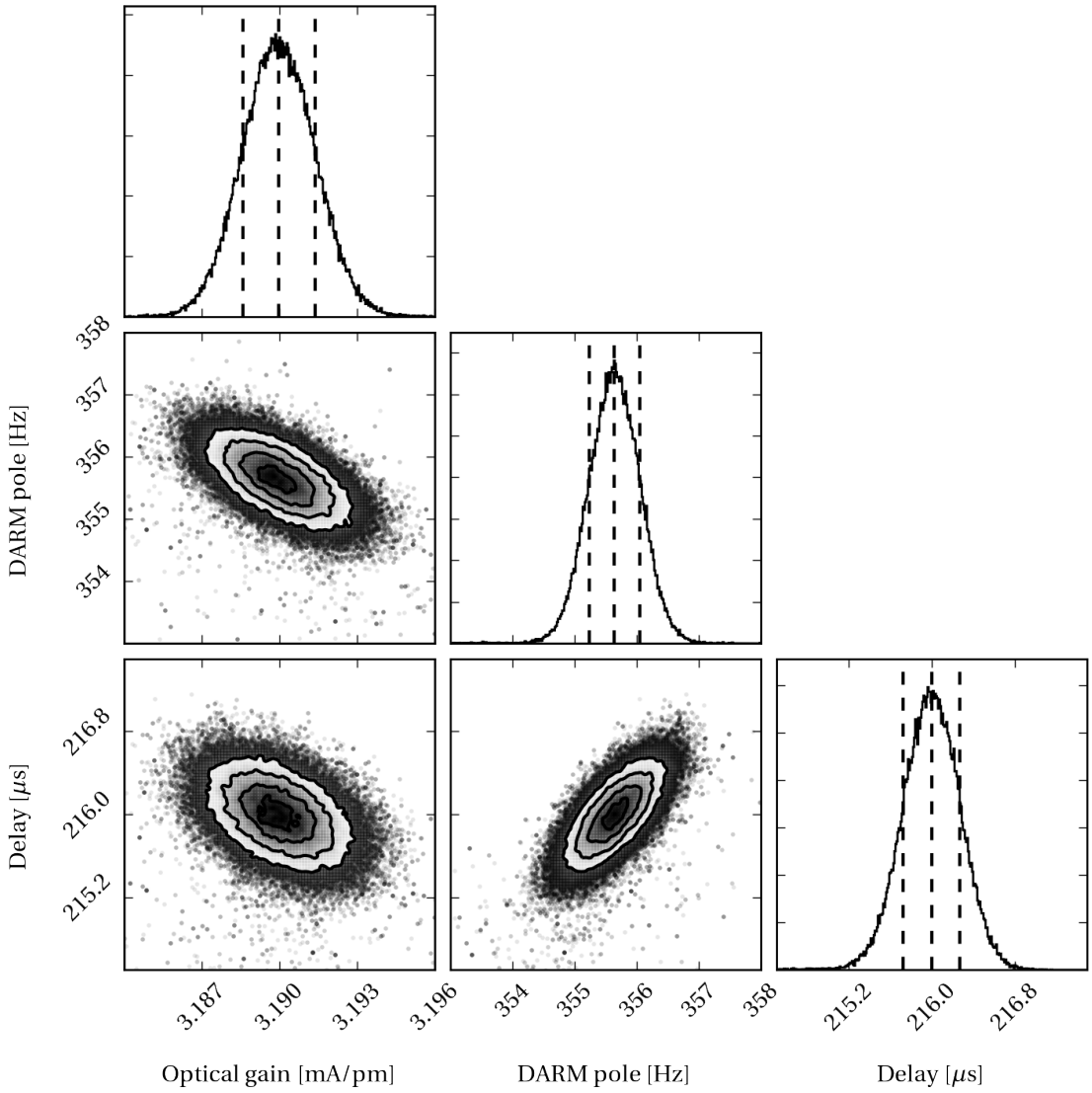
5 Corner plot of posterior

Now we make a corner plot of the posterior. This shows the 2D and 1D marginalizations over k, f_0, and τ.

For the three 1D marginalizations, the `corner` package will return the 16th, 50th, and 84th quantiles for us.

```
In [77]: range_list = [[3.184, 3.196], [353, 358], [214.5, 217.5]]
         h_c = corner.corner(results, bins=300,
                             labels=[r'Optical gain [mA/pm]',
                                     r'DARM pole [Hz]',
                                     r'Delay [$\mu$s]'],
                             quantiles=[0.16, 0.5, 0.84],
                             smooth=2,
                             range=range_list,
                             verbose=True)
```

```
Quantiles:
[(0.16, 3.1885855399594085), (0.5, 3.1899713399141421), (0.84, 3.1913818118898547)]
Quantiles:
[(0.16, 355.23410566367505), (0.5, 355.64323313417935), (0.84, 356.04852187121259)]
Quantiles:
[(0.16, 215.7246032037597), (0.5, 215.9981689385807), (0.84, 216.26800321106808)]
```



6 TF plot: measurement and MCMC

Here we plot the measurement, along with the MCMC estimate using median values.

```
In [12]: def gain_pole_delay(ff, gain, pole, delay):
             return gain/(1+1j*ff/pole)*np.exp(-2j*np.pi*delay*ff)
```

```
In [93]: f_full = np.logspace(0, 4, 200)
         g_med = 3.190 # mA/pm
         p_med = 355.6 # Hz
         t_med = 216.0e-6 # s
```

```
In [94]: h_tf = plt.figure()
         axm_tf = h_tf.add_subplot(211)
         axp_tf = h_tf.add_subplot(212, sharex=axm_tf)
```

```
axm_tf.errorbar(ff, np.abs(tf), 10*uncs_mag, fmt='o', ms=4, alpha=0.9, c=(0, 0, 0.7))
axm_tf.loglog(f_full, np.abs(gain_pole_delay(f_full, g_med, p_med, t_med)),
              c=(1, 0, 0), alpha=0.8)
axm_tf.set_xscale('log')
axm_tf.set_yscale('log')
axm_tf.set_ylim(0.75, 10.5)
axm_tf.set_ylabel('Magnitude [mA/pm]')
axm_tf.grid('on', which='both', alpha=0.7)
axm_tf.grid(ls='solid', which='major', c=(0.5, 0.5, 0.5))
#axm_tf.text(120, 4.5, r'Error bars £\times\;10£')
axp_tf.errorbar(ff, np.angle(tf, deg=True), uncs_pha*180/np.pi*10,
                fmt='o', ms=4, alpha=0.7, c=(0, 0, 0.7),
                label=r'Measured [error bars $\times\;10$]')
axp_tf.semilogx(f_full, np.angle(gain_pole_delay(f_full, g_med, p_med, t_med), deg=True),
               label='MCMC median values', c=(1, 0, 0), alpha=0.9)
axp_tf.legend()
axp_tf.set_xscale('log')
myyticks = np.arange(-225, 226, 45)
myyticklabels = ['$'+str(tick)+'^\circ$' for tick in myyticks]
axp_tf.set_yticks(myyticks)
axp_tf.set_yticklabels(myyticklabels)
axp_tf.set_yticks(np.arange(-180, 181, 15), minor=True)
axp_tf.set_xlim(9, 1.5e3)
axp_tf.set_ylim(-185, 20)
axp_tf.set_ylabel('Phase')
axp_tf.grid('on', which='both', alpha=0.7)
axp_tf.grid(ls='solid', which='major', c=(0.5, 0.5, 0.5))
```