SensingFunction

July 5, 2016

1 Fitting sensing functions

created: 2016-Jul-5 In order to get the estimated parameters for calibration. See the original by E. Hall at https://dcc.ligo.org/LIGO-T1500553

One difference is that now we include the signal recycling induced optical spring. The sensing function we use is in form of

$$S(f; H, f_p, \tau, f_s) = \frac{He^{-2i\pi f\tau}}{1 + if/f_p} \frac{f^2}{f^2 + f_s^2}$$

where H, f_p, τ and f_s are an optical gain, cavity pole, time delay and spring frequency, respectively.

```
In [74]: import numpy as np
         import emcee as mc
         import matplotlib.pyplot as plt
         import corner
         from matplotlib import rc



         %matplotlib inline

         ############ plot settings
         font={'family':'sans-serif',
                 'size':18}
         axes={'linewidth':2.0}
         legend_s={'fontsize':'small'}
         rc('font',**font)
         rc('axes',**axes)
         rc('legend',**legend_s)
```

2 Load data

```
In [53]: a = np.loadtxt("../data/2016-07-01_H1DARM_SensingFunction.txt", skiprows = 1)
         freq = a[:,0]
         tfdata = a[:,1] * np.exp(1j*a[:,3])
         tfunc = a[:,2] * np.exp(1j*a[:,4])
```

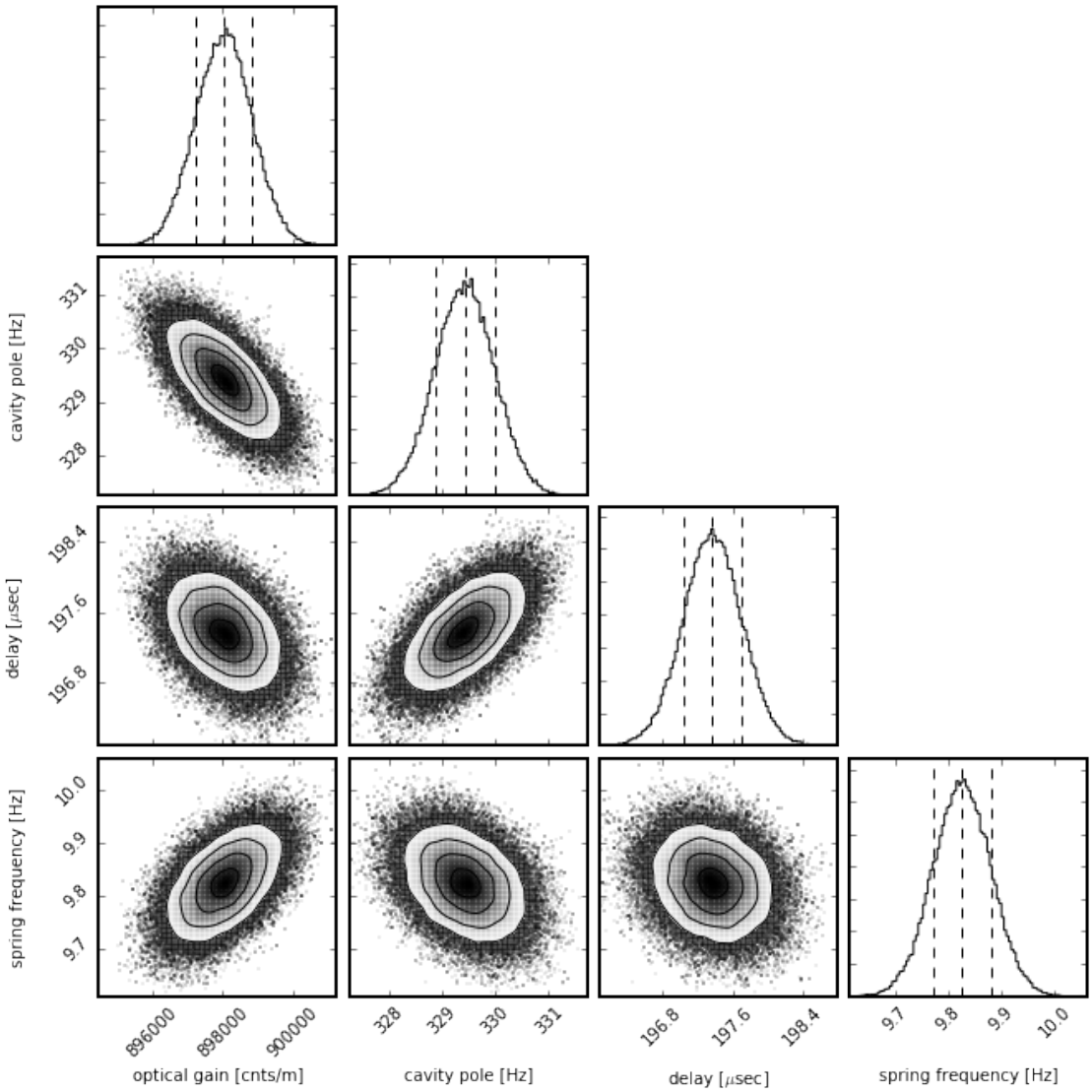
```
In [94]: # checking the uncertanties
         #if True:
         if False:
             plt.figure(101)
             plt.semilogx(freq, np.abs(tfunc)/np.abs(tfdata), '.', ms=20, label='normalized abs. unc.')
             plt.semilogx(freq, np.angle(tfunc), label='phase unc.')
```

```
            plt.grid()
            plt.legend()
            plt.show()
```

