BayesianOpticalPlant

E. D. Hall

December 2, 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:

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

For the MCMC, we use the `emcee` package. To visualize the resulting posterior, we use the `corner` package. We also use the `uncertainties` package a little bit.

2 Notebook settings

In [73]:
```
%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

from uncertainties import ufloat as uf

# List of non-awful colors
cList = [
        (0, 0.2, 0.9, 0.8),
        (1.0, 0, 0, 0.8),
        (0, 0.7, 0, 0.8),
        (1.0, 0, 0.9, 0.8),
        (0.8, 0.8, 0, 0.8),
        (0, 0.6, 0.9, 0.8),
        (1, 0.5, 0, 0.8),
        (0.5, 0.5, 0.5, 0.8),
        (0.4, 0, 0.5, 0.8),
        (0, 0, 0, 0.8),
        (0.5, 0.3, 0, 0.8),
```

```
            (0, 0.3, 0, 0.8),
           ]

    # 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,
                         'text.latex.preamble': r'\usepackage{amsmath}, \usepackage{upgreek}'
                         })
    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 [74]: 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 [75]: 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:

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

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

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

where Z is a normalization constant.

```
In [76]: 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 [77]: 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 [78]: samp = mc.EnsembleSampler(nwalkers, ndim, lnprob, args=[ff, tf, uncs_rel])

In [79]: pos, prob, state = samp.run_mcmc(p0, 500)
         samp.reset()

In [80]: samp.run_mcmc(pos, 2000);

In [81]: results = np.copy(samp.flatchain)
         results[:,2] *= 1e6
```

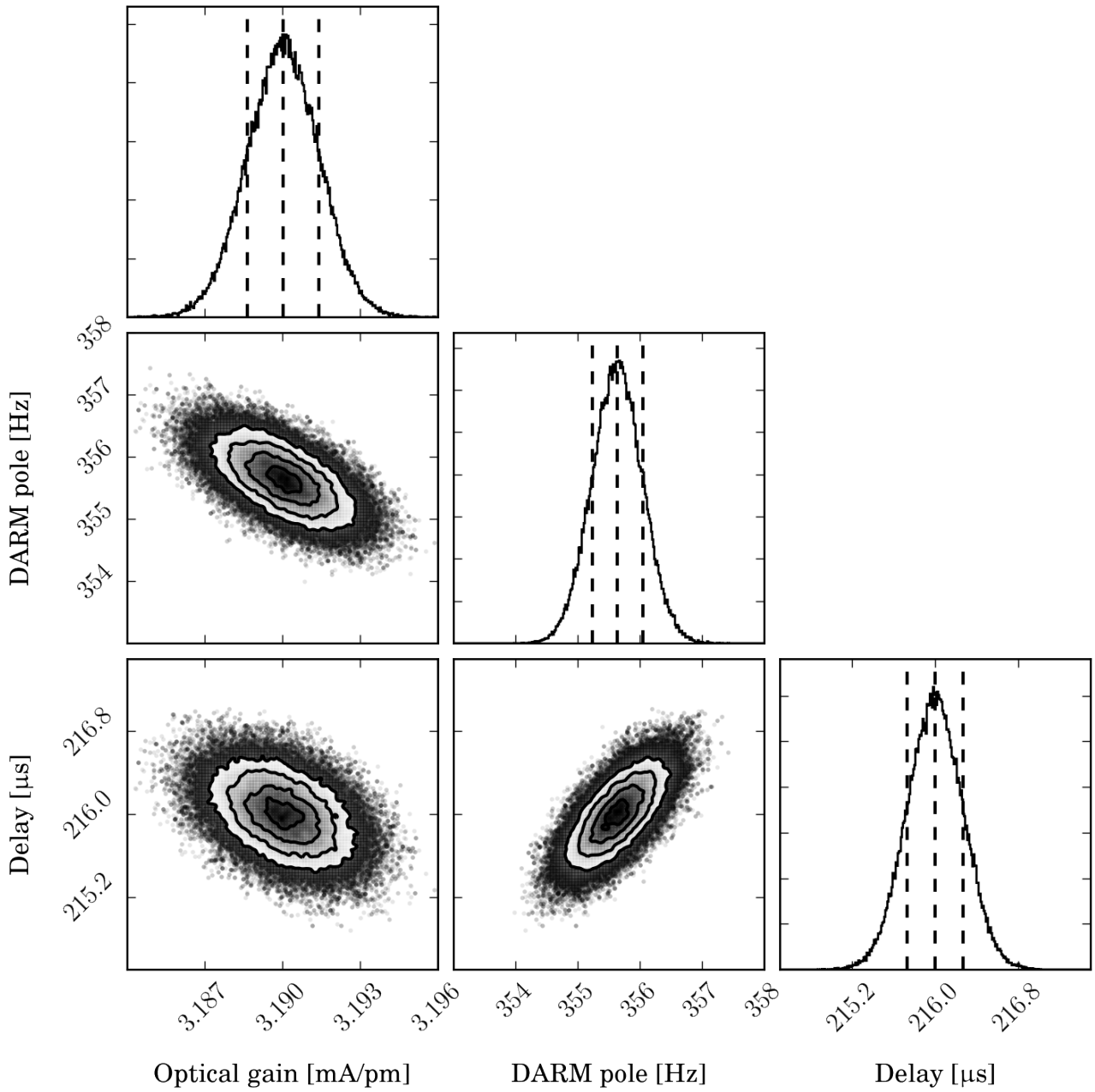
5 Corner plot of posterior

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 [82]: 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 [$\upmu$s]'],
                             quantiles=[0.16, 0.5, 0.84],
                             smooth=2,
                             range=range_list,
                             verbose=True)
```

```
Quantiles:
[(0.16, 3.1886562416233959), (0.5, 3.1900361386134706), (0.84, 3.1914103154534041)]
Quantiles:
[(0.16, 355.23347543999387), (0.5, 355.63941894429274), (0.84, 356.04557356816281)]
Quantiles:
[(0.16, 215.73122663271232), (0.5, 216.00015850460909), (0.84, 216.26914344615835)]
```



