hasasia Documentation

Release 1.2.3

Jeffrey S. Hazboun

Apr 04, 2023

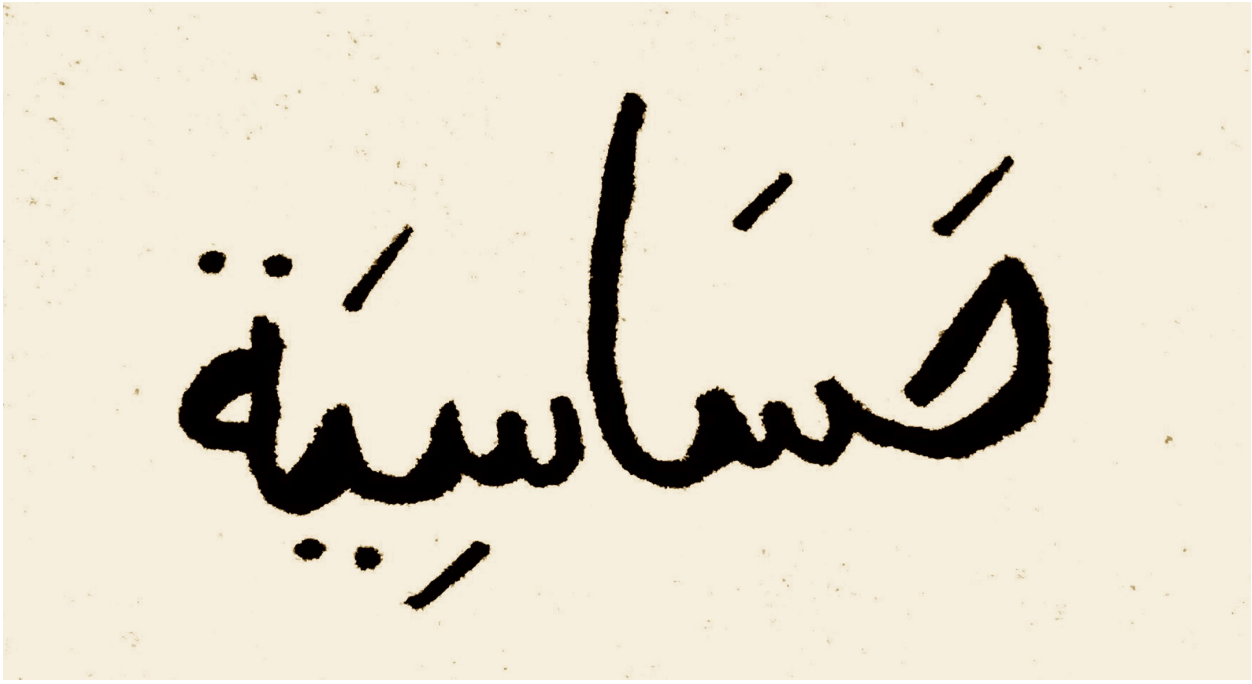
CONTENTS:

CHAPTER

ONE

HASASIA

A Python package to calculate gravitational-wave sensitivity curves for pulsar timing arrays.



(hasasia) is Arabic for sensitivity . Image Credit: Reem Tasyakan

- Free software: MIT license
- Documentation: https://hasasia.readthedocs.io.

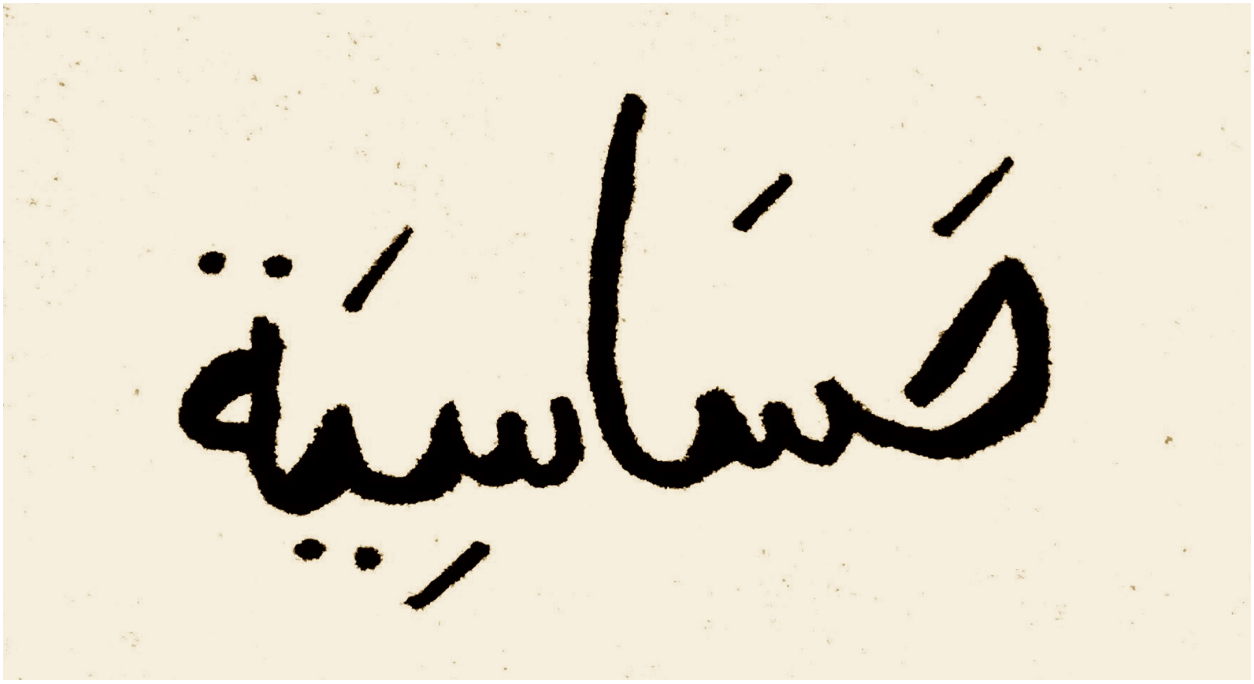
1.1 Features

Calculates the following structures needed for signal analysis with pulsars:

- Pulsar transmission functions
- Inverse-noise-weighted transmission functions
- Individual pulsar sensitivity curves.
- Pulsar timing array sensitivity curves as characteristic strain, strain sensitivity or energy density.
- Power-law integrated sensitivity curves.
- Sensitivity sky maps for pulsar timing arrays

1.1.1 hasasia

A Python package to calculate gravitational-wave sensitivity curves for pulsar timing arrays.



(hasasia) is Arabic for sensitivity . Image Credit: Reem Tasyakan

- Free software: MIT license
- Documentation: https://hasasia.readthedocs.io.

Features

Calculates the following structures needed for signal analysis with pulsars:

- Pulsar transmission functions
- Inverse-noise-weighted transmission functions
- Individual pulsar sensitivity curves.
- Pulsar timing array sensitivity curves as characteristic strain, strain sensitivity or energy density.
- Power-law integrated sensitivity curves.
- Sensitivity sky maps for pulsar timing arrays

Getting Started

hasasia is on the Python Package Inventory, so the easiest way to get started is by using *pip* to install:

```
pip install hasasia
```

The pulsar and spectrum objects are used to build sensitivity curves for full PTAs. The Spectrum object has all of the information needed for the pulsar.

```
import hasasia.sensitivity as hsen

toas = np.arange(54378,59765,22) #Choose a range of times-of-arrival
toaerrs = 1e-7*np.ones_like(toas) #Set all errors to 100 ns
psr = hsen.Pulsar(toas=toas,toaerrs=toaerrs)
spec = hsen.Spectrum(psr)
```

Publication

This work is featured in a publication, currently released on the arXiv. If you would like to reference the formalism used in this work please use the following attribution:

```
@article{Hazboun:2019vhv,
            author = {{Hazboun}, Jeffrey S. and {Romano}, Joseph D. and {Smith}, Tristan␣
↪L.},
             title = "{Realistic sensitivity curves for pulsar timing arrays}",
           journal = {\prd},
          keywords = {General Relativity and Quantum Cosmology, Astrophysics -␣
↪Instrumentation and Methods for Astrophysics},
              year = 2019,
             month = nov,
            volume = {100},
            number = {10},
               eid = {104028},
             pages = {104028},
               doi = {10.1103/PhysRevD.100.104028},
      archivePrefix = {arXiv},
            eprint = {1907.04341},
      primaryClass = {gr-qc},
            adsurl = {https://ui.adsabs.harvard.edu/abs/2019PhRvD.100j4028H},
```

(continues on next page)

(continued from previous page)

```
          adsnote = {Provided by the SAO/NASA Astrophysics Data System}
        }
```

Otherwise if you would like to reference the Python package use the following citation:

```
@article{Hazboun2019Hasasia,
               journal      = {Journal of Open Source Software},
               doi          = {10.21105/joss.01775},
               issn         = {2475-9066},
               number       = {42},
               publisher    = {The Open Journal},
        title        = {Hasasia: A Python package for Pulsar Timing Array Sensitivity␣
↪Curves},
        url          = {http://dx.doi.org/10.21105/joss.01775},
               volume       = {4},
               author       = {Hazboun, Jeffrey and Romano, Joseph and Smith, Tristan},
               pages        = {1775},
               date         = {2019-10-23},
               year         = {2019},
               month        = {10},
               day          = {23},
        }
```

Credits

Development Team: Jeffrey S. Hazboun, Joseph D. Romano and Tristan L. Smith

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

1.1.2 Installation

Stable release

To install hasasia, run this command in your terminal:

```
$ pip install hasasia
```

This is the preferred method to install hasasia, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for hasasia can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/Hazboun6/hasasia
```

Or download the tarball:

```
$ curl  -OL https://github.com/Hazboun6/hasasia/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

1.1.3 Getting Started

To use hasasia, first import these useful modules:

```
import numpy as np
import hasasia.sensitivity as sens
import hasasia.sim as sim
```

The simplest way to get started is by making a set of simulated pulsars, all with the same parameters, except the sky positions:

```
phi = np.random.uniform(0, 2*np.pi,size=34)
theta = np.random.uniform(0, np.pi,size=34)

psrs = sim.sim_pta(timespan=11.4, cad=23, sigma=1e-7,
                   phi=phi, theta=theta, Npsrs=34)
```

The *sim.sim_pta* method can take single values or a list/array of timespans [yrs], cadences [1/yr], TOA errors [sec] and sky locations [rad].

Next make a spectra object for each pulsar. Here we calculate the inverse-noise-weighted transmission function along the way:

```
freqs = np.logspace(np.log10(5e-10),np.log10(5e-7),400)
spectra = []
for p in psrs:
     sp = sens.Spectrum(p, freqs=freqs)
     sp.NcalInv
     spectra.append(sp)
```

Enter the list of spectra into the GWB and deterministic sensitivity curve classes.:

```
scGWB = sens.GWBSensitivityCurve(spectra)
scDeter = sens.DeterSensitivityCurve(spectra)
```

Compare this to a set of sensitivity curves made with 3-year pulsar baselines:

```
psrs2 = sim.sim_pta(timespan=3.0, cad=23, sigma=1e-7,
                    phi=phi, theta=theta, Npsrs=34)

spectra2 = [sens.Spectrum(p, freqs=freqs) for p in psrs2]
scGWB2 = sens.GWBSensitivityCurve(spectra2)
```

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

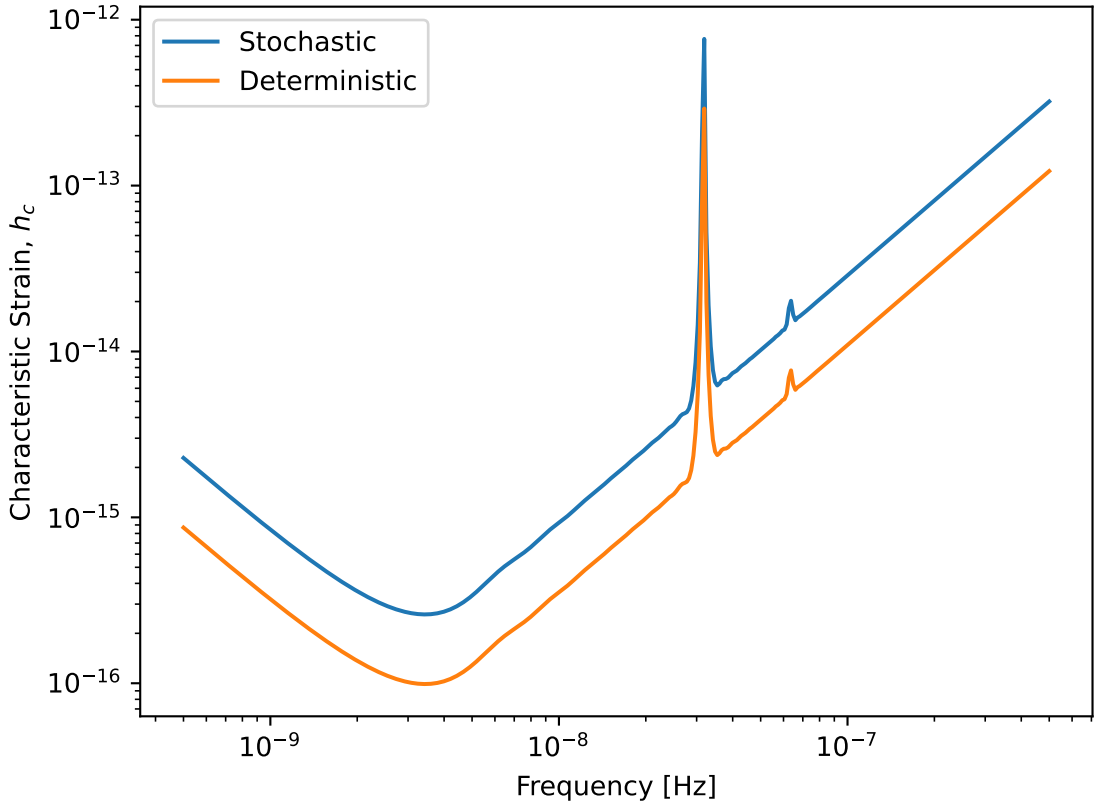
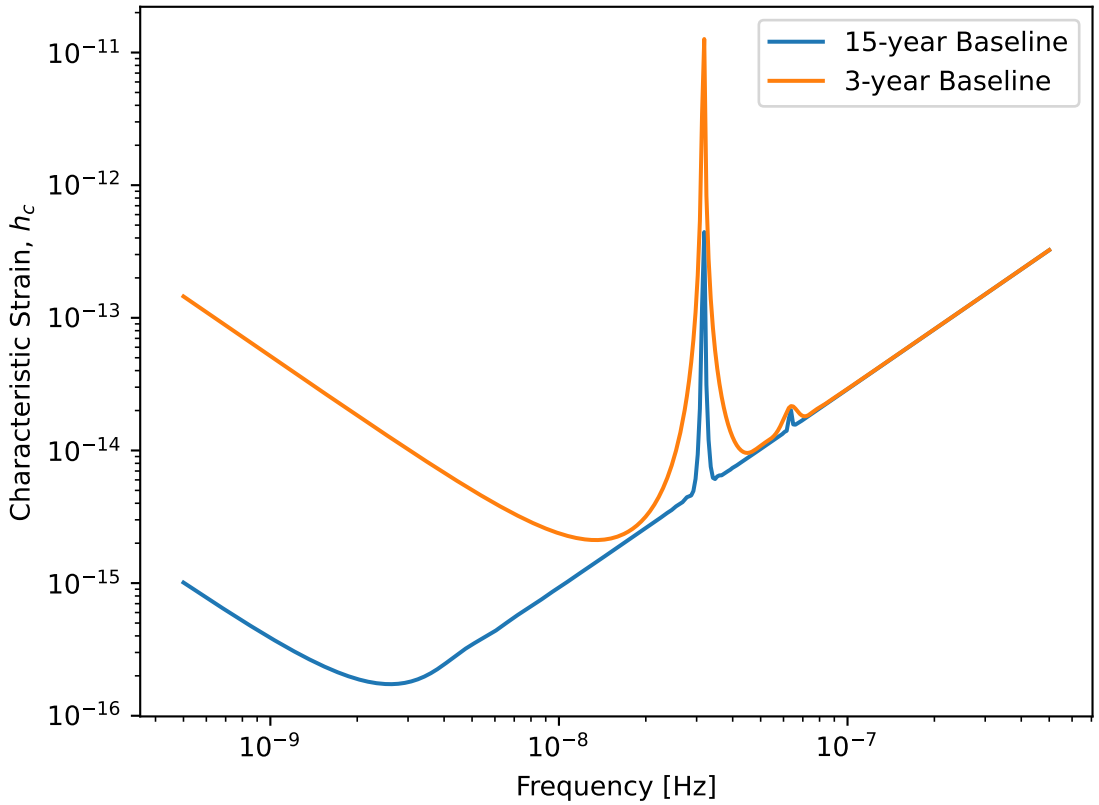


Fig. 1: Comparison of a sensitivity curve for a deterministic and stochastic gravitational wave signal.