3 Define some functions

```
In [54]: # define sensing functional form
         def sens (theta, ff):
             tf0 = theta[0]/(1+1j*ff/theta[1]) * np.exp(-2j*np.pi*ff*theta[2])\
                     * ff**2/(ff**2 + theta[3]**2)
             return tf0

         def lnprob(theta, ff, tf, uncs):
             tf0 = sens(theta, ff)
             if (theta[0]< 0) or (theta[1]<0) or (theta[2]<0) or (theta[3]<0):
                 return -np.inf
             else:
                 return -np.sum(np.abs(tf-tf0)**2/(2*uncs**2))

In [56]: ndim = 4
         nwalkers = 100
         rr = np.transpose(np.vstack([np.random.normal(0, 0.03, nwalkers),
                                      np.random.normal(0, 1, nwalkers),
                                      np.random.normal(0, 1e-6, nwalkers),
                                      np.random.normal(0, 1, nwalkers)]))

         p0 = np.tile(np.array([547200, 360.0, 50e-6, 10]), (nwalkers,1))+rr

In [57]: samp = mc.EnsembleSampler(nwalkers, ndim, lnprob, args=[freq, tfdata, np.abs(tfunc)])
         pos, prob, state = samp.run_mcmc(p0, 500)
         samp.reset()

In [58]: samp.run_mcmc(pos, 2000);

In [59]: results = np.copy(samp.flatchain)
         results[:,2] *= 1e6

In [60]: #range_list = [[3.184, 3.196], [353, 358], [214.5, 217.5], [0, 20]]
         h_c = corner.corner(results, bins=100,
                                        labels=['optical gain [cnts/m]',
                                                'cavity pole [Hz]',
                                                'delay [$\mu$sec]',
                                                'spring frequency [Hz]'],
                             quantiles = [0.16, 0.5, 0.84],
                             smooth=2,
                             verbose = True)
```

```
Quantiles:
[(0.16, 897242.15219886787), (0.5, 898044.36817667249), (0.84, 898831.01946488628)]
Quantiles:
[(0.16, 328.86057831675572), (0.5, 329.42188411436558), (0.84, 329.97975584356061)]
Quantiles:
[(0.16, 197.03951628437522), (0.5, 197.37155698229674), (0.84, 197.70826146300416)]
Quantiles:
[(0.16, 9.7704649117354982), (0.5, 9.825221402378336), (0.84, 9.879268389249658)]
```



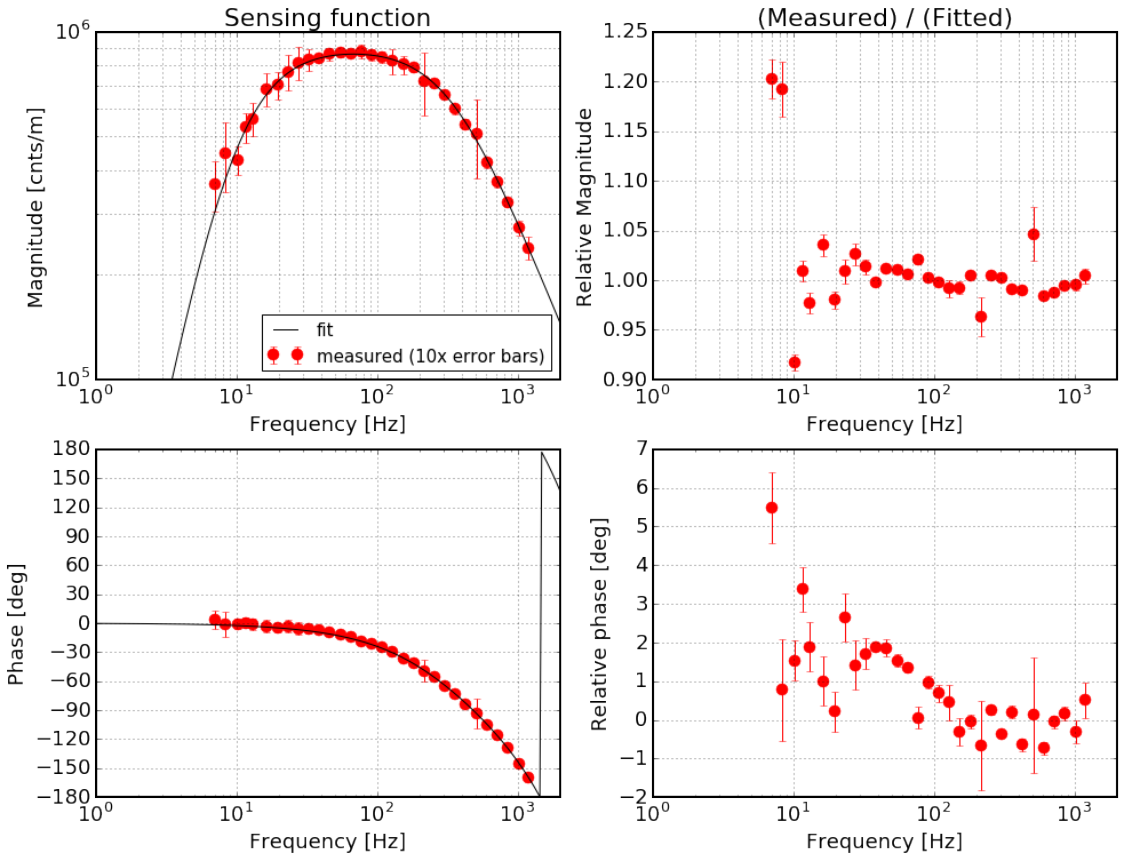
In [61]:
```
means = np.mean(samp.flatchain, axis=0)
medians = np.median(samp.flatchain, axis=0)
covmat = np.cov(samp.flatchain.T)
#print(covmat)
```