6 Gaussian approximation

We take the posterior pdf and extract mean values and a covariance matrix. Since our posterior is quite Gaussian, the means and the covariance matrix provide a good description of the pdf (i.e., we don't need to compute any higher moments), and the medians more or less coincide with the means.

```
In [83]: means = np.mean(results, axis=0)
         medians = np.median(results, axis=0)
         covmat = np.cov(results.T)
```

```
In [84]: print(covmat)
```

```
[[  1.91500300e-06  -3.45995551e-04  -1.55013526e-04]
 [ -3.45995551e-04   1.65519824e-01   7.59134232e-02]
 [ -1.55013526e-04   7.59134232e-02   7.29783123e-02]]
```

```
In [85]: print('Optical gain: {:.2uS} mA/pm  [median: {:.5g} mA/pm]'.format(uf(means[0], covmat[0,0]**0.5), medians[
         print('DARM pole: {:.2uS} Hz  [median: {:.5g} Hz]'.format(uf(means[1], covmat[1,1]**0.5), medians[1]))
         print('Delay: {:.2uS} us  [median: {:.5g} us]'.format(uf(means[2], covmat[2,2]**0.5), medians[2]))
         print('----')
         print('Covariance of optical gain and DARM pole: {:.3g} (mA/pm) Hz'.format(covmat[0,1]))
         print('Covariance of optical gain and delay: {:.3g} (mA/pm) us'.format(covmat[0,2]))
         print('Covariance of DARM pole and delay: {:.3g} Hz us'.format(covmat[1,2]))
```