15-year Baseline
3-year Baseline
Characteristic Strain, h_c
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
10^{-9}
10^{-8}
10^{-7}
Frequency [Hz]

1.1.4 GWBSensitivity and DeterSensitivity Tutorial

This tutorial is an introduction to the `sensitivity` module of the pulsar timing array sensitivity curve package `hasasia`. For an introduction to sensitivity sky maps see later tutorials.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import hasasia.sensitivity as hsen
import hasasia.sim as hsim
```

```
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 300
mpl.rcParams['figure.figsize'] = [5,3]
mpl.rcParams['text.usetex'] = True
```

After importing various useful packages, including the needed `hasasia` submodules, and setting some `matplotlib` preferences, we instantiate 34 pulsar positions and timespans.

```
phi = np.random.uniform(0, 2*np.pi,size=34)
cos_theta = np.random.uniform(-1,1,size=34)
#This ensures a uniform distribution across the sky.
theta = np.arccos(cos_theta)
```

```
timespan=[11.4 for ii in range(10)]
timespan.extend([3.0 for ii in range(24)])
```

The simplest way to build a sensitivity curve is to use the `hasasia.sim` module to make a list of `hasasia.sensitivity.Pulsar` objects. One can use single values, a list/array of values or a mix for the parameters.

```
psrs = hsim.sim_pta(timespan=timespan, cad=23, sigma=1e-7,
                    phi=phi,theta=theta)
```

If red (time-correlated) noise is desired for the pulsars then one first needs to define a frequency array (in [Hz]) over-which to calculate the red noise spectrum.

```
freqs = np.logspace(np.log10(5e-10),np.log10(5e-7),500)
```

```
psrs2 = hsim.sim_pta(timespan=timespan,cad=23,sigma=1e-7,
                     phi=phi,theta=theta,
                     A_rn=6e-16,alpha=-2/3.,freqs=freqs)
```

These lists of pulsars are then used to make a set of `hasasia.sensitivity.Spectrum` objects. These objects either build an array of frequencies, or alternatively take an array of frequencies, over which to calculate the various spectra.

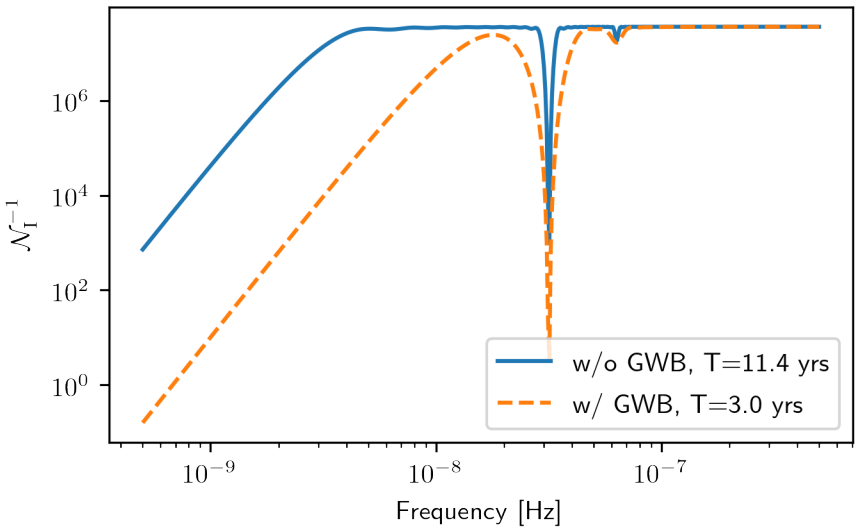
All frequency arrays need to match across spectra and red noise realizations.

```
spectra = []
for p in psrs:
    sp = hsen.Spectrum(p, freqs=freqs)
    sp.NcalInv
    spectra.append(sp)
```

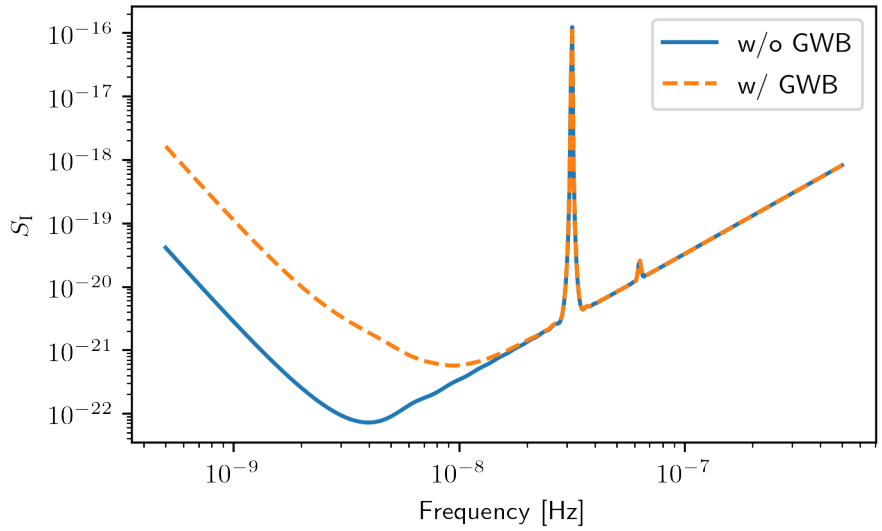
```
spectra2 = []
for p in psrs2:
    sp = hsen.Spectrum(p, freqs=freqs)
    sp.NcalInv
    spectra2.append(sp)
```

Each spectra contains a number of attributes for that particular pulsar, including the inverse-noise-weighted transmission function, and sensitivity curve.

```
plt.loglog(spectra[0].freqs,spectra[0].NcalInv,
           label='w/o GWB, T={0} yrs'.format(timespan[0]))
plt.loglog(spectra2[20].freqs,spectra2[20].NcalInv,'--',
           label='w/ GWB, T={0} yrs'.format(timespan[20]))
plt.xlabel('Frequency [Hz]')
plt.ylabel(r'$\mathcal{N}^{-1}_{\rm I}$')
plt.legend()
plt.show()
```




```
plt.loglog(spectra[0].freqs,spectra[0].S_I,label='w/o GWB')
plt.loglog(spectra2[0].freqs,spectra2[0].S_I,'--',label='w/ GWB')
plt.xlabel('Frequency [Hz]')
plt.ylabel(r'$S_{\rm I}$')
plt.legend()
plt.show()
```

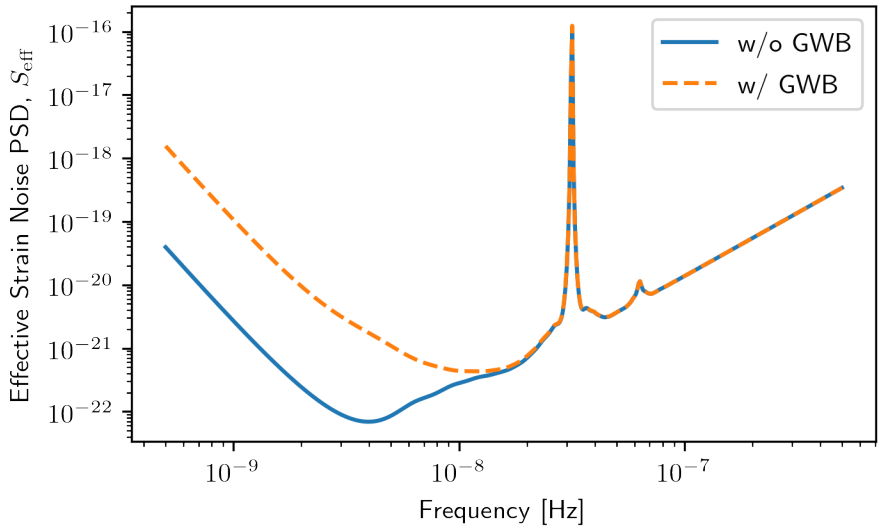


Senstivity Curves

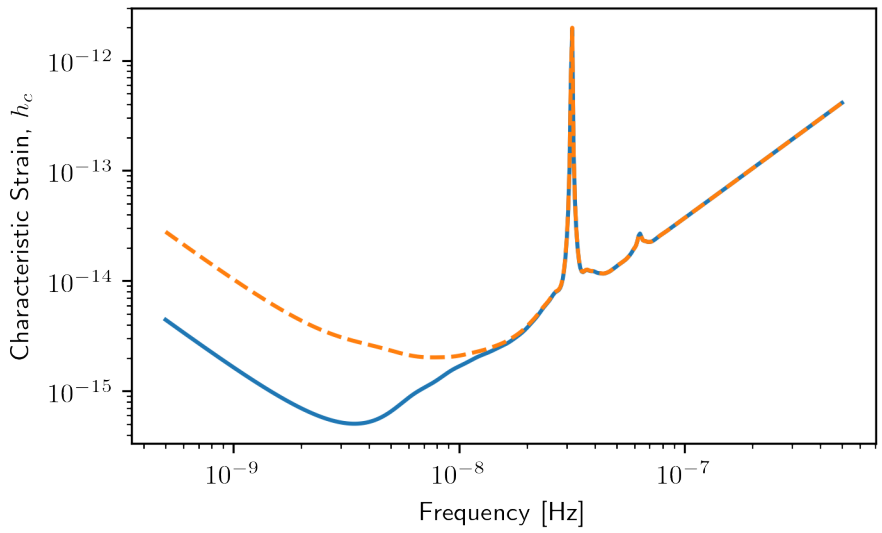
The list of spectra are then the input for the sensitivity curve classes.

```
sc1a = hsen.GWBSensitivityCurve(spectra)
sc1b = hsen.DeterSensitivityCurve(spectra)
sc2a = hsen.GWBSensitivityCurve(spectra2)
sc2b = hsen.DeterSensitivityCurve(spectra2)
```

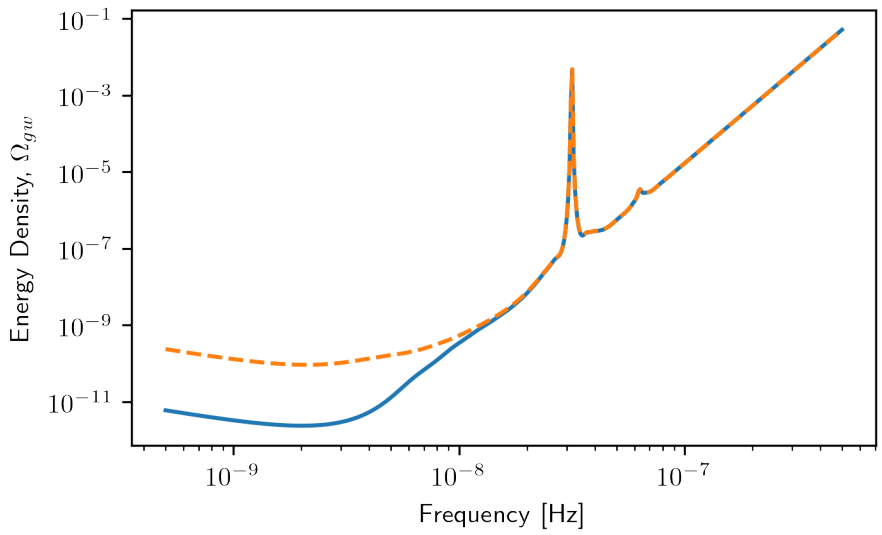
```
plt.loglog(sc1a.freqs,sc1a.S_eff,label='w/o GWB')
plt.loglog(sc2a.freqs,sc2a.S_eff,'--',label='w/ GWB')
plt.xlabel('Frequency [Hz]')
plt.ylabel(r'Effective Strain Noise PSD, $S_{\rm eff}$')
plt.legend()
plt.show()
```



```
plt.loglog(sc1a.freqs,sc1a.h_c)
plt.loglog(sc2a.freqs,sc2a.h_c,'--')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain, $h_c$')
plt.show()
```




```
plt.loglog(sc1a.freqs,sc1a.Omega_gw)
plt.loglog(sc2a.freqs,sc2a.Omega_gw,'--')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Energy Density, $\Omega_{gw}$')
plt.show()
```

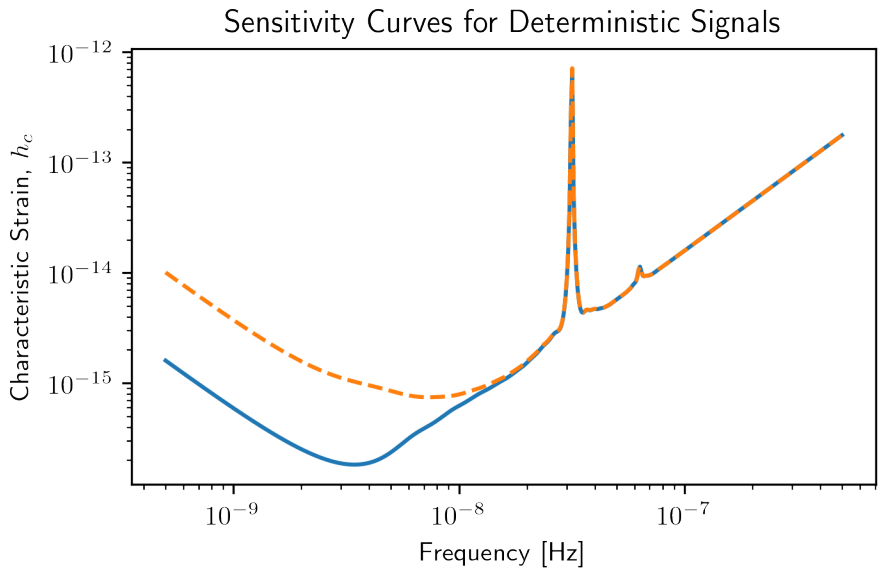



```
plt.loglog(sc1b.freqs,sc1b.h_c)
plt.loglog(sc2b.freqs,sc2b.h_c,'--')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain, $h_c$')
```

(continues on next page)

(continued from previous page)

```
plt.title('Sensitivity Curves for Deterministic Signals')
plt.show()
```

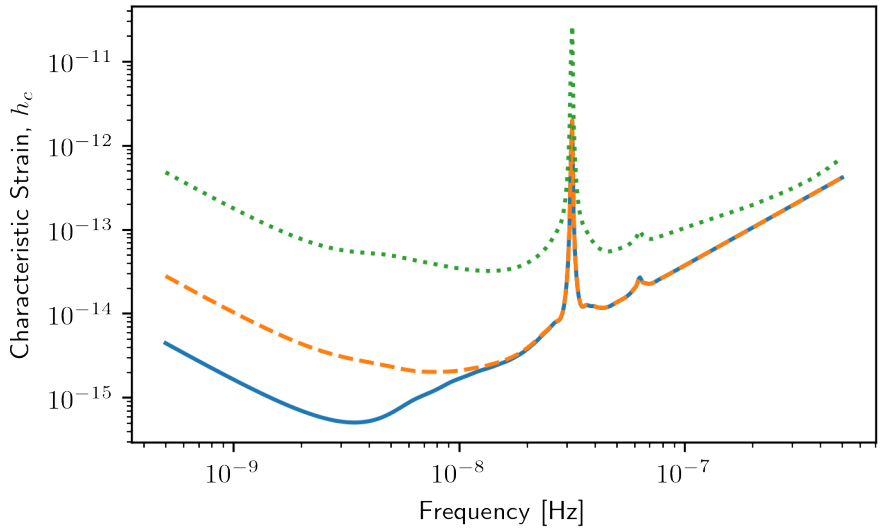


Multiple Values for Red Noise

One can give each pulsar its own value for the red noise power spectrum.