In [62]:
```
print("= = = =")
print("Optical gain = %e +/- %e [cnts/m]"%(means[0], covmat[0,0]**0.5) )
print("Cavity pole = %e +/- %e [Hz]"%(means[1], covmat[1,1]**0.5) )
print("Time delay = %e +/- %e [usec]"%(means[2] * 1e6, covmat[2,2]**0.5*1e6) )
print("Spring frequency = %e +/- %e [Hz]"%(means[3], covmat[3,3]**0.5) )
```

```
= = = =
Optical gain = 8.980393e+05 +/- 7.965381e+02 [cnts/m]
Cavity pole = 3.294194e+02 +/- 5.626792e-01 [Hz]
Time delay = 1.973719e+02 +/- 3.386483e-01 [usec]
Spring frequency = 9.825078e+00 +/- 5.453538e-02 [Hz]
```

4 Plot the resulting TF

In [80]:
```
frange  = [1.0, 2e3]
fff = np.logspace(np.log10(frange[0]), np.log10(2e3), 256)


h = plt.figure(101, figsize=(16,12))
fu = h.add_subplot(221)
fu.errorbar(freq, np.abs(tfdata), 10*np.abs(tfunc), fmt='.', ms=20, color='r',
           label='measured (10x error bars)')
fu.loglog(fff, np.abs(sens(means, fff)), color='k', label='fit')
fu.set_ylim(1e5, 1e6)
fu.set_xlim(frange)
fu.grid(which ='both')
fu.set_xlabel('Frequency [Hz]')
fu.set_ylabel('Magnitude [cnts/m]')
fu.legend(loc='best')
fu.set_title('Sensing function')

fl = h.add_subplot(223)
fl.errorbar(freq, np.angle(tfdata, deg=True), 10*np.angle(tfunc, deg=True),
            fmt='.', color='r', ms=20,
           label = 'measured')
fl.semilogx(fff, np.angle(sens(means, fff), deg=True), color='k', label='fit')
fl.grid()
fl.set_yticks(np.arange(13)*30-180)
fl.set_xlim(frange)
fl.set_ylim(-180, 180)
fl.set_xlabel('Frequency [Hz]')
fl.set_ylabel('Phase [deg]')

fuu = h.add_subplot(222)
fuu.errorbar(freq, np.abs(tfdata / sens(means, freq)), np.abs(tfunc / sens(means, freq)),fmt='
fuu.set_xscale('log')
plt.grid(which='both')
fuu.set_xlim(frange)
fuu.set_title('(Measured) / (Fitted)')
fuu.set_xlabel('Frequency [Hz]')
fuu.set_ylabel('Relative Magnitude')

fll = h.add_subplot(224)
fll.errorbar(freq, np.angle(tfdata/sens(means, freq), deg=True), np.angle(tfunc, deg=True),fmt
fll.grid()
fll.set_xscale('log')
fll.set_xlim(frange)
fll.set_xlabel('Frequency [Hz]')
fll.set_ylabel('Relative phase [deg]')

plt.savefig('../figures/SensingFunciton_detuned_20160705.png', transparent=True)
plt.show()
```


Sensing function
Magnitude [cnts/m]
Frequency [Hz]
fit
measured (10x error bars)
(Measured) / (Fitted)
Relative Magnitude
Frequency [Hz]
Phase [deg]
Frequency [Hz]
Relative phase [deg]
Frequency [Hz]