SensingFunctionSimulation

September 19, 2016

1 Fitting sensing functions

created: 2016-Sep-15 See the original at LHO alog 28274 https://alog.ligo-wa.caltech.edu/aLOG/index.php?callRep=28274

In [14]:
```
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 [15]:
```
a = np.loadtxt("../data/2016-09-16_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 [16]:
```
# 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 [17]:
```
# 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 - 1j*ff*theta[3]*theta[4])
    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:
        abslike = -np.sum((np.abs(tf)-np.abs(tf0))**2/(2*np.abs(uncs)**2))
        phaselike = -np.sum((np.angle(tf)-np.angle(tf0))**2/(2*np.angle(uncs)**2))
        return abslike + phaselike
```

In [18]:
```
ndim = 5
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),
                             np.random.normal(0, 5e-3, nwalkers)]))

p0 = np.tile(np.array([1.1e6, 360.0, 50e-6, 10, 0.03]), (nwalkers,1))+rr
```

In [20]:
```
samp = mc.EnsembleSampler(nwalkers, ndim, lnprob, args=[freq, tfdata, tfunc])
pos, prob, state = samp.run_mcmc(p0, 1000)
samp.reset()
```

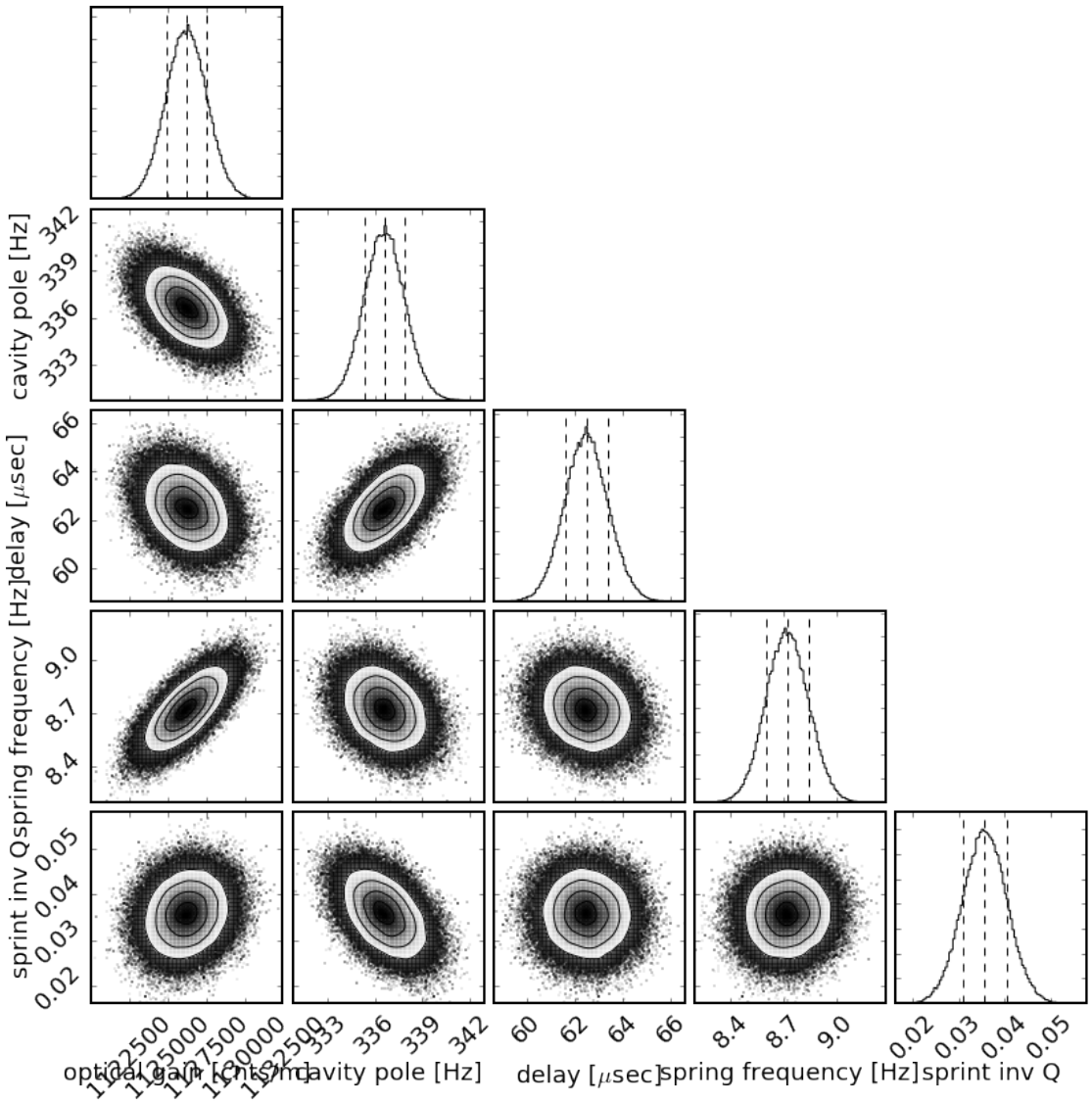
In [21]:
```
samp.run_mcmc(pos, 4000);
```

In [22]:
```
results = np.copy(samp.flatchain)
results[:,2] *= 1e6
```

In [23]:
```
#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]',
                            'sprint inv Q'],
                    quantiles = [0.16, 0.5, 0.84],
                    smooth=2,
                    verbose = True)
h_c.savefig('../figures/SensingFunction_detuned_with_invQ_Cornerplot.png', transparent=True)
```