```
A_rn = np.random.uniform(1e-16,1e-12,size=phi.shape[0])
alphas = np.random.uniform(-3/4,1,size=phi.shape[0])

psrs3 = hsim.sim_pta(timespan=timespan,cad=23,sigma=1e-7,
                     phi=phi,theta=theta,
                     A_rn=A_rn,alpha=alphas,freqs=freqs)

spectra3 = []
for p in psrs3:
    sp = hsen.Spectrum(p, freqs=freqs)
    sp.NcalInv
    spectra3.append(sp)

sc3a=hsen.GWBSensitivityCurve(spectra3)
sc3b=hsen.DeterSensitivityCurve(spectra3)
```

```
plt.loglog(sc1a.freqs,sc1a.h_c)
plt.loglog(sc2a.freqs,sc2a.h_c,'--')
plt.loglog(sc3a.freqs,sc3a.h_c,':')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain, $h_c$')
plt.show()
```



Power Law-Integrated Sensitivity Curves

There are a few additional functions in the `hasasia.senstivity` module for calculating a power law-integrated noise curve for stochastic senstivity curves.

First we use the `Agwb_from_Seff_plaw` method to calculate the amplitude of a GWB needed to obtain an SNR=3 with the usual spectral index, which is the default value of the spectral index.

```
hgw = hsen.Agwb_from_Seff_plaw(sc1a.freqs, Tspan=sc1a.Tspan, SNR=3,
                               S_eff=sc1a.S_eff)

#We calculate the power law across the frequency range for plotting.
fyr = 1/(365.25*24*3600)
plaw_h = hgw*(sc1a.freqs/fyr)**(-2/3)
```

The `Agwb_from_Seff_plaw` is used by the `PI_hc` method to conveniently calculate the power law-integrated sensitivity across a frequency range of the user's choice. The method returns the PI sensitivity curve and the set of power law values solved for in the process.

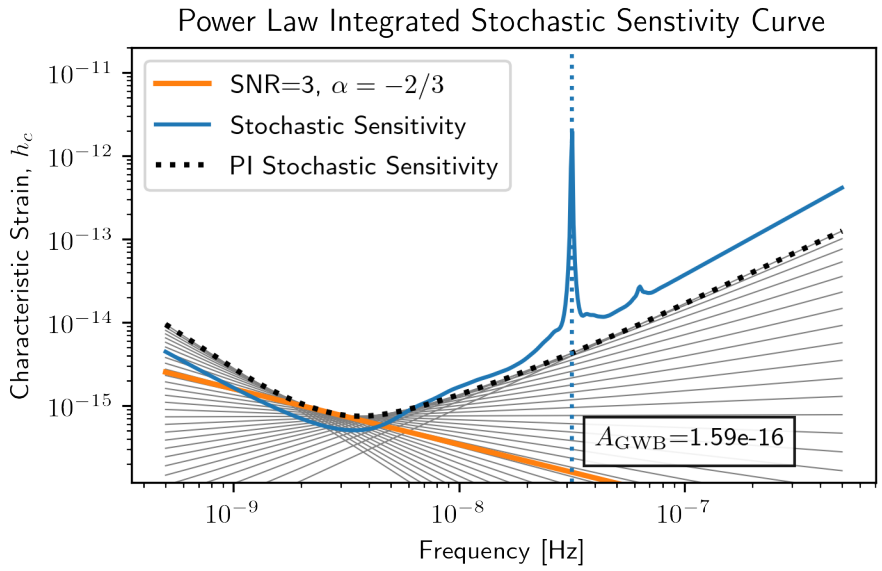
```
PI_sc, plaw = hsen.PI_hc(freqs=sc1a.freqs, Tspan=sc1a.Tspan,
                         SNR=3, S_eff=sc1a.S_eff, N=30)
```

```
for ii in range(plaw.shape[1]):
    plt.loglog(sc1a.freqs,plaw[:,ii],
               color='gray',lw=0.5)
plt.loglog(sc1a.freqs,plaw_h,color='C1',lw=2,
           label=r'SNR=3, $\alpha=-2/3$')
plt.loglog(sc1a.freqs,sc1a.h_c, label='Stochastic Sensitivity')
plt.loglog(sc1a.freqs,PI_sc, linestyle=':',color='k',lw=2,
           label='PI Stochastic Sensitivity')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain, $h_c$')
plt.axvline(fyr,linestyle=':')
plt.title('Power Law Integrated Stochastic Senstivity Curve')
```

(continues on next page)

(continued from previous page)

```
plt.ylim(hgw*0.75,2e-11)
plt.text(x=4e-8,y=3e-16,
         s=r'$A_{\rm GWB}$='+'{0:1.2e}'.format(hgw),
         bbox=dict(facecolor='white', alpha=0.9))
plt.legend(loc='upper left')
plt.show()
```

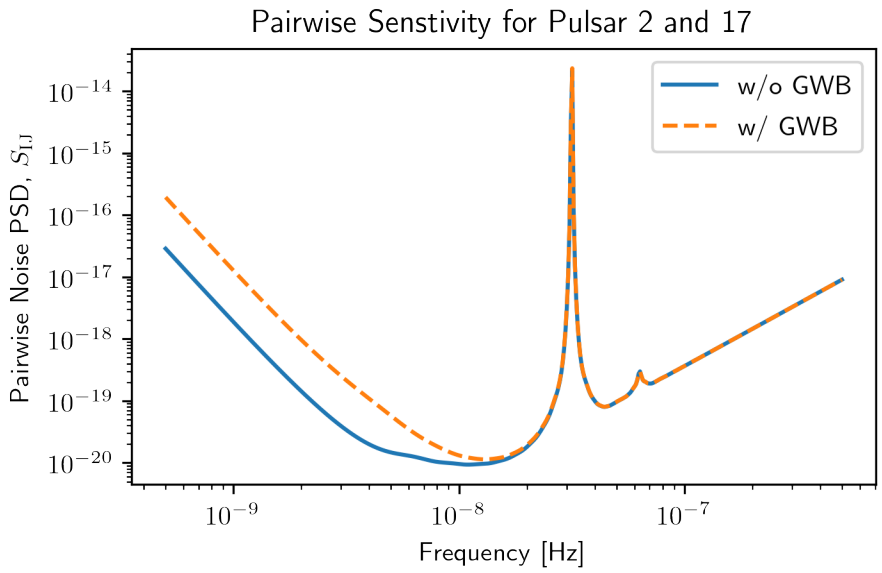


Here we see a fairly optimistic sensitivity at the SNR=3 threshold since this PTA is made up of 100 ns-precision pulsars with no red noise.

Pairwise Senstivity Curves

One can also access each term of the series in the calculation of the stochastic effective sensitivity curve. Each unique pair is available in the `GWBSensitivity.S_effIJ` attribute. The pulsars can be identified using the `GWBSensitivity.pairs` attribute.

```
plt.loglog(sc1a.freqs,sc1a.S_effIJ[79],label='w/o GWB')
plt.loglog(sc2a.freqs,sc2a.S_effIJ[79],'--',label='w/ GWB')
plt.xlabel('Frequency [Hz]')
plt.ylabel(r'Pairwise Noise PSD, $S_{\rm IJ}$')
p1,p2 = sc1a.pairs[:,79]
plt.title(r'Pairwise Senstivity for Pulsar {0} and {1}'.format(p1,p2))
plt.legend()
plt.show()
```



Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.1.5 SkySensitivity Tutorial

This tutorial is an introduction to the skymap module of the pulsar timing array sensitivity curve package hasasia. For an introduction to straight forward sensitivity curves see prior tutorials.

```
#Import the usual suspects.
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
# You'll need these packages to make the skymaps and deal with units.
import healpy as hp
import astropy.units as u
import astropy.constants as c
```

```
#Import the needed modules.
import hasasia.sensitivity as hsen
import hasasia.sim as hsim
import hasasia.skymap as hsky
```

```
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 300
mpl.rcParams['figure.figsize'] = [5,3]
mpl.rcParams['text.usetex'] = True
```

```
#Make a set of random sky positions
phi = np.random.uniform(0, 2*np.pi,size=33)
cos_theta = np.random.uniform(-1,1,size=33)
theta = np.arccos(cos_theta)
```

(continues on next page)

(continued from previous page)

```
#Adding one well-placed sky position for plots.
phi = np.append(np.array(np.deg2rad(60)),phi)
theta = np.append(np.array(np.deg2rad(50)),theta)

#Define the timsespans and TOA errors for the pulsars
timespans = np.random.uniform(3.0,11.4,size=34)
Tspan = timespans.max()*365.25*24*3600
sigma = 1e-7 # 100 ns
```

Here we use the `sim_pta` method in the `hasasia.sim` module to simulate a set of `hasasia.senstivity.Pulsar` objects. This function takes either single values or lists/array as inputs for the set of pulsars.

```
#Simulate a set of identical pulsars, with different sky positions.
psrs = hsim.sim_pta(timespan=11.4, cad=23, sigma=sigma,
                    phi=phi, theta=theta)
```

Next define the frequency range over which to characterize the spectra for the pulsars and enter each `Pulsar` object into a `hasasia.sensitivity.Spectrum` object.

```
freqs = np.logspace(np.log10(1/(5*Tspan)),np.log10(2e-7),500)
spectra = []
for p in psrs:
    sp = hsen.Spectrum(p, freqs=freqs)
    sp.NcalInv
    spectra.append(sp)
```

Note above that we have called `sp.NcalInv`, which calculates the inverse-noise-weighted transmission function for the pulsar along the way. For realistic pulsars with +100k TOAs this step will take the most time.

Define a SkySensitivity Object

Before defining a `hasasia.skymap.SkySensitivity` object we will need to choose a set of sky locations. Here we use the `healpy` Python package to give us a healpix pixelation of the sky.

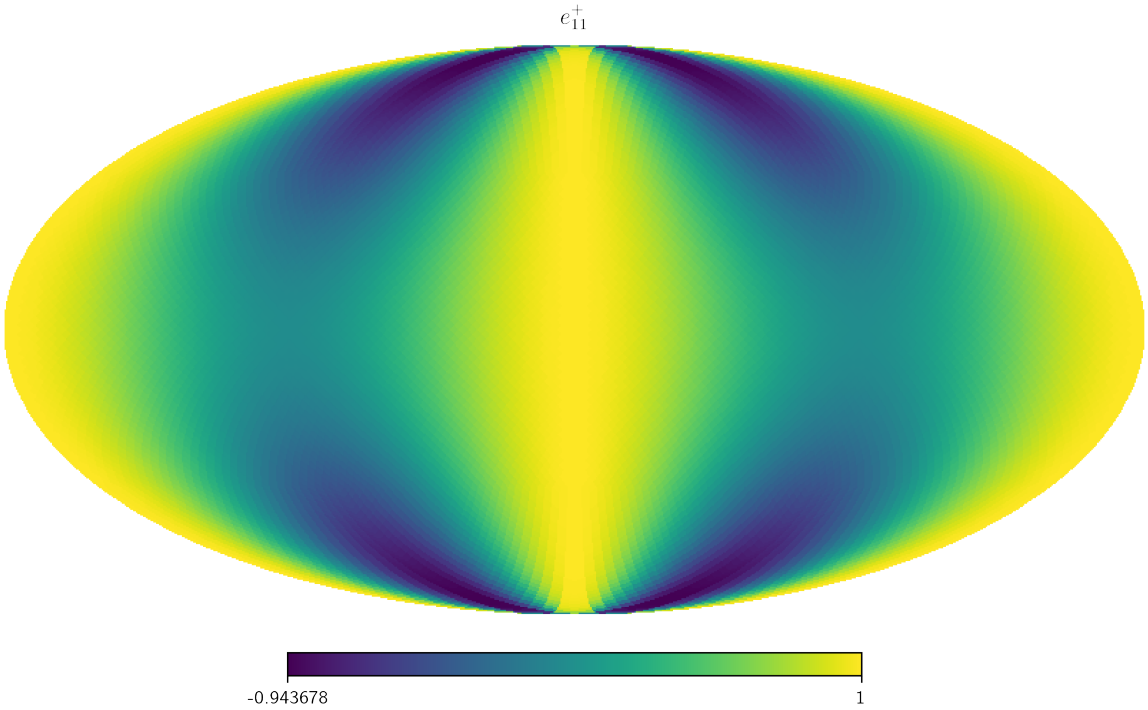
```
#Use the healpy functions to get the sky coordinates
NSIDE = 32
NPIX = hp.nside2npix(NSIDE)
IPIX = np.arange(NPIX)
theta_gw, phi_gw = hp.pix2ang(nside=NSIDE,ipix=IPIX)
```

Next enter the list of `Spectrum` objects and the sky coordinates into the `SkySensitivity` class.