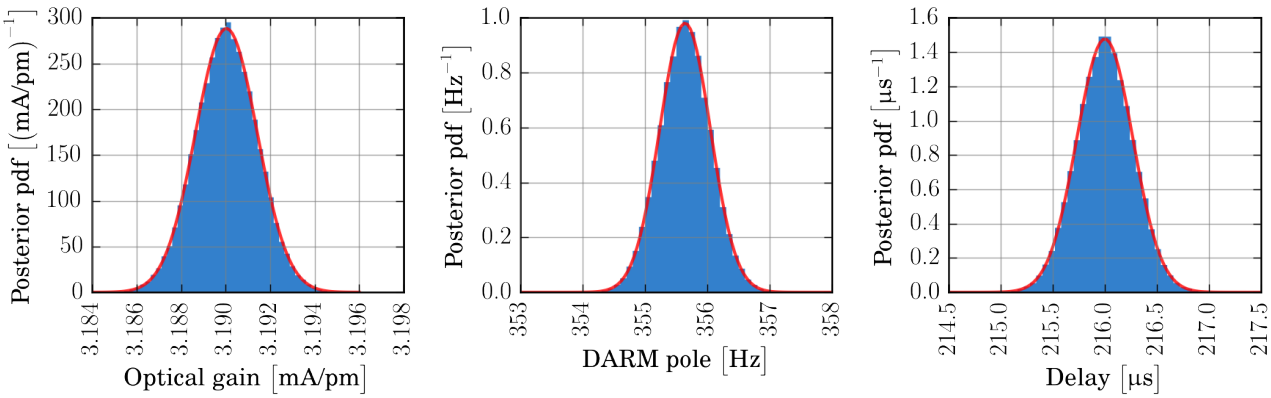
```
Optical gain: 3.1900(14) mA/pm  [median: 3.19 mA/pm]
DARM pole: 355.64(41) Hz  [median: 355.64 Hz]
Delay: 216.00(27) us  [median: 216 us]
----
Covariance of optical gain and DARM pole: -0.000346 (mA/pm) Hz
Covariance of optical gain and delay: -0.000155 (mA/pm) us
Covariance of DARM pole and delay: 0.0759 Hz us
```

7 Normalized posteriors

Here we plot the normalized 1D posteriors, along with the Gaussian approximation (using the above covariance matrix).

```
In [86]: def gaussian(x, m, s):
             return np.exp(-(x-m)**2/(2*s**2))/(np.sqrt(2*np.pi)*s)
```

```
In [87]: h_post = plt.figure(figsize=(9,3))
         ax_opt = plt.subplot(131)
         ax_pole = plt.subplot(132)
         ax_delay = plt.subplot(133)

         hist_kwargs = {'normed': True,
                        'bins': 50,
                        'alpha': 1,
                        'color': (0.2, 0.5, 0.8),
                        'histtype': 'stepfilled',
                        'linewidth': 0}

         _ = ax_opt.hist(results[:,0], range=range_list[0], **hist_kwargs)
         opt_arr = np.linspace(range_list[0][0], range_list[0][1], 100)
         ax_opt.plot(opt_arr, gaussian(opt_arr, means[0], covmat[0,0]**0.5), c=cList[1])
         ax_opt.grid('on', ls='solid', c=(0.5, 0.5, 0.5), alpha=0.7, zorder=0)
         ax_opt.set_ylabel(r'Posterior pdf $\bigl[\bigl(\text{mA/pm}\bigr)^{-1}\bigr]$')
         ax_opt.set_xlabel(r'Optical gain $\bigl[\text{mA/pm}\bigr]$')
         #ax_opt.legend(['Gaussian approx.', 'Raw'])
         plt.setp(ax_opt.xaxis.get_majorticklabels(), rotation=90)

         _ = ax_pole.hist(results[:,1], range=range_list[1], **hist_kwargs)
         pole_arr = np.linspace(range_list[1][0], range_list[1][1], 100)
         ax_pole.plot(pole_arr, gaussian(pole_arr, means[1], covmat[1,1]**0.5), c=cList[1])
         ax_pole.grid('on', ls='solid', c=(0.5, 0.5, 0.5), alpha=0.7, zorder=0)
         ax_pole.set_ylabel(r'Posterior pdf $\bigl[\text{Hz}^{-1}\bigr]$')
         ax_pole.set_xlabel(r'DARM pole $\bigl[\text{Hz}\bigr]$')
         plt.setp(ax_pole.xaxis.get_majorticklabels(), rotation=90)

         _ = ax_delay.hist(results[:,2], range=range_list[2], **hist_kwargs)
         delay_arr = np.linspace(range_list[2][0], range_list[2][1], 100)
         ax_delay.plot(delay_arr, gaussian(delay_arr, means[2], covmat[2,2]**0.5), c=cList[1])
         ax_delay.grid('on', ls='solid', c=(0.5, 0.5, 0.5), alpha=0.7, zorder=0)
         ax_delay.set_ylabel(r'Posterior pdf $\bigl[\upmu\text{s}^{-1}\bigr]$')
         ax_delay.set_xlabel(r'Delay $\bigl[\upmu\text{s}\bigr]$')
         plt.setp(ax_delay.xaxis.get_majorticklabels(), rotation=90)

         h_post.tight_layout()
```