```
Quantiles:
[(0.16, 1124922.1079782848), (0.5, 1126239.3633761399), (0.84, 1127564.8534067543)]
Quantiles:
[(0.16, 335.35160467842866), (0.5, 336.61684749531003), (0.84, 337.89820706936757)]
Quantiles:
[(0.16, 61.606232253404301), (0.5, 62.489020225273336), (0.84, 63.393935835713648)]
Quantiles:
[(0.16, 8.6041691404103418), (0.5, 8.7209862028296481), (0.84, 8.8409988320470454)]
```

```
Quantiles:
[(0.16, 0.03094674571975831), (0.5, 0.035723517071771481), (0.84, 0.040488210290198452)]
```




```
In [24]: means = np.mean(samp.flatchain, axis=0)
         medians = np.median(samp.flatchain, axis=0)
         covmat = np.cov(samp.flatchain.T)
         #print(covmat)
```

```
In [25]: 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) )
         print("Spring Inverse Q = %e +/- %e [Hz]"%(means[4], covmat[4,4]**0.5) )
```

```
= = = =
Optical gain = 1.126242e+06 +/- 1.333590e+03 [cnts/m]
```

```
Cavity pole = 3.366236e+02 +/- 1.283924e+00 [Hz]
Time delay = 6.249683e+01 +/- 9.003380e-01 [usec]
Spring frequency = 8.721946e+00 +/- 1.188382e-01 [Hz]
Spring Inverse Q = 3.572832e-02 +/- 4.797796e-03 [Hz]
```

```
In [33]: fs = means[3]
         Q = 1./means[4]
         fs, Q
```

```
Out[33]: (8.7219459284897578, 27.988999533599269)
```

```
In [38]: smajor = 0.5*(-1./fs/Q + np.sqrt(1./(fs*Q)**2 + 4 *fs**2) )
         sminor = 0.5*(-1./fs/Q - np.sqrt(1./(fs*Q)**2 + 4 *fs**2) )
         smajor, sminor
```

```
Out[38]: (8.7198979837039197, -8.7239943542534526)
```

```
In [39]: 1./means[0]
```

```
Out[39]: 8.8790898225146722e-07
```

4 Plot the resulting TF

```
In [27]: 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(medians, fff)), color='k', label='fit')
         fu.set_ylim(1e5, 2e6)
         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(medians, 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(medians, freq)), np.abs(tfunc / sens(medians, freq)),fr
         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(medians, freq), deg=True), np.angle(tfunc, deg=True),fr
fll.grid()
fll.set_xscale('log')
fll.set_xlim(frange)
fll.set_xlabel('Frequency [Hz]')
fll.set_ylabel('Relative phase [deg]')

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