```
SM=hsky.SkySensitivity(spectra,theta_gw, phi_gw)
```

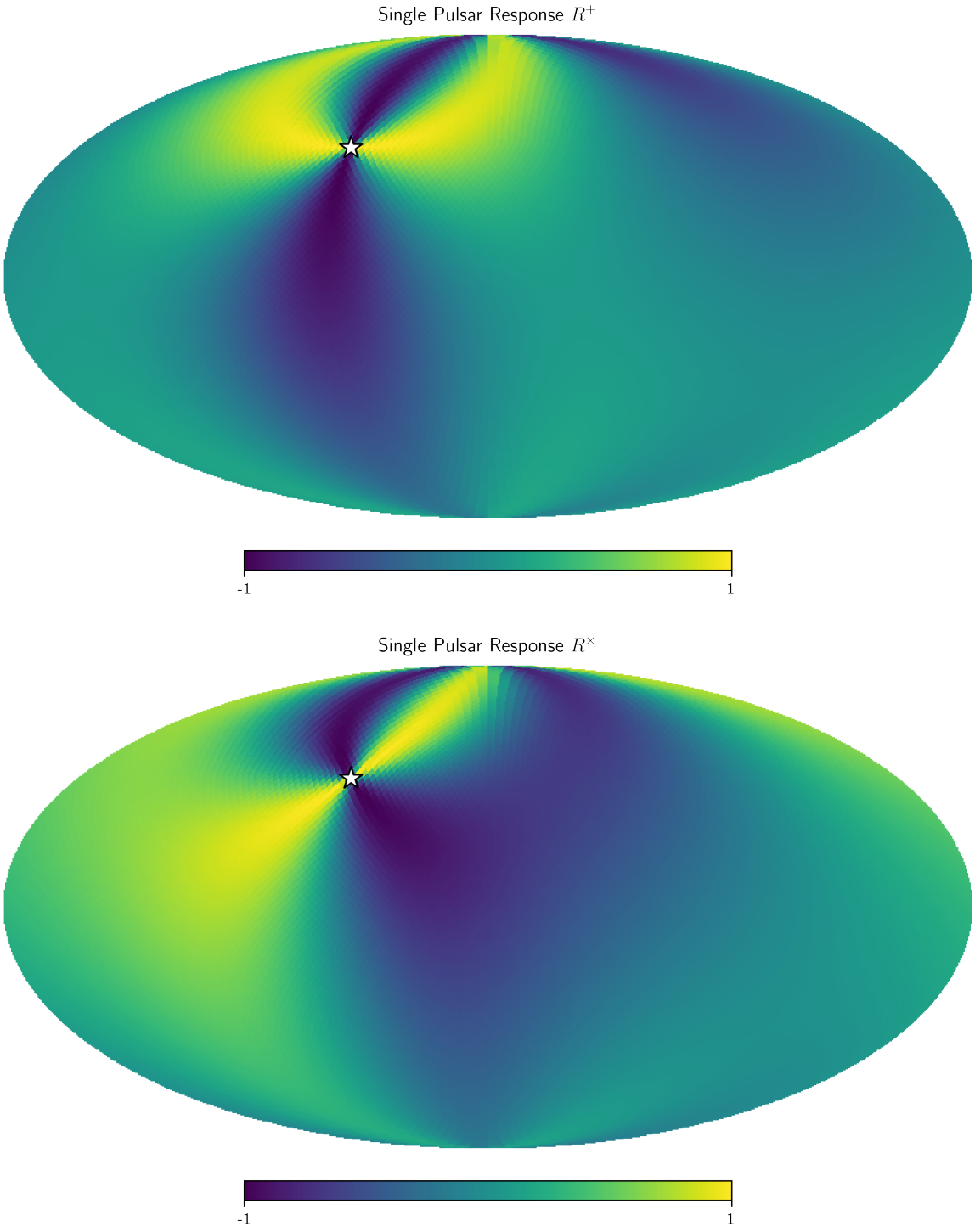
The `SkySensitivity` class has a number of accessible attributes and methods. The polarization tensors :math:`e^+` and :math:`e^-` are available.

```
hp.mollview(SM.eplus[1,1,:], title='$e_{11}^+$',)
```



One can also access the residual response functions for each of the individual pulsars, as `SkySensitivity.Fplus` and `SkySensitivity.Fcross`.

```
idx = 0
hp.mollview(SM.Fplus[idx], fig=1,
            title="Single Pulsar Response $R^+$",min=-1,max=1)
hp.visufunc.projscatter(SM.thetas[idx],SM.phis[idx],
                        marker='*',color='white',
                        edgecolors='k',s=200)
hp.mollview(SM.Fcross[idx], fig=2,
            title=r"Single Pulsar Response $R^\times$",min=-1,max=1)
hp.visufunc.projscatter(SM.thetas[idx],SM.phis[idx],
                        marker='*',color='white',
                        edgecolors='k',s=200)

plt.show()
```



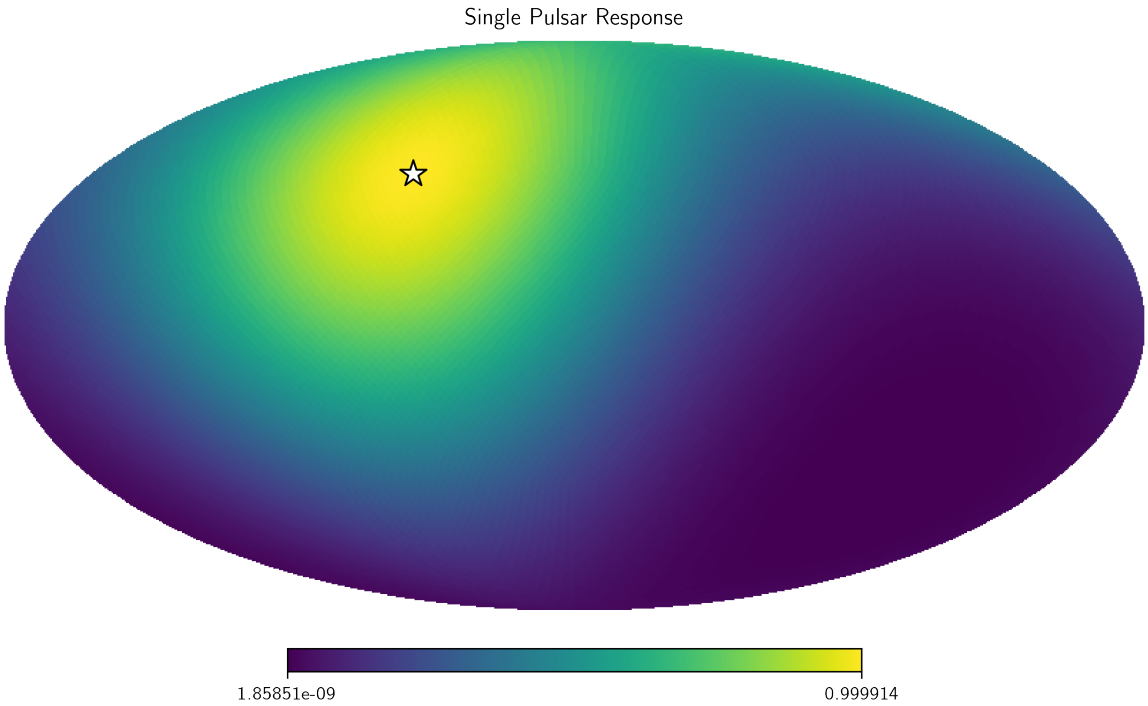
And the full residual response as `SkySensitivity.sky_response`.

```
idx =0
hp.mollview(SM.sky_response[idx], title="Single Pulsar Response")
hp.visufunc.projscatter(SM.thetas[idx], SM.phis[idx],
                        marker='*',color='white',
                        edgecolors='k',s=200)
```

(continues on next page)

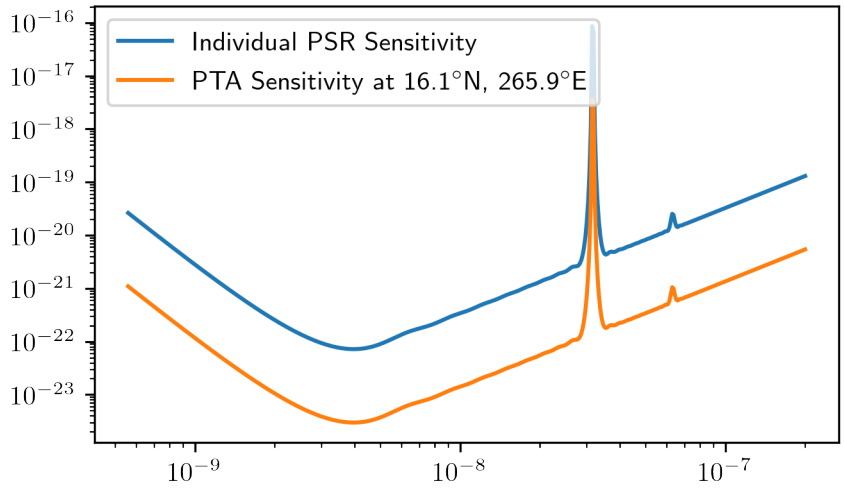
(continued from previous page)

```
plt.show()
```



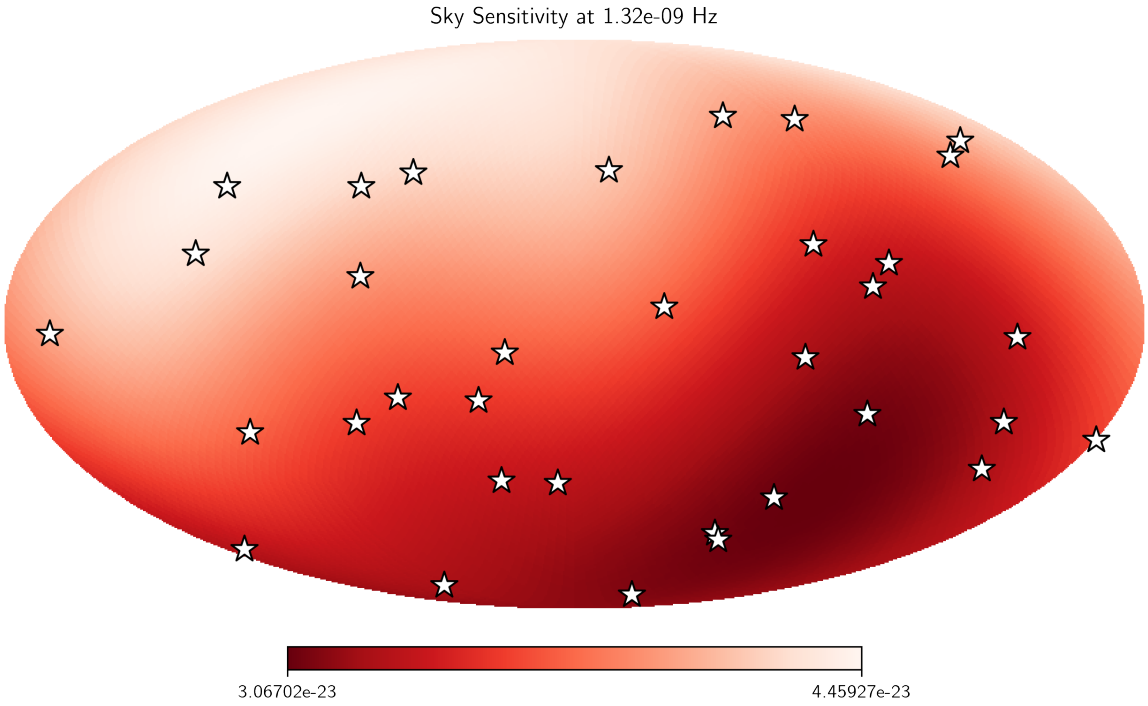
The full frequency and sky location sensitivity information is available as `SkySensitivity.S_effSky`. The first index is across frequency, while the second index is across sky position. Here we compare the sensitivity from an individual pulsar to the full PTA's senstivity at a particular sky position.

```
sky_loc = 'PTA Sensitivity at '
sky_loc += '{0:2.1f}$^\circ$N, {1:2.1f}$^\circ$E'.format(np.rad2deg(theta_gw[252]),
                                                        np.rad2deg(phi_gw[252]))
plt.loglog(SM.freqs,spectra[0].S_I, label='Individual PSR Sensitivity')
plt.loglog(SM.freqs,SM.S_effSky[:,252],
           label=sky_loc)
plt.legend(loc='upper left')
plt.show()
```



Here we plot the `SkySensitivity.S_effSky` across the sky at a given frequency.

```
idx = 73
hp.mollview(SM.S_effSky[idx],
            title="Sky Sensitivity at {0:2.2e} Hz".format(SM.freqs[idx]),
            cmap='Reds_r')
hp.visufunc.projscatter(SM.thetas,SM.phis,
                        marker='*',color='white',
                        edgecolors='k',s=200)
plt.show()
```



Calculating SNR across the Sky

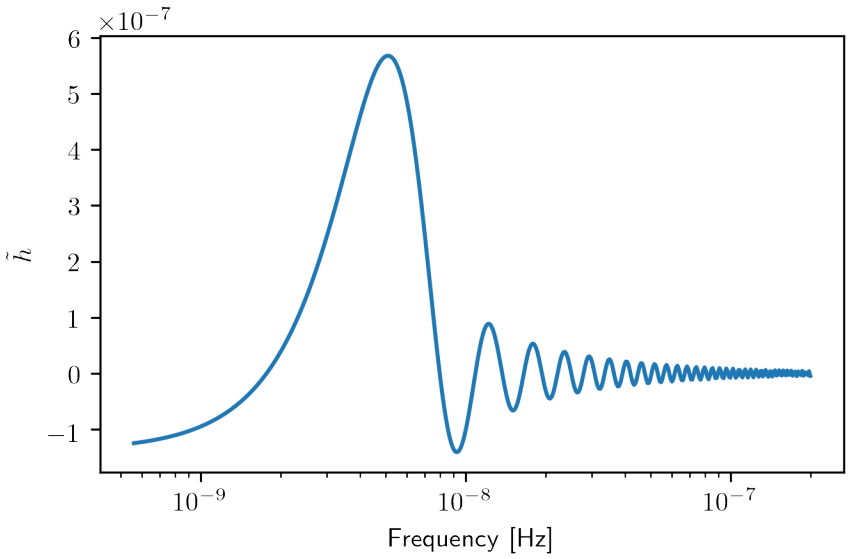
The `SkySensitivity.S_effSky` class comes with a method for calculating the signal-to-noise ratio for a given signal. Rather than calculate a signal from a single sky position, the method will calculate the SNR from every sky position initially provided, given a particular signal provided in strain across the frequency band.

There is a convenience function for circular binaries provided as `hasasia.skymap.h_circ`.

```
hCirc = hsky.h_circ(1e9,200,5e-9,Tspan,SM.freqs).to('')
```

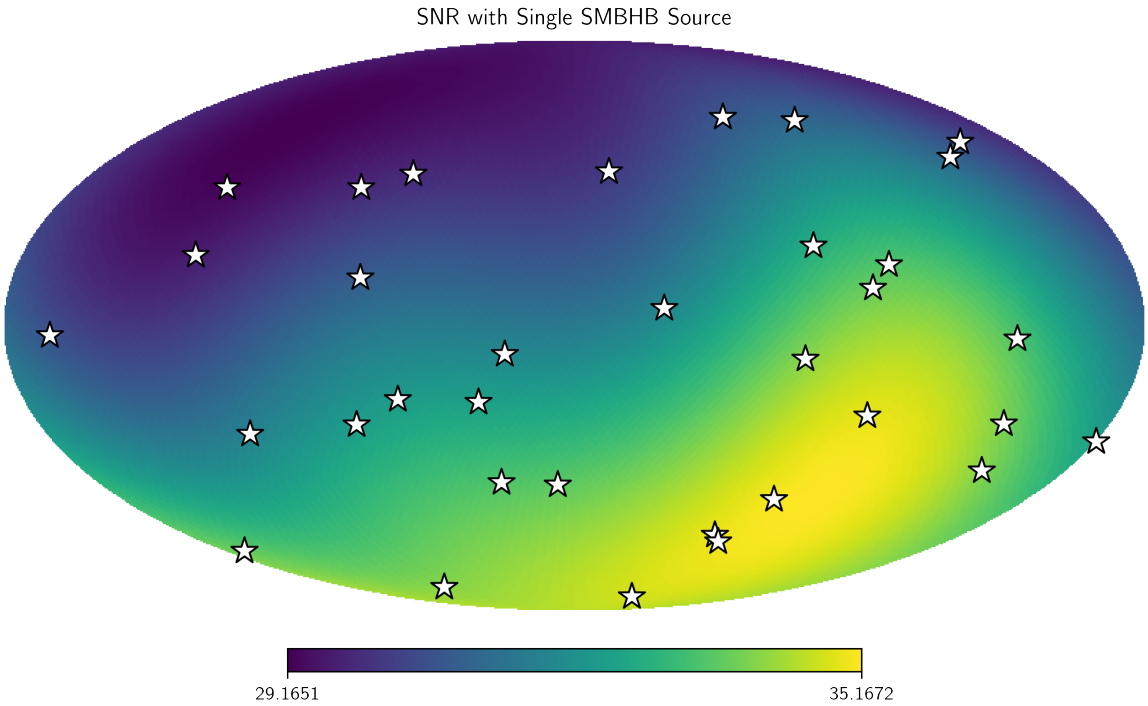
Here we plot the signal in the frequency domain, for a finite integration time provided as the time span of the data set.