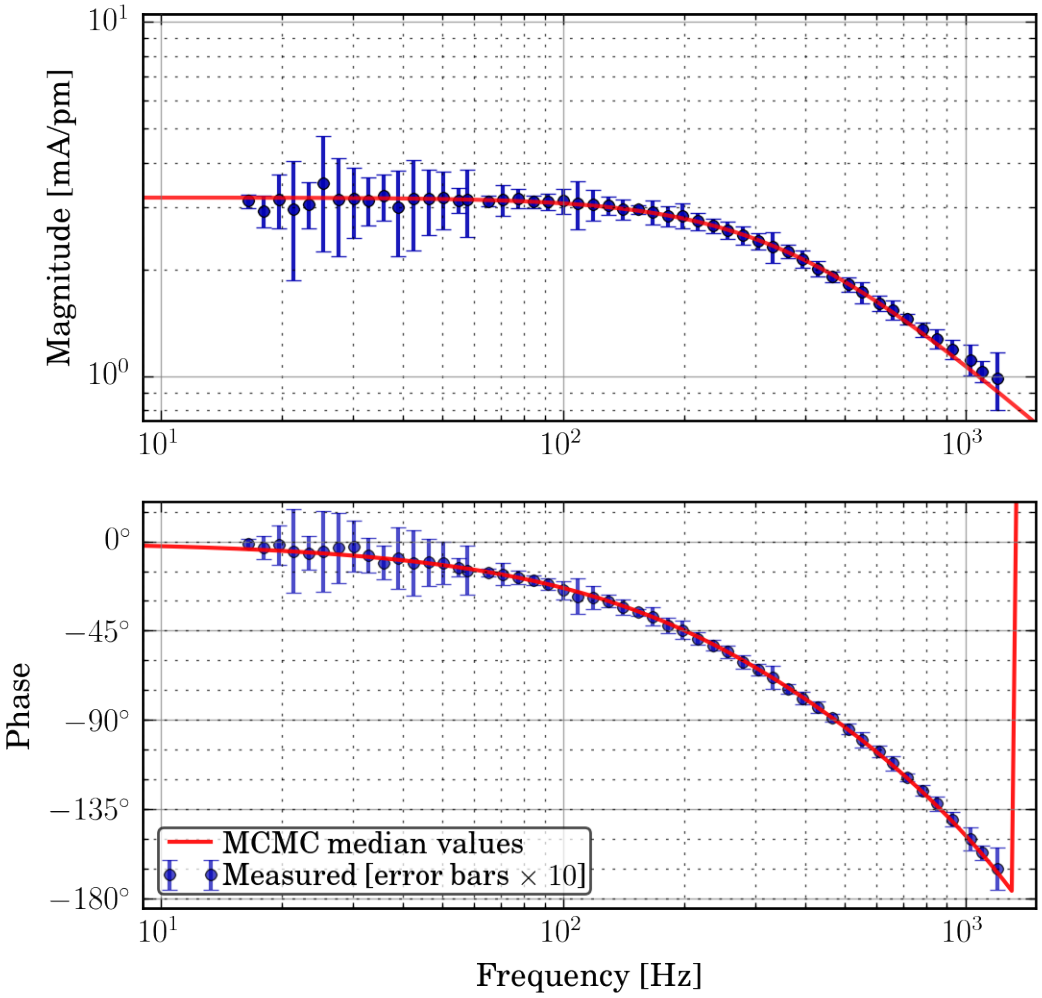


Blue shows the raw marginalized posteriors for each parameter. Red shows the Gaussian approximation.

8 TF plot: measurement and MCMC

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

```
In [88]: def gain_pole_delay(ff, gain, pole, delay):
             return gain/(1+1j*ff/pole)*np.exp(-2j*np.pi*delay*ff)

In [89]: 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 [90]: h_tf = plt.figure(figsize=(6,6))
         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))
         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.set_xlabel('Frequency [Hz]')
         axp_tf.grid('on', which='both', alpha=0.7)
         axp_tf.grid(ls='solid', which='major', c=(0.5, 0.5, 0.5))
```


Magnitude [mA/pm]
10^1
10^0
10^1
10^2
10^3
Phase
$0°$
$-45°$
$-90°$
$-135°$
$-180°$
MCMC median values
Measured [error bars × 10]
Frequency [Hz]

In [ ]: