Simple SGWB parameter estimation with Cobaya

(Actually too simple – do not use for real projects!)

1. Defining and testing the likelihood

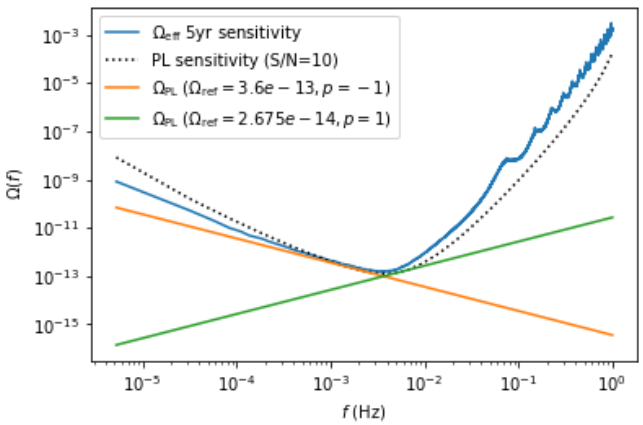
In [2]:
```
from __future__ import division
%matplotlib inline
import sys, platform, os
from matplotlib import pyplot as plt
import numpy as np

# Mock binned 5yr sensitivity from eLISAToolBox v 0.3 by A. Petiteau (last m
odified 15/05/2015)
sens_file_path = r'../MBH/StochBkgdSNR_Cfgv1.1/Sens_L6A2M5N2P2D28.txt'

# Load it
res = np.loadtxt(sens_file_path, usecols=[0, 2, 3])
freqs = res[:,0]  # Centres of freq bins
OmEff = res[:,1]  # Effective Power-spectrum sensitivity, per sec and Hz
PLS = res[:,2]    # Power-law sensitity of Thrane and Romano

# Scale effective sensitivity appropriately
# Assume equally spaced bins. Actually they aren't quite.. but closeish
df = freqs[1] - freqs[0]
yr=365.25*86400.
T = 5*yr
# Actual sensitivity (squared): variance per bin
bin_vars = OmEff**2/(T*df)


def GWSpecAcoustic(s, OmMax) :
    """
    Function returning the spectrum corresponding to a causal broken power s
pectrum
    OmMax * s**3 (7/(4 + 3 * s**2))**(3.5), with s := freq/freq_peak
    """
    GW = OmMax * s**3 * (7./(4. + 3.*s**2))**3.5
    return GW


def GWSpecPowerLaw(s, OmRef, p) :
    """
    Function returning a power law to test PL sensitivity
    """
    GW = OmRef * s**p
    return GW


def chi_squared(freq_peak, OmMax, fiducial_omega=0):
    theory_omega = GWSpecAcoustic(freqs/freq_peak,OmMax)
    return np.sum((theory_omega-fiducial_omega)**2/bin_vars)


def chi_squared_pl(freq_ref, OmRef, p, fiducial_omega=0):
    theory_omega = GWSpecPowerLaw(freqs/freq_ref, OmRef, p)
    return np.sum((theory_omega-fiducial_omega)**2/bin_vars)


# For tests S/N from eLISA toolbox
def StockBkg_ComputeSNR(SensFr, SensOm, GWFr, GWOm, Tobs, fmin=-1, fmax=-1)
:
    """

    ***From eLISAToolBox v 0.3 by A. Petiteau (last modified 15/05/2015)***

    Compute Signal to Noise Ratio and the used frequency range fmin and fmax
    for a given sensitivity defined by the two numpy array (same size) SensF
r for frequency
    and SensOm for sensitivity in Omega unit,  a given spectrum defined  by
the two numpy array (same size) GWFr for frequency
    and GWOm for GW in Omega unit, and a given observation time Tobs in year
s.
    If frange is not defined, the frequency range will be adjust on the two
```

In [2]:
```
def get_plot_with_sensitivity(power_law=False):
    """
    Returns axis for a plot, already containing 5yr sensitivity,
    and optionally (``power_law=True``) power law sensitivity.
    """
    plt.figure()
    #plt.loglog(freqs, OmEff, ls=':', label=r"$\Omega_\mathrm{eff}$ sensitiv
ity")
    plt.loglog(freqs, np.sqrt(bin_vars), label=r"$\Omega_\mathrm{eff}$ 5yr s
ensitivity")
    if power_law:
        plt.loglog(freqs, PLS, ls=':', color='k',
                   label="PL sensitivity (S/N=10)")
    plt.xlabel(r"$f$ (Hz)")
    plt.ylabel(r"$\Omega(f)$")
    plt.legend()
    return plt.gca()
```

In [3]:
```
# Plotting power laws

plplot = get_plot_with_sensitivity(power_law=True)

# Values below have been selected to prove that the given PL sensitivity is,
# as defined, the envelope of PL's with S/N=10

freq_ref = 1e-3
Om_pl1, p1 =  0.36e-12, -1
print "chi-sq ", chi_squared_pl(freq_ref,Om_pl1, p1)
snr1 = StockBkg_ComputeSNR(freqs, OmEff, freqs, GWSpecPowerLaw(freqs/freq_re
f,Om_pl1, p1), T)
print "SNR^2: ", snr1[0]**2

plplot.loglog(freqs, GWSpecPowerLaw(freqs/freq_ref,Om_pl1, p1), label=r"$\Om
ega_\mathrm{PL}\;(\Omega_\mathrm{ref}=%g, p=%g)$"%(Om_pl1, p1))

Om_pl2, p2 =  0.2675e-13, 1
print "chi-sq ", chi_squared_pl(freq_ref,Om_pl2, p2)
snr2 = StockBkg_ComputeSNR(freqs, OmEff, freqs, GWSpecPowerLaw(freqs/freq_re
f,Om_pl2, p2), T)
print "SNR^2: ", snr2[0]**2

plplot.loglog(freqs, GWSpecPowerLaw(freqs/freq_ref,Om_pl2, p2), label=r"$\Om
ega_\mathrm{PL}\;(\Omega_\mathrm{ref}=%g, p=%g)$"%(Om_pl2, p2))

# Update the legend!
plplot.legend()
```

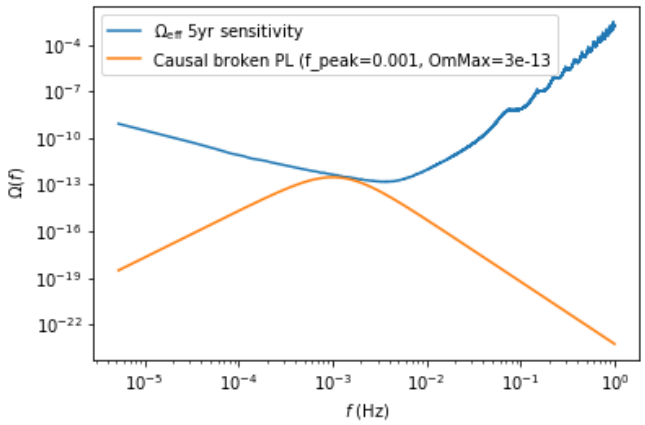
```
chi-sq  100.35702887262131
SNR^2:  100.35702887261893
chi-sq  100.06539987388027
SNR^2:  100.06539987386257
```

Out[3]:
```
<matplotlib.legend.Legend at 0x7efe03fd38d0>
```



In [4]:
```
# Plotting phase transitions

phtplot = get_plot_with_sensitivity(power_law=False)

freq_peak = 1e-3
OmMax=0.3e-12

print "chi-sq: ", chi_squared(freq_peak, OmMax)
print "SNR^2:  ", StockBkg_ComputeSNR(freqs, OmEff, freqs, GWSpecAcoustic(fr
eqs/freq_peak,OmMax), T)[0]**2


phtplot.loglog(freqs, GWSpecAcoustic(freqs/freq_peak,OmMax),
           label="Causal broken PL (f_peak=%g, OmMax=%g"%(freq_peak, OmMax))

# Update the legend!
phtplot.legend();
```

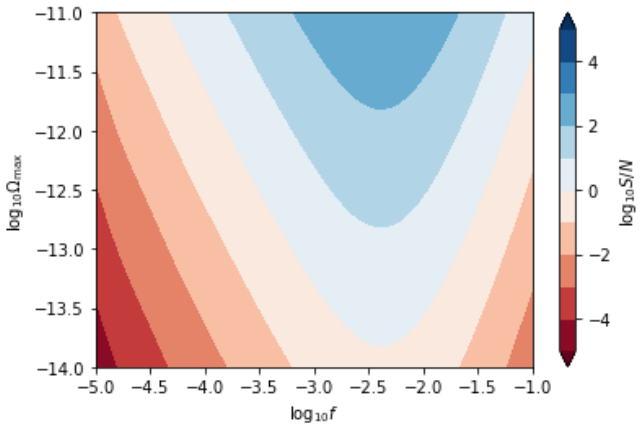
```
chi-sq:  33.810297530568064
SNR^2:   33.81029753056777
```



2. S/N and detectability of a phase transition

In [5]:
```
peaks = np.logspace(-5,-1,num=60, base=10)
oms = np.logspace(-14,-11, base=10)
chis2 = np.zeros((peaks.size,oms.size))
for i, peak in enumerate(peaks):
    for j, om in enumerate(oms):
        chis2[i,j] = np.exp(-chi_squared(peak,om)/2)
peaksm, omsm = np.meshgrid(np.log10(peaks),np.log10(oms))
```

In [7]:
```
# In terms of SN (log10)
log10SN = np.zeros((peaks.size,oms.size))
for i, peak in enumerate(peaks):
    for j, om in enumerate(oms):
        log10SN[i,j] = np.log10(StockBkg_ComputeSNR(freqs, OmEff, freqs, GWS
pecAcoustic(freqs/peak,om), T)[0])
levels = range(-5,1+5)
log10SN = np.clip(log10SN, a_min=levels[0], a_max=levels[-1])
p = plt.contourf(peaksm, omsm, log10SN.T, cmap="RdBu", extend="both", levels
=levels)
cbar = plt.colorbar()
cbar.ax.set_ylabel(r"$\log_{10}S/N$")
plt.xlabel(r'$\log_{10}f$')
plt.ylabel(r'$\log_{10}\Omega_{\rm max}$');
#labels are log_10
```



3. Basic forecasts (req. cobaya)

In [3]:
```
# Prepare a fiducial model and a likelihod function
log10_freq_peak_fiducial = -3  # (log10Hz)
log10_OmMax_fiducial = -12.5

print "Fiducial model: log10fpeak=%g, log10OmMax=%g"%(log10_freq_peak_fiduci
al, log10_OmMax_fiducial)
fiducial_model =  GWSpecAcoustic(freqs/10.**log10_freq_peak_fiducial, 10.**l
og10_OmMax_fiducial)

def loglike(log10_OmMax, log10_freq_peak):
    return -0.5*chi_squared(10.**log10_freq_peak, 10.**log10_OmMax, fiducial
_model)
```

```
Fiducial model: log10fpeak=-3, log10OmMax=-12.5
```

In [8]:
```
# Preparing Cobaya's input
info = {"params":
        {"log10_freq_peak":
            {"prior": {"min":  -5, "max":  1}, "proposal": 0.12, "latex": "
\log_{10}f"},
         "log10_OmMax":
            {"prior": {"min": -16, "max": -2}, "proposal": 0.12, "latex": r
"\log_{10}\Omega_\mathrm{max}"}},
        "likelihood": {"lisa": loglike},
        "sampler": {"mcmc": {"max_samples": 10000, "Rminus1_stop": 0.01, "ma
x_tries": np.inf}}
       }
```

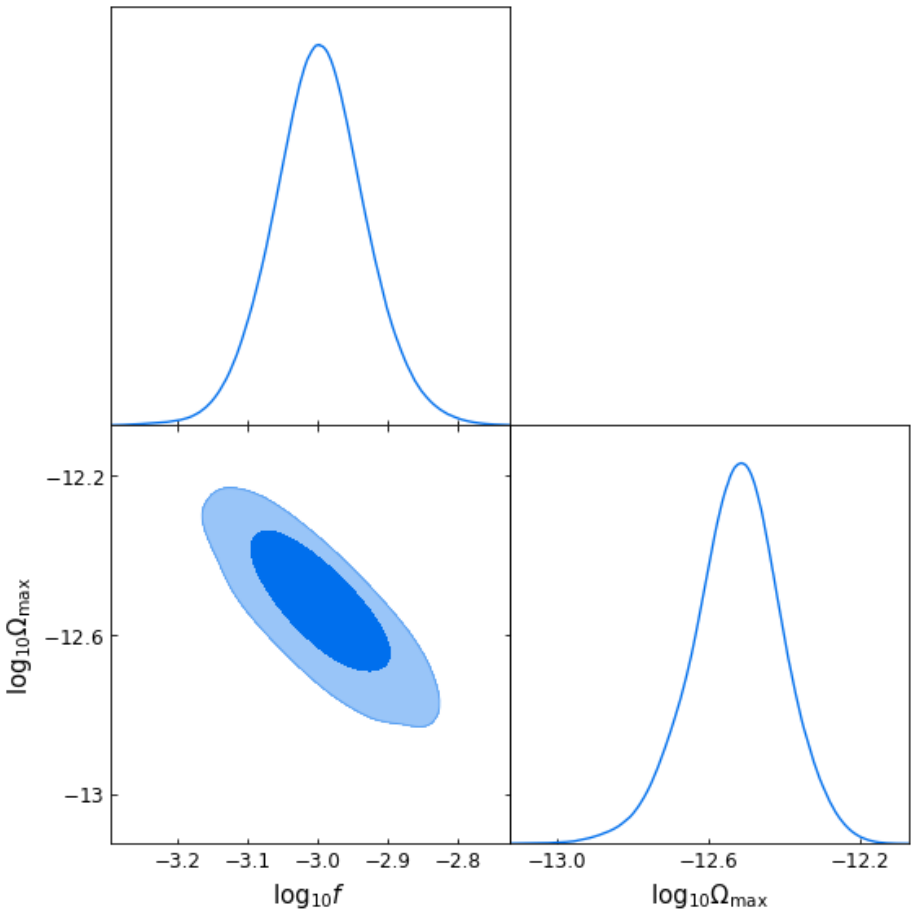
In [9]:
```
# Starting points for chain (optional)
from random import random
info["params"]["log10_freq_peak"]["ref"] = log10_freq_peak_fiducial*(1+(rand
om()-0.5)*0.05)
info["params"]["log10_OmMax"]["ref"] = log10_OmMax_fiducial*(1+(random()-0.5
)*0.05)
```

In [10]:
```
from cobaya.run import run
updated_info, products = run(info)
```

```
[mcmc] Covariance matrix not present. We will start learning the covariance o
f the proposal earlier: R-1 = 30 (was 2).
[mcmc] Initial covariance matrix.
[mcmc] Initial point:
[mcmc]
weight  minuslogpost  log10_freq_peak  log10_OmMax  minuslogprior  minuslogpr
ior__0      chi2  chi2__lisa
   1.0      9.293121        -3.009965   -12.794205       4.430817          4.
430817  9.724607    9.724607
[mcmc] Sampling! (NB: nothing will be printed until 40 burn-in samples have b
een obtained)
[mcmc] Finished burn-in phase: discarded 40 accepted steps.
[mcmc] Checkpoint: 80 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.300542 after 80 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 160 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.287350 after 160 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 240 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.112756 after 240 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 320 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.049191 after 320 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 400 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.101102 after 400 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 480 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.048315 after 480 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 560 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.014219 after 560 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 640 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.031509 after 640 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 720 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.021638 after 720 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 800 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.017710 after 800 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 880 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.036086 after 880 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 960 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.027806 after 960 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 1040 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.010229 after 1040 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
[mcmc] Checkpoint: 1120 samples accepted.
[mcmc] Ready to check convergence and learn a new proposal covmat
[mcmc] Convergence of means: R-1 = 0.030713 after 1120 accepted steps
[mcmc] Updated covariance matrix of proposal pdf.
```

In [12]:
```
# Load chains in GetDist
from getdist.mcsamples import loadCobayaSamples
gdsamples = loadCobayaSamples(updated_info, products["sample"])

# Plot
import getdist.plots as gdplt
gdplot = gdplt.getSubplotPlotter(subplot_size=4)
gdplot.triangle_plot(gdsamples, ["log10_freq_peak", "log10_OmMax"], filled=T
rue)
print "Covariace matrix:\n", gdsamples.getCovMat().matrix[:2,:2]
```

```
[root] *WARNING* outlier fraction 0.0100961538462
Covariace matrix:
[[ 0.0044914  -0.00626477]
 [-0.00626477  0.01446813]]
```



In [ ]:
```
```