```
plt.semilogx(SM.freqs, hCirc)
plt.xlabel('Frequency [Hz]')
plt.ylabel(r'$\tilde{h}$')
plt.show()
```




```
SNR = SM.SNR(hCirc.value[167])
```

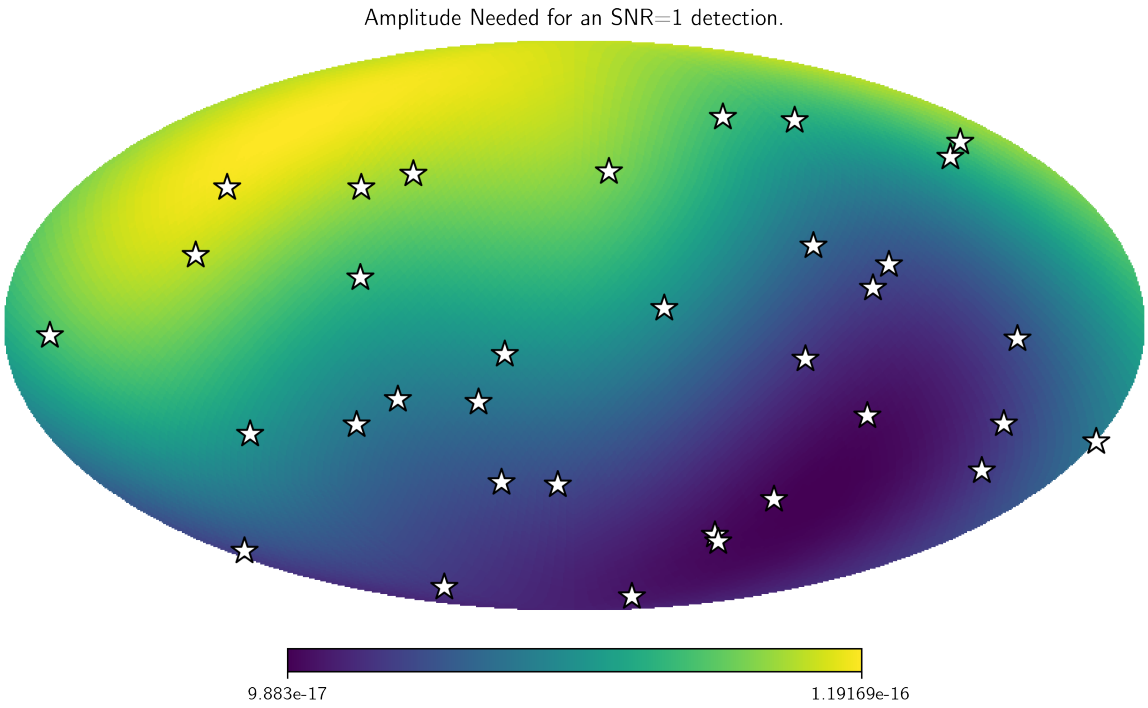
```
idx = 167
hp.mollview(SNR[idx],
            title="SNR with Single SMBHB Source",
            cmap='viridis')
hp.visufunc.projscatter(SM.thetas,SM.phis,marker='*',
                        color='white',edgecolors='k',s=200)
plt.show()
```




```
h_divA = (hsky.h_circ(1e9,200,5e-9,Tspan,SM.freqs)
          /hsky.h0_circ(1e9,200,5e-9)).value
```

```
Amp = SM.A_gwb(h_divA)
```

```
hp.mollview(Amp,
            title="Amplitude Needed for an SNR=1 detection.",
            cmap='viridis')
hp.visufunc.projscatter(SM.thetas,SM.phis,marker='*',
                        color='white',edgecolors='k',s=200)
plt.show()
```



Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.1.6 Real Pulsar Timing Array Data Tutorial

Here we present the details of using real `par/tim` files, as often provided publicly by Pulsar Timing Array collaborations, to construct senstivity curves. The `hasasia` code base deals with real data as easily as simulated date. The most challenging part of the code will be getting the data into a format compatible with `hasasia`.

First we import some important modules.

```
import numpy as np
import scipy.linalg as sl
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import glob, pickle, json
```

```
import hasasia.sensitivity as hsen
import hasasia.sim as hsim
import hasasia.skymap as hsky
```

```
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 300
mpl.rcParams['figure.figsize'] = [5,3]
mpl.rcParams['text.usetex'] = True
```

Making `hasasia.Pulsar` objects with real data.

Here we use the Python-based pulsar timing data analysis package enterprise. We choose this software, as it has a Pulsar class well suited for dealing with the needs of our code base. In particular the pulsar TOAs, errors, and sky positions are handled cleanly. The sky positions are converted into ecliptic longitude and co-latitude, which are the angles taken as input by `hasasia`.

```
from enterprise.pulsar import Pulsar as ePulsar
```

The following paths point to the location of the data files (tim), parameter files (par) and noise files.

```
pardir = '/Users/hazboun/GoogleDrive/NANOGrav_Detection/data/nanograv/11yr_v2/'
timdir = '/Users/hazboun/GoogleDrive/NANOGrav_Detection/data/nanograv/11yr_v2/'
noise_dir = '/Users/hazboun/GoogleDrive/NANOGrav_Detection/11yr_stochastic_analysis'
noise_dir += '/nano11y_data/noisefiles/'
pars = sorted(glob.glob(pardir+'*.par'))
tims = sorted(glob.glob(timdir+'*.tim'))
noise_files = sorted(glob.glob(noise_dir+'*.json'))
```

Here we load the list of pulsars included in the NANOGrav 11-year analysis. This includes the pulsars in the dataset with longer than 3-year baselines.

```
psr_list = np.load('/Users/hazboun/GoogleDrive/NANOGrav_Detection/SlicedData/PSR_by_Obs_
↪dict.npy')
psr_list = psr_list[-1]['psr_names']
```

```
def get_psrname(file,name_sep='_'):
    return file.split('/')[-1].split(name_sep)[0]
```

```
pars = [f for f in pars if get_psrname(f) in psr_list]
tims = [f for f in tims if get_psrname(f) in psr_list]
noise_files = [f for f in noise_files if get_psrname(f) in psr_list]
len(pars), len(tims), len(noise_files)
```

```
(34, 34, 34)
```

Here we collate the noise parameters into one large dictionary.

```
noise = {}

for nf in noise_files:
    with open(nf,'r') as fin:
        noise.update(json.load(fin))
```

The following loop loads the pulsars into `enterprise.pulsar.Pulsar` class instances. This uses a pulsar timing package in the background, either `Pint` or `TEMPO2` (via the Python wrapper `libstempo`).

Note that warnings about pulsar distances are usual and do not affect this analysis.

```
ePsrs = []
for par,tim in zip(pars,tims):
    ePsr = ePulsar(par, tim,  ephem='DE436')
    ePsrs.append(ePsr)
    print('\rPSR {0} complete'.format(ePsr.name),end='',flush=True)
```

```
PSR B1953+29 completeWARNING: Could not find pulsar distance for PSR J0023+0923. Setting␣
↪value to 1 with 20% uncertainty.
PSR J0030+0451 completeWARNING: Could not find pulsar distance for PSR J0340+4130.␣
↪Setting value to 1 with 20% uncertainty.
PSR J0613-0200 completeWARNING: Could not find pulsar distance for PSR J0645+5158.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1600-3053 completeWARNING: Could not find pulsar distance for PSR J1614-2230.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1713+0747 completeWARNING: Could not find pulsar distance for PSR J1738+0333.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1738+0333 completeWARNING: Could not find pulsar distance for PSR J1741+1351.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1744-1134 completeWARNING: Could not find pulsar distance for PSR J1747-4036.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1747-4036 completeWARNING: Could not find pulsar distance for PSR J1853+1303.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1853+1303 completeWARNING: Could not find pulsar distance for PSR J1903+0327.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1918-0642 completeWARNING: Could not find pulsar distance for PSR J1923+2515.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1923+2515 completeWARNING: Could not find pulsar distance for PSR J1944+0907.␣
↪Setting value to 1 with 20% uncertainty.
PSR J1944+0907 completeWARNING: Could not find pulsar distance for PSR J2010-1323.␣
↪Setting value to 1 with 20% uncertainty.
PSR J2010-1323 completeWARNING: Could not find pulsar distance for PSR J2017+0603.␣
↪Setting value to 1 with 20% uncertainty.
PSR J2017+0603 completeWARNING: Could not find pulsar distance for PSR J2043+1711.␣
↪Setting value to 1 with 20% uncertainty.
PSR J2145-0750 completeWARNING: Could not find pulsar distance for PSR J2214+3000.␣
↪Setting value to 1 with 20% uncertainty.
PSR J2214+3000 completeWARNING: Could not find pulsar distance for PSR J2302+4442.␣
↪Setting value to 1 with 20% uncertainty.
PSR J2317+1439 complete
```

Constructing the Correlation Matrix

The following function makes a correlation matrix using the NANOGrav noise model and the parameters furnished in the data analysis release. For a detailed treatment of the noise modeling see Lam, et al., 2015.

```
def make_corr(psr):
    N = psr.toaerrs.size
    corr = np.zeros((N,N))
    _, _, fl, _, bi = hsen.quantize_fast(psr.toas,psr.toaerrs,
                                         flags=psr.flags['f'],dt=1)
    keys = [ky for ky in noise.keys() if psr.name in ky]
    backends = np.unique(psr.flags['f'])
    sigma_sqr = np.zeros(N)
    ecorrs = np.zeros_like(fl,dtype=float)
    for be in backends:
        mask = np.where(psr.flags['f']==be)
        key_ef = '{0}_{1}_{2}'.format(psr.name,be,'efac')
```

(continues on next page)

(continued from previous page)

```
        key_eq = '{0}_{1}_log10_{2}'.format(psr.name,be,'equad')
        sigma_sqr[mask] = (noise[key_ef]**2 * (psr.toaerrs[mask]**2)
                           + (10**noise[key_eq])**2)
        mask_ec = np.where(fl==be)
        key_ec = '{0}_{1}_log10_{2}'.format(psr.name,be,'ecorr')
        ecorrs[mask_ec] = np.ones_like(mask_ec) * (10**noise[key_ec])
    j = [ecorrs[ii]**2*np.ones((len(bucket),len(bucket)))
         for ii, bucket in enumerate(bi)]

    J = sl.block_diag(*j)
    corr = np.diag(sigma_sqr) + J
    return corr
```

Below we enter the red noise values from the NANOGrav 11-year data set release paper. These were the only pulsars in that paper that were deemed significant in that analysis.

```
rn_psrs = {'B1855+09':[10**-13.7707, 3.6081],
           'B1937+21':[10**-13.2393, 2.46521],
           'J0030+0451':[10**-14.0649, 4.15366],
           'J0613-0200':[10**-13.1403, 1.24571],
           'J1012+5307':[10**-12.6833, 0.975424],
           'J1643-1224':[10**-12.245, 1.32361],
           'J1713+0747':[10**-14.3746, 3.06793],
           'J1747-4036':[10**-12.2165, 1.40842],
           'J1903+0327':[10**-12.2461, 2.16108],
           'J1909-3744':[10**-13.9429, 2.38219],
           'J2145-0750':[10**-12.6893, 1.32307],
           }
```

The following function retrieves the time span across the full set of pulsars.

```
Tspan = hsen.get_Tspan(ePsrs)
```

Set the frequency array across which to calculate the red noise and sensitivity curves.

```
fyr = 1/(365.25*24*3600)
freqs = np.logspace(np.log10(1/(5*Tspan)),np.log10(2e-7),600)
```

Constructing the Array

Here we instantiate `hasasia.Pulsar` class instances using those from `enterprise`. The `make_corr` function constructs a noise correlation matrix based on the noise model used by the NANOGrav collaboration.

Note that the TOAs (and hence the TOA erros and design matrix) are thinnned by a factor of ten. NANOGrav keeps many TOAs from a given observation (often >50), which are not necessary to characterize the sensitivity of the PTA. The differences in these TOAs would only be needed to characterize frequencies much higher than investigated here. Here we thin the TOAs because there are upwards of ~50 TOAs per observing epoch in NANOGrav `tim` files and we don't need all of these to characterize the sensitivity we are interested in. If one has the memory capabilities then the more data the better.

```
psrs = []
thin = 10
```

(continues on next page)

(continued from previous page)

```
for ePsr in ePsrs:
    corr = make_corr(ePsr)[::thin,::thin]
    plaw = hsen.red_noise_powerlaw(A=9e-16, gamma=13/3., freqs=freqs)
    if ePsr.name in rn_psrs.keys():
        Amp, gam = rn_psrs[ePsr.name]
        plaw += hsen.red_noise_powerlaw(A=Amp, gamma=gam, freqs=freqs)

    corr += hsen.corr_from_psd(freqs=freqs, psd=plaw,
                               toas=ePsr.toas[::thin])
    psr = hsen.Pulsar(toas=ePsr.toas[::thin],
                      toaerrs=ePsr.toaerrs[::thin],
                      phi=ePsr.phi,theta=ePsr.theta,
                      N=corr, designmatrix=ePsr.Mmat[::thin,:])
    psr.name = ePsr.name
    psrs.append(psr)
    del ePsr
    print('\rPSR {0} complete'.format(psr.name),end='',flush=True)
```

```
PSR J2317+1439 complete
```

The next step instantiates a `hasasia.Spectrum` class instance for each pulsar. We also calculate the inverse-noie-weighted transmission function, though this is not necessary.

```
specs = []
for p in psrs:
    sp = hsen.Spectrum(p, freqs=freqs)
    _ = sp.NcalInv
    specs.append(sp)
    print('\rPSR {0} complete'.format(p.name),end='',flush=True)
```

```
PSR J2317+1439 complete
```

Individual Pulsar Sensitivity Curves

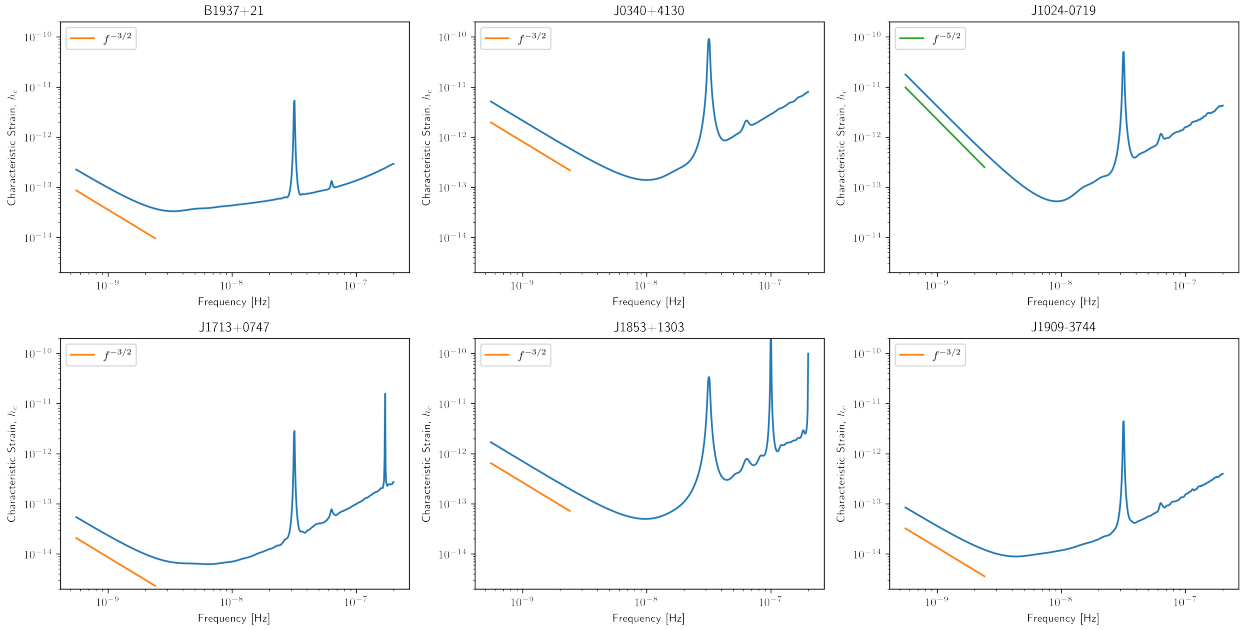
Here we plot a sample of individual pulsar sensitivity curves.

```
fig=plt.figure(figsize=[15,45])
j = 1
names = ['B1937+21','J0340+4130','J1024-0719',
         'J1713+0747','J1853+1303','J1909-3744',]
for sp,p in zip(specs,psrs):
    if p.name in names:
        fig.add_subplot(12,3,j)
        a = sp.h_c[0]/2*1e-14
        if p.name == 'J1024-0719':
            alp = -5/2
            a *= 8e-10
            plt.loglog(sp.freqs[:150],a*(sp.freqs[:150])**(alp),
                       color='C2',label=r'$f^{-5/2}$')
        else:
```

(continues on next page)

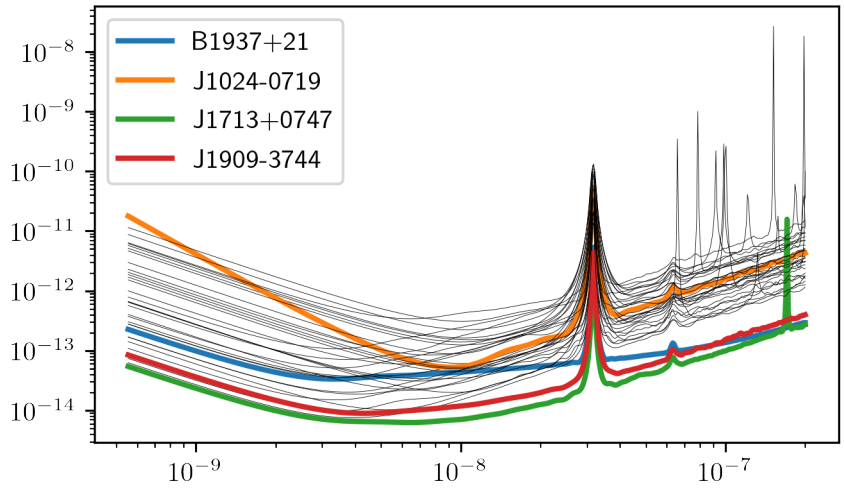
(continued from previous page)

```
            alp = -3/2
            plt.loglog(sp.freqs[:150],a*(sp.freqs[:150])**(alp),
                       color='C1',label=r'$f^{-3/2}$')
        plt.ylim(2e-15,2e-10)
        plt.loglog(sp.freqs,sp.h_c, color='C0')
        plt.rc('text', usetex=True)
        plt.xlabel('Frequency [Hz]')
        plt.ylabel('Characteristic Strain, $h_c$')
        plt.legend(loc='upper left')
        plt.title(p.name)
        j+=1
fig.tight_layout()
plt.show()
plt.close()
```



Below sensitivity curves of the full PTA are plotted, with a few pulsars highlighted.

```
names = ['J1713+0747','B1937+21','J1909-3744','J1024-0719']
for sp,p in zip(specs,psrs):
    if p.name in names:
        plt.loglog(sp.freqs,sp.h_c,lw=2,label=p.name)
    else:
        plt.loglog(sp.freqs,sp.h_c, color='k',lw=0.2)

plt.legend()
plt.show()
plt.close()
```

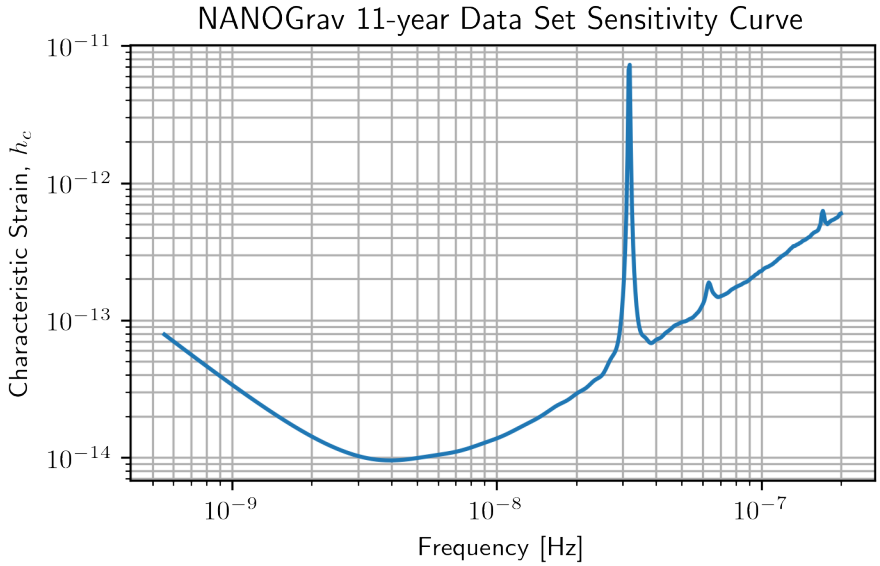


PTA Sensitivity Curves

Full PTA sensitivity curves are constructed by passing a list of `Spectrum` instances to either the `GWBSensitivity` class or `DeterSensitivity` class. See the Sensitivity Curve class for more details.

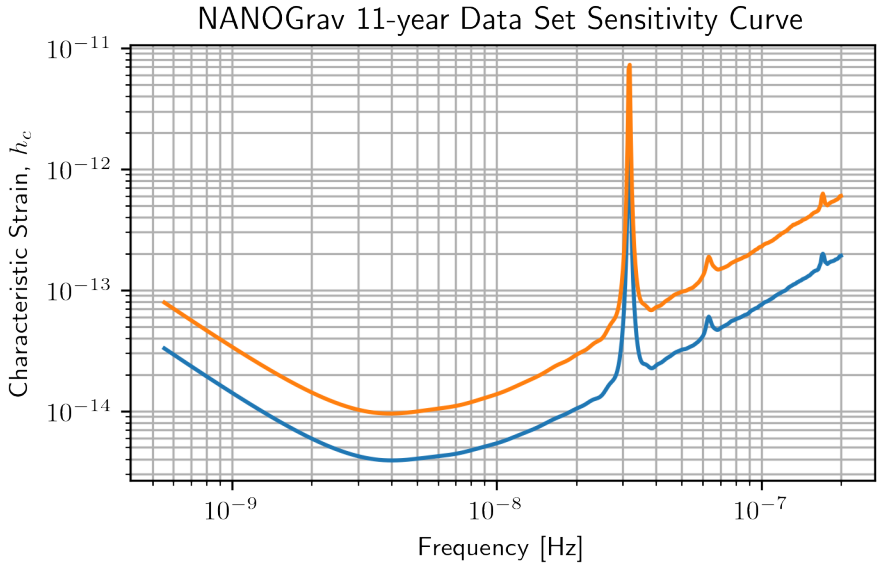
```
ng11yr_sc = hsen.GWBSensitivityCurve(specs)
```

```
plt.loglog(ng11yr_sc.freqs,ng11yr_sc.h_c)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain, $h_c$')
plt.title('NANOGrav 11-year Data Set Sensitivity Curve')
plt.grid(which='both')
# plt.ylim(1e-15,9e-12)
plt.show()
```



```
ng11yr_dsc = hsen.DeterSensitivityCurve(specs)
```

```
plt.loglog(ng11yr_dsc.freqs,ng11yr_dsc.h_c,label='Deterministic')
plt.loglog(ng11yr_sc.freqs,ng11yr_sc.h_c,label='Stochastic')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain, $h_c$')
plt.title('NANOGrav 11-year Data Set Sensitivity Curve')
plt.grid(which='both')
# plt.ylim(1e-15,9e-12)
plt.show()
```

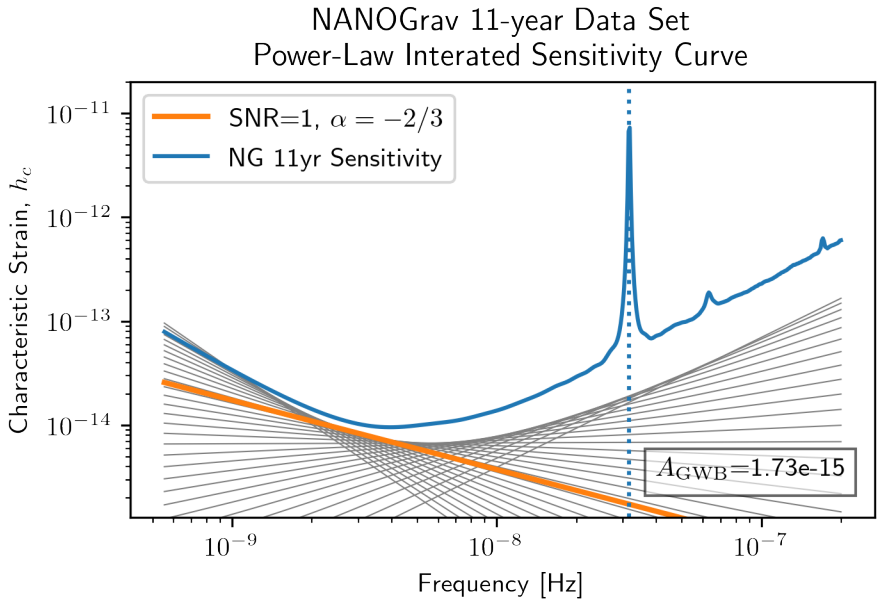


Power-Law Integrated Sensitivity Curves

The `hasasia.sensitivity` module also contains functionality for calculating power-law integrated sensitivity curves. These can be used to calculate the sensitivity to a power-law GWB with a specific spectral or index, or an array of them.

```
#First for alpha=-2/3 (the default value).
SNR=1
hgw=hsen.Agwb_from_Seff_plaw(ng11yr_sc.freqs,
                             Tspan=Tspan,
                             SNR=SNR,
                             S_eff=ng11yr_sc.S_eff)
plaw_h = hgw*(ng11yr_sc.freqs/fyr)**(-2/3)

#And for an array of alpha values.
alpha = np.linspace(-7/4,5/4,30)
h=hsen.Agwb_from_Seff_plaw(freqs=ng11yr_sc.freqs,Tspan=Tspan,SNR=SNR,
                           S_eff=ng11yr_sc.S_eff,alpha=alpha)

plaw = np.dot((ng11yr_sc.freqs[:,np.newaxis]/fyr)**alpha,
              h[:,np.newaxis]*np.eye(30))
```

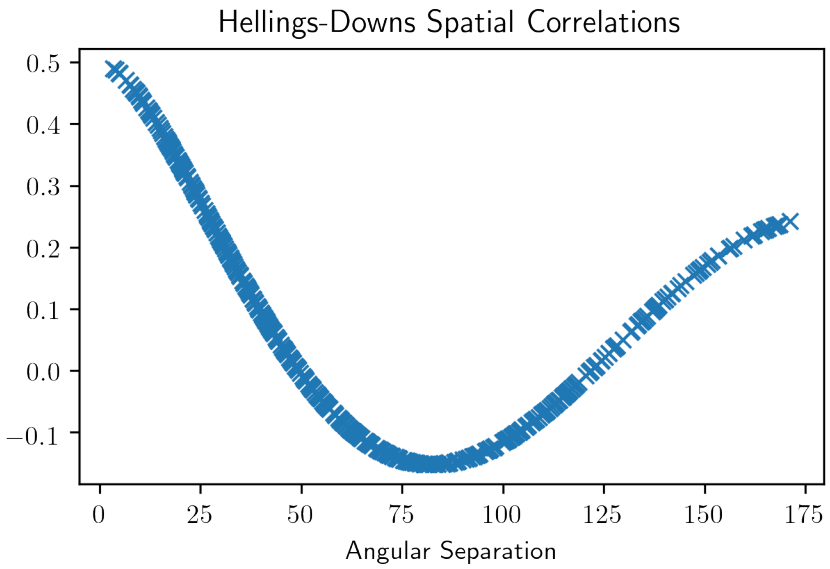
```
for ii in range(len(h)):
    plt.loglog(ng11yr_sc.freqs,plaw[:,ii],
               color='gray',lw=0.5)
plt.loglog(ng11yr_sc.freqs,plaw_h,color='C1',lw=2,
           label='SNR={0}, '.format(SNR)+r'$\alpha=-2/3$')
plt.loglog(ng11yr_sc.freqs,ng11yr_sc.h_c, label='NG 11yr Sensitivity')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain, $h_c$')
plt.axvline(fyr,linestyle=':')
plt.rc('text', usetex=True)
plt.title('NANOGrav 11-year Data Set\nPower-Law Interated Sensitivity Curve')
plt.ylim(hgw*0.75,2e-11)
plt.text(x=4e-8,y=3e-15,
         s=r'$A_{\rm GWB}$='+'{0:1.2e}'.format(hgw),
         bbox=dict(facecolor='white', alpha=0.6))
plt.legend(loc='upper left')
plt.show()
```



Hellings-Downs Curve

The sensitivity curve classes have all of the information needed to make a Hellings-Downs curve for the pulsar pairs in the PTA.

```
ThetaIJ,chiIJ,_,_=hsen.HellingsDownsCoeff(ng11yr_sc.phis,ng11yr_sc.thetas)
plt.plot(np.rad2deg(ThetaIJ),chiIJ,'x')
plt.title('Hellings-Downs Spatial Correlations')
plt.xlabel('Angular Separation')
plt.show()
```



Pairwise Sensitivity Curves

The use can also access the pairwise sensitivity curves through the full PTA `GWBSensitivityCurve`.

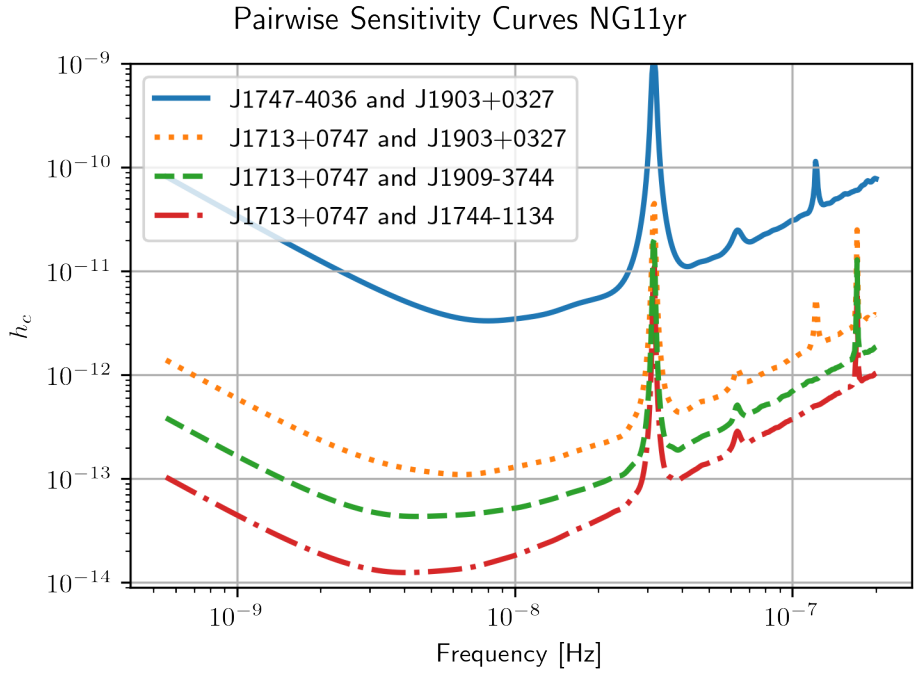
```
psr_names = [p.name for p in psrs]
```

```
fig=plt.figure(figsize=[5,3.5])
j = 0
col = ['C0','C1','C2','C3']
linestyle = ['-',':','--','-.']
for nn,(ii,jj) in enumerate(zip(ng11yr_sc.pairs[0],ng11yr_sc.pairs[1])):
    pair = psr_names[ii], psr_names[jj]
    if ('J1747-4036' in pair and 'J1903+0327' in pair):
        lbl = '{0} and {1}'.format(psr_names[ii],psr_names[jj])
        plt.loglog(ng11yr_sc.freqs,
                   np.sqrt(ng11yr_sc.S_effIJ[nn]*ng11yr_sc.freqs),
                   label=lbl,lw=2, color=col[j],linestyle=linestyle[j],
                   zorder=1)
        j+=1

for nn,(ii,jj) in enumerate(zip(ng11yr_sc.pairs[0],ng11yr_sc.pairs[1])):
    pair = psr_names[ii], psr_names[jj]
    if ('J1713+0747' in pair and 'J1903+0327' in pair):
        lbl = '{0} and {1}'.format(psr_names[ii],psr_names[jj])
        plt.loglog(ng11yr_sc.freqs,
                   np.sqrt(ng11yr_sc.S_effIJ[nn]*ng11yr_sc.freqs),
                   label=lbl,lw=2, color=col[j],linestyle=linestyle[j],
                   zorder=2)
        j+=1

for nn,(ii,jj) in enumerate(zip(ng11yr_sc.pairs[0],ng11yr_sc.pairs[1])):
    pair = psr_names[ii], psr_names[jj]
```

(continues on next page)

(continued from previous page)

```
    if ('J1713+0747' in pair and 'J1909-3744' in pair):
        lbl = '{0} and {1}'.format(psr_names[ii],psr_names[jj])
        plt.loglog(ng11yr_sc.freqs,
                   np.sqrt(ng11yr_sc.S_effIJ[nn]*ng11yr_sc.freqs),
                   label=lbl,lw=2, color=col[j],linestyle=linestyle[j],
                   zorder=4)
        j+=1

for nn,(ii,jj) in enumerate(zip(ng11yr_sc.pairs[0],ng11yr_sc.pairs[1])):
    pair = psr_names[ii], psr_names[jj]
    if ('J1713+0747' in pair and 'J1744-1134' in pair):
        lbl = '{0} and {1}'.format(psr_names[ii],psr_names[jj])
        plt.loglog(ng11yr_sc.freqs,
                   np.sqrt(ng11yr_sc.S_effIJ[nn]*ng11yr_sc.freqs),
                   label=lbl,lw=2, color=col[j],linestyle=linestyle[j],
                   zorder=3)
        j+=1

plt.rc('text', usetex=True)
plt.xlabel('Frequency [Hz]')
plt.ylabel('$h_c$')
plt.ylim(9e-15,1e-9)
plt.legend(loc='upper left')
plt.grid()
# plt.rcParams.update({'font.size':11})
fig.suptitle('Pairwise Sensitivity Curves NG11yr',y=1.03)
fig.tight_layout()
plt.show()
plt.close()
```



SkySensitivity with Real Data

Here we recap the SkySensitivity tutorial using the real NANOGrav data. See the `SkySenstivity` tutorial for more details.

```
#healpy imports
import healpy as hp
import astropy.units as u
import astropy.constants as c
```

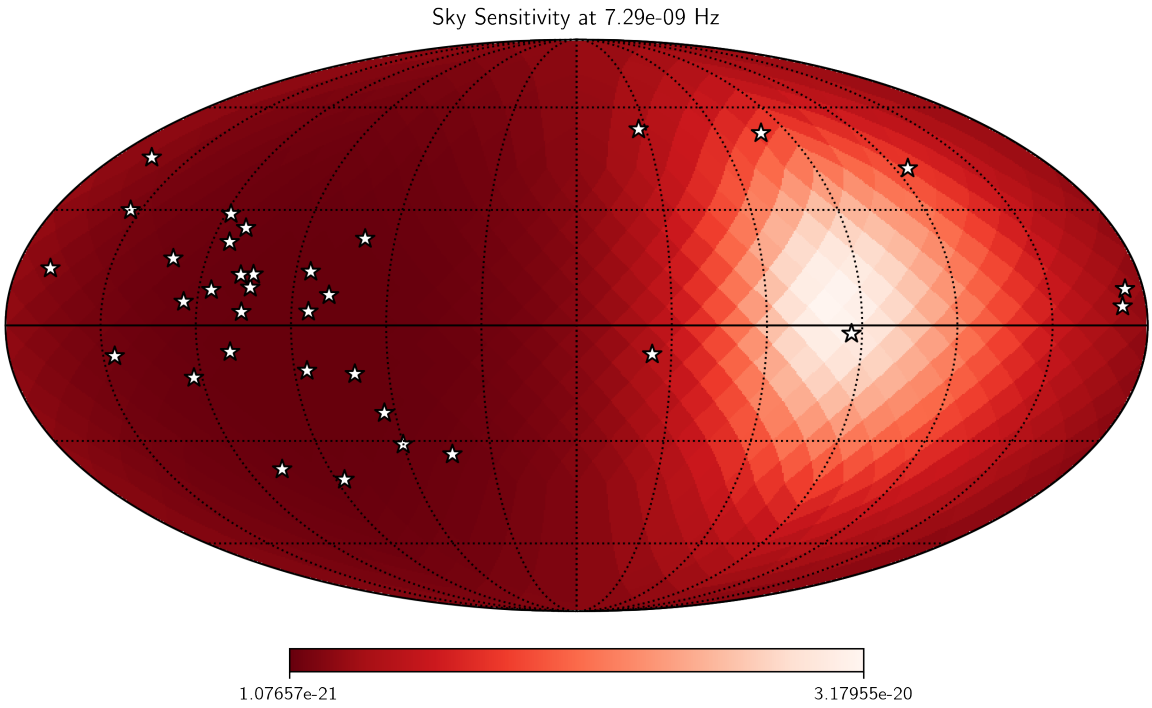
```
NSIDE = 8
NPIX = hp.nside2npix(NSIDE)
IPIX = np.arange(NPIX)
theta_gw, phi_gw = hp.pix2ang(nside=NSIDE,ipix=IPIX)
```

```
SM = hsky.SkySensitivity(specs,theta_gw, phi_gw)
```

```
min_idx = np.argmin(ng11yr_sc.S_eff)
```

```
idx = min_idx
hp.mollview(SM.S_effSky[idx],
            title="Sky Sensitivity at {0:2.2e} Hz".format(SM.freqs[idx]),
            cmap='Reds_r',rot=(180,0,0))
hp.visufunc.projscatter(SM.thetas,SM.phis,marker='*',
                        color='white',edgecolors='k',s=100)
hp.graticule()
plt.show()
```

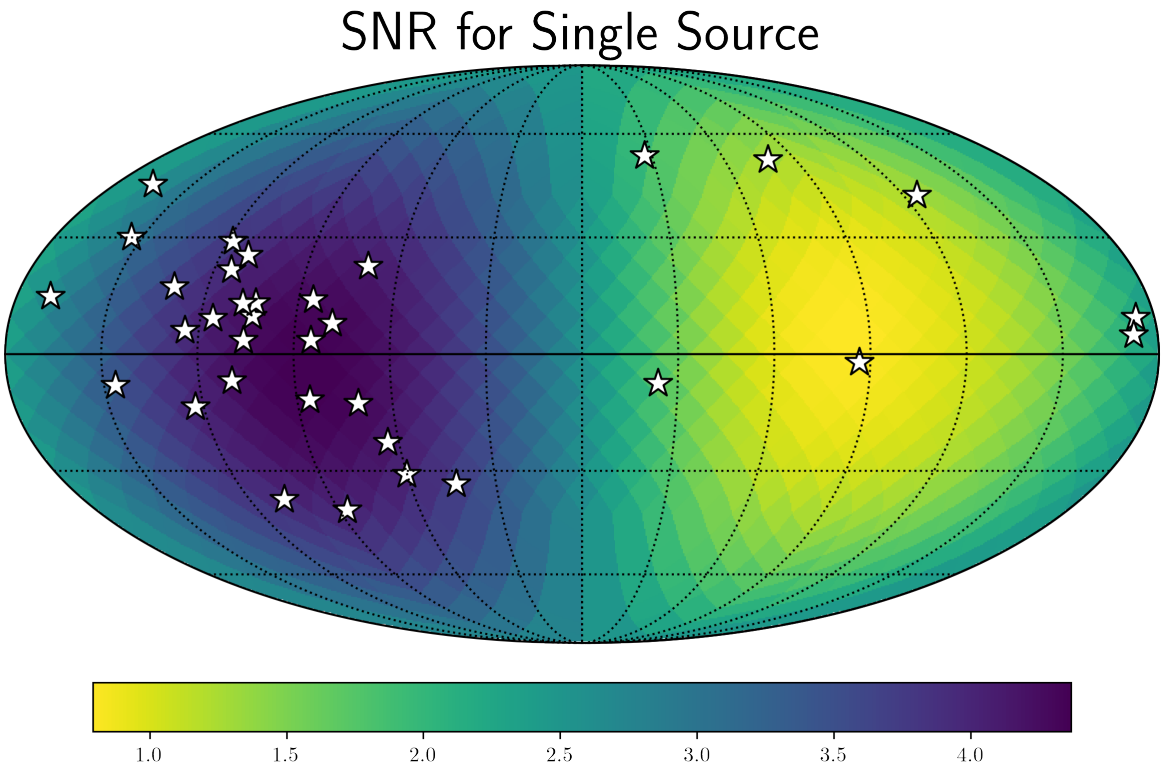
```
0.0 180.0 -180.0 180.0
The interval between parallels is 30 deg -0.00'.
The interval between meridians is 30 deg -0.00'.
```




```
f0=8e-9
hcw = hsky.h_circ(1e9,120,f0,Tspan,SM.freqs).to('').value
SkySNR = SM.SNR(hcw)
```

```
plt.rc('text', usetex=True)
hp.mollview(SkySNR,rot=(180,0,0),#np.log10(1/SM.Sn[idx]),"SNR with Single Source"
            cmap='viridis_r',cbar=None,title='')
hp.visufunc.projscatter(SM.thetas,SM.phis,marker='*',
                        color='white',edgecolors='k',s=200)
hp.graticule()
fig = plt.gcf()
ax = plt.gca()
image = ax.get_images()[0]
cmap = fig.colorbar(image, ax=ax,orientation='horizontal',shrink=0.8,pad=0.05)

plt.rcParams.update({'font.size':22,'text.usetex':True})
ax.set_title("SNR for Single Source")
plt.show()
```

```
0.0 180.0 -180.0 180.0
The interval between parallels is 30 deg -0.00'.
The interval between meridians is 30 deg -0.00'.
```




```
import matplotlib.ticker as ticker
```

```
hdivA= hcw / hsky.h0_circ(1e9,120,f0)
Agw = SM.A_gwb(hdivA).to('').value
```

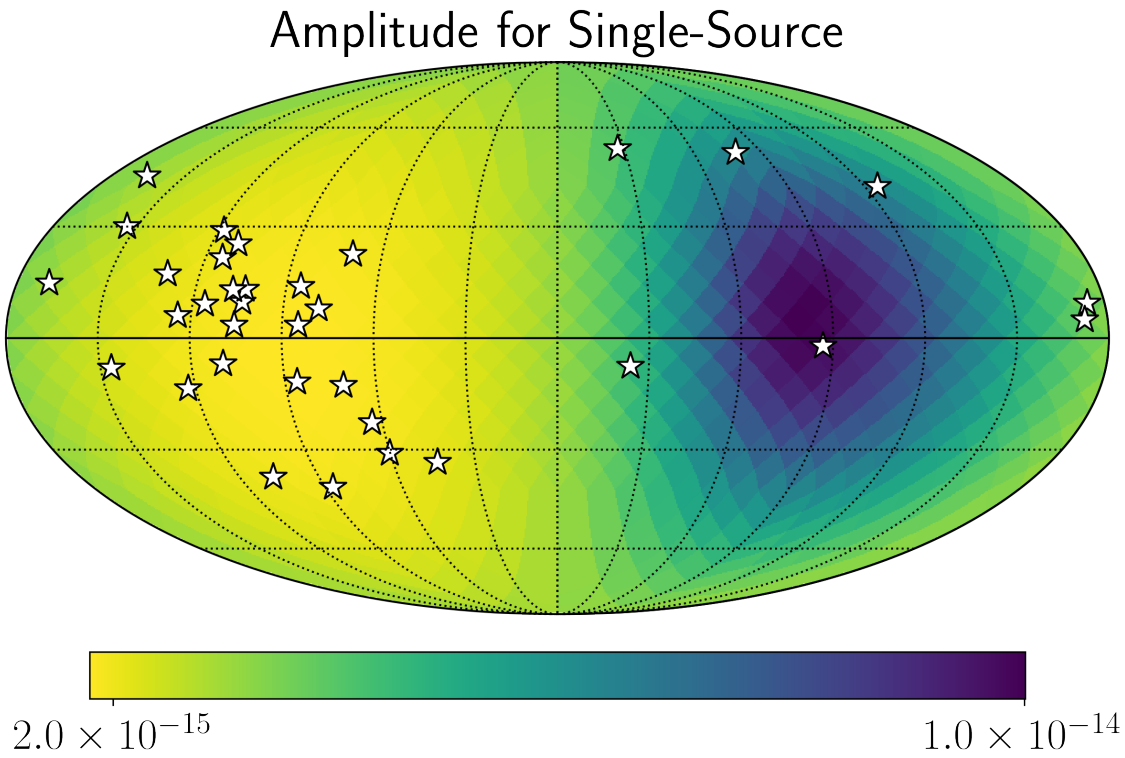
```
idx = min_idx
hp.mollview(Agw,rot=(180,0,0),
            title="",cbar=None,
            cmap='viridis_r')
hp.visufunc.projscatter(SM.thetas,SM.phis,marker='*',
                        color='white',edgecolors='k',s=200)
hp.graticule()
#
fig = plt.gcf()
ax = plt.gca()
image = ax.get_images()[0]
cbar_ticks = [2.02e-15,1e-14]

plt.rcParams.update({'font.size':22,'text.usetex':True})
def fmt(x, pos):
    a, b = '{:.1e}'.format(x).split('e')
    b = int(b)
    return r'${} \times 10^{{{}}}$'.format(a, b)
ax.set_title("Amplitude for Single-Source")
cmap = fig.colorbar(image, ax=ax,orientation='horizontal',
                    ticks=cbar_ticks,shrink=0.8,
                    format=ticker.FuncFormatter(fmt),pad=0.05)
```

(continues on next page)

(continued from previous page)

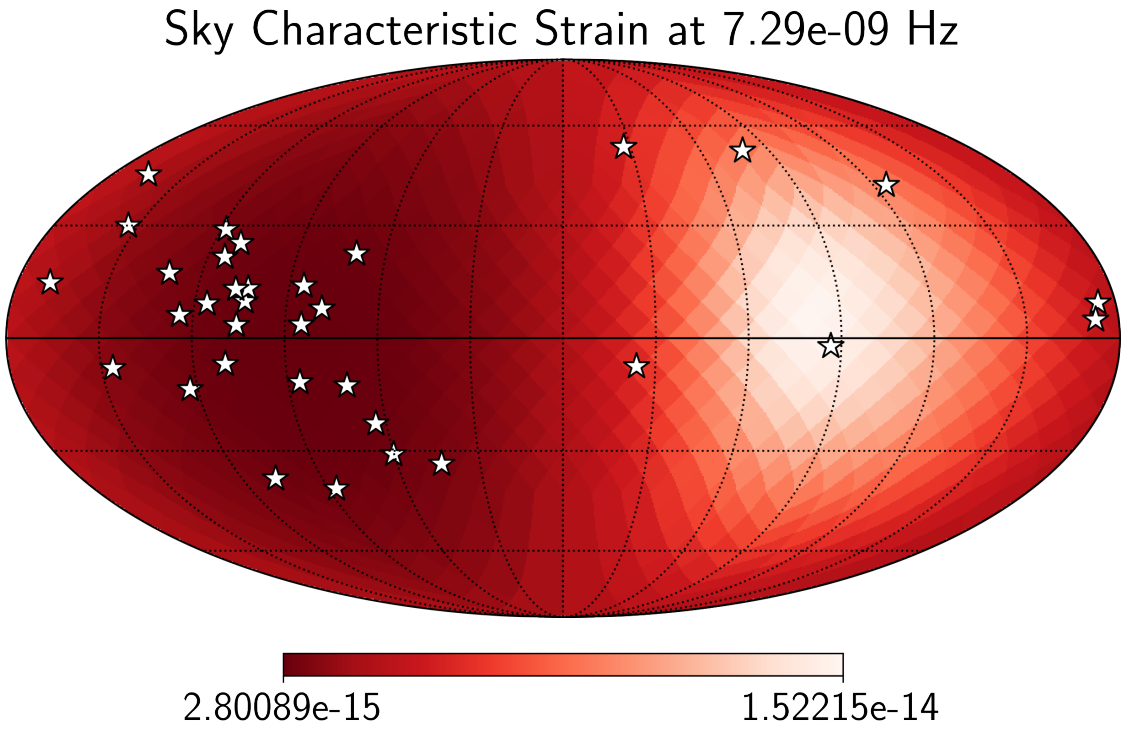
```
plt.show()
```

```
0.0 180.0 -180.0 180.0
The interval between parallels is 30 deg -0.00'.
The interval between meridians is 30 deg -0.00'.
```




```
idx = min_idx
hp.mollview(SM.h_c[idx],
            title="Sky Characteristic Strain at {0:2.2e} Hz".format(SM.freqs[idx]),
            cmap='Reds_r',rot=(180,0,0))
hp.visufunc.projscatter(SM.thetas,SM.phis,marker='*',
                        color='white',edgecolors='k',s=200)
hp.graticule()
plt.show()
```

```
0.0 180.0 -180.0 180.0
The interval between parallels is 30 deg -0.00'.
The interval between meridians is 30 deg -0.00'.
```



1.1.7 Detailed User Interface Information

Sensitivity

The main module in *hasasia* is the Sensitivity module which contains all of the needed methods for building a senstivity curve.

hasasia.sensitivity.**Agwb_from_Seff_plaw**(*freqs*, *Tspan*, *SNR*, *S_eff*, *gamma=4.333333333333333*, *alpha=None*)

Must supply numpy.ndarrays.

class hasasia.sensitivity.**DeterSensitivityCurve**(*spectra*, *pulsar_term=True*, *include_corr=False*, *A_GWB=None*)

Parameters

include_corr
[bool] Whether to include cross correlations from the GWB as an additional noise source in full PTA correlation matrix. (Has little to no effect and adds a lot of computation time.)

A_GWB
[float] Value of GWB amplitude for use in cross correlations.

property H_0

Hubble Constant. Assumed to be in units of km /(s Mpc) unless supplied as an *astropy.quantity*.

property NcalInvIJ

Inverse Noise Weighted Transmission Function that includes cross-correlation noise from GWB.

property Omega_gw

Energy Density sensitivity

SNR(*h0*)

Calculate the signal-to-noise ratio of a source given the strain amplitude. This is based on Equation (79) from Hazboun, et al., 2019 [1].

$$\rho(\hat{n}) = h_0 \sqrt{\frac{T_{\rm obs}}{S_{\rm eff}(f_0, \hat{k})}}$$

property S_eff

Strain power sensitivity.

property h_c

Characteristic strain sensitivity

class hasasia.sensitivity.**GWBSensitivityCurve**(*spectra*, *orf='hd'*, *autocorr=False*)

Class to produce a sensitivity curve for a gravitational wave background, using Hellings-Downs spatial correlations.

Parameters

orf
[str, optional {'hd', 'st', 'dipole', 'monopole'}] Overlap reduction function to be used in the sensitivity curve. Maybe be Hellings-Downs, Scalar-Tensor, Dipole or Monopole.

property H_0

Hubble Constant. Assumed to be in units of km /(s Mpc) unless supplied as an *astropy.quantity*.

property Omega_gw

Energy Density sensitivity

SNR(*Sh*)

Calculate the signal-to-noise ratio of a given signal strain power spectral density, *Sh*. Must match frequency range and *df* of *self*.

property S_eff

Strain power sensitivity.

property S_effIJ

Strain power sensitivity.

property h_c

Characteristic strain sensitivity

hasasia.sensitivity.**G_matrix**(*designmatrix*)

Create G matrix as defined in van Haasteren 2013

Parameters

designmatrix
[array] Design matrix for a pulsar timing model.

Returns

G matrix

hasasia.sensitivity.**HellingsDownsCoeff**(*phi*, *theta*, *autocorr=False*)

Calculate Hellings and Downs coefficients from two lists of sky positions.

Parameters

phi
[array, list] Pulsar axial coordinate.

theta
[array, list] Pulsar azimuthal coordinate.

Returns

ThetaIJ
[array] An Npair-long array of angles between pairs of pulsars.

chiIJ
[array] An Npair-long array of Hellings and Downs relation coefficients.

pairs
[array] A 2xNpair array of pair indices corresponding to input order of sky coordinates.

chiRSS
[float] Root-sum-squared value of all Hellings-Downs coefficients.

hasasia.sensitivity.**PI_hc**(*freqs*, *Tspan*, *SNR*, *S_eff*, *N=200*)

Power law-integrated characteristic strain.

class hasasia.sensitivity.**Pulsar**(*toas*, *toaerrs*, *phi=None*, *theta=None*, *designmatrix=None*, *N=None*, *pdist=<Quantity 1. kpc>*)

Class to encode information about individual pulsars.

Parameters

toas
[array] Pulsar Times of Arrival [sec].

toaerrs
[array] Pulsar TOA errors [sec].

phi
[float] Ecliptic longitude of pulsar [rad].

theta
[float] Ecliptic latitude of pulsar [rad].

designmatrix
[array] Design matrix for pulsar's timing model. N_TOA x N_param.

N
[array] Covariance matrix for the pulsar. N_TOA x N_TOA. Made from toaerrs if not provided.

pdist
[astropy.quantity, float] Earth-pulsar distance. Default units is kpc.

property G

Inverse Noise Weighted Transmission Function.

hasasia.sensitivity.**R_matrix**(*designmatrix*, *N*)

Create R matrix as defined in Ellis et al (2013) and Demorest et al (2012)

Parameters

designmatrix
[array] Design matrix of timing model.

N
[array] TOA uncertainties [s]

Returns

R matrix

class hasasia.sensitivity.**Spectrum**(*psr*, *nf=400*, *fmin=None*, *fmax=2e-07*, *freqs=None*, *tm_fit=True*, ***Tf_kwargs*)

Class to encode the spectral information for a single pulsar.

Parameters

psr
[*hasasia.Pulsar*] A *hasasia.Pulsar* instance.

nf
[int, optional] Number of frequencies over which to build the various spectral densities.

fmin
[float, optional [Hz]] Minimum frequency over which to build the various spectral densities. Defaults to the timespan/5 of the pulsar.

fmax
[float, optional [Hz]] Minimum frequency over which to build the various spectral densities.

freqs
[array, optional [Hz]] Optionally supply an array of frequencies over which to build the various spectral densities.

property **NcalInv**

Inverse Noise Weighted Transmission Function.

property **Omega_gw**

Energy Density sensitivity.

$$\Omega_{gw} = \frac{2\pi^2}{3\ H_0^2} f^3\ S_I$$

property **P_n**

Inverse Noise Weighted Transmission Function.

property **S_I**

Strain power sensitivity for this pulsar. Equation (74) in [1]

$$S_I = \frac{1}{\mathcal{N}^{-1}\ \mathcal{R}}$$

property **S_R**

Residual power sensitivity for this pulsar.

$$S_R = \frac{1}{\mathcal{N}^{-1}}$$

add_noise_power(*noise*)

Add any spectrum of noise. Must match length of frequency array.

Note: All noise information is furnished by the covariance matrix in the *hasasia.Pulsar* object, this is simply useful for bookkeeping and plots.

add_red_noise_power(*A=None*, *gamma=None*, *vals=False*)

Add power law red noise to the prefit residual power spectral density. As $P = A^2(f/fyr)^{-\gamma}$.

Note: All noise information is furnished by the covariance matrix in the *hasasia.Pulsar* object, this is simply useful for bookkeeping and plots.

Parameters

A
[float] Amplitude of red noise.

gamma
[float] Spectral index of red noise powerlaw.

vals
[bool] Whether to return the psd values as an array. Otherwise just added to *self.psd_prefit*.

add_white_noise_power(*sigma=None*, *dt=None*, *vals=False*)

Add power law red noise to the prefit residual power spectral density.

Note: All noise information is furnished by the covariance matrix in the *hasasia.Pulsar* object, this is simply useful for bookkeeping and plots.

Parameters

sigma
[float] TOA error.

dt
[float] Time between observing epochs in [seconds].

vals
[bool] Whether to return the psd values as an array. Otherwise just added to *self.psd_prefit*.

property h_c

Characteristic strain sensitivity for this pulsar.

$$h_c = \sqrt{f\ S_I}$$

property psd_postfit

Postfit Residual Power Spectral Density

property psd_prefit

Prefit Residual Power Spectral Density

hasasia.sensitivity.**corr_from_psd**(*freqs*, *psd*, *toas*, *fast=True*)

Calculates the correlation matrix over a set of TOAs for a given power spectral density.

Parameters

freqs
[array] Array of freqs over which the psd is given.

psd
[array] Power spectral density to use in calculation of correlation matrix.

toas
[array] Pulsar times-of-arrival to use in correlation matrix.

fast
[bool, optional] Fast mode uses a matix inner product, while the slower mode uses the numpy.trapz function which is slower, but more accurate.

Returns

corr
[array] A 2-dimensional array which represents the correlation matrix for the given set of TOAs.

hasasia.sensitivity.**get_NcalInv**(*psr*, *nf=200*, *fmin=None*, *fmax=2e-07*, *freqs=None*, *exact_yr_freqs=False*, *full_matrix=False*, *return_Gtilde_Ncal=False*, *tm_fit=True*, *Gmatrix=None*)

Calculate the inverse-noise-wieghted transmission function for a given pulsar. This calculates $\mathcal{N}^{-1}(f, f')$, $\mathcal{N}^{-1}(f)$ in [1], see Equations (19-20).

Parameters

psr
[array] Pulsar object.

nf
[int, optional] Number of frequencies at which to calculate transmission function.

fmin
[float, optional] Minimum frequency at which to calculate transmission function.

fmax
[float, optional] Maximum frequency at which to calculate transmission function.

exact_yr_freqs
[bool, optional] Whether to use exact 1/year and 2/year frequency values in calculation.

full_matrix
[bool, optional] Whether to return the full, two frequency NcalInv.

return_Gtilde_Ncal
[bool, optional] Whether to return Gtilde and Ncal. Gtilde is the Fourier transform of the G-matrix.

tm_fit
[bool, optional] Whether to include the timing model fit in the calculation.

Gmatrix
[ndarray, optional] Provide already calculated G-matrix. This can speed up calculations since the singular value decomposition can take time for large matrices.

Returns

inverse-noise-weighted transmission function

hasasia.sensitivity.**get_Tf**(*designmatrix*, *toas*, *N=None*, *nf=200*, *fmin=None*, *fmax=2e-07*, *freqs=None*, *exact_astro_freqs=False*, *from_G=True*, *twofreqs=False*, *Gmatrix=None*)

Calculate the transmission function for a given pulsar design matrix, TOAs and TOA errors.

Parameters

designmatrix
[array] Design matrix for a pulsar timing model, N_TOA x N_param.

toas
[array] Times-of-arrival for pulsar, N_TOA long.

N
[array] Covariance matrix for pulsar time-of-arrivals, N_TOA x N_TOA. Often just a diagonal matrix of inverse TOA errors squared.

nf
[int, optional] Number of frequencies at which to calculate transmission function.

fmin
[float, optional] Minimum frequency at which to calculate transmission function.

fmax
[float, optional] Maximum frequency at which to calculate transmission function.

exact_astro_freqs
[bool, optional] Whether to use exact 1/year and 2/year frequency values in calculation.

from_G
[bool, optional] Whether to use G matrix for transmission function calculate. If False R-matrix is used.

twofreqs
[bool, optional] Whether to calculate a two frequency transmission function.

Gmatrix
[ndarray, optional] Provide already calculated G-matrix. This can speed up calculations since the singular value decomposition can take time for large matrices.

`hasasia.sensitivity.get_Tspan(psrs)`

Returns the total timespan from a list or arry of Pulsar objects, psrs.

`hasasia.sensitivity.get_TspanIJ(psr1, psr2)`

Returns the overlapping timespan of two Pulsar objects, psr1/psr2.

`hasasia.sensitivity.nanograv_11yr_deter()`

Returns a *DeterSensitivityCurve* object built using with the NANOGrav 11-year data set.

`hasasia.sensitivity.nanograv_11yr_stoch()`

Returns a *GWBSensitivityCurve* object built using with the NANOGrav 11-year data set.

`hasasia.sensitivity.quantize_fast(toas, toaerrs, flags=None, dt=0.1)`

Function to quantize and average TOAs by observation epoch. Used especially for NANOGrav multiband data.

Pulled from [3].

Parameters

times
[array] TOAs for a pulsar.

flags
[array, optional] Flags for TOAs.

dt
[float] Coarse graining time [days].

`hasasia.sensitivity.red_noise_powerlaw(A, freqs, gamma=None, alpha=None)`

Add power law red noise to the prefit residual power spectral density. As $P = A^2(f/fyr)^{-\gamma}$

Parameters

A
[float] Amplitude of red noise.

gamma
[float] Spectral index of red noise powerlaw.

freqs
[array] Frequencies at which to calculate the red noise power law.

hasasia.sensitivity.**resid_response**(*freqs*)

Returns the timing residual response function for a pulsar across as set of frequencies. See Equation (53) in [1].

$$\mathcal{R}(f) = \frac{1}{12\pi^2\ f^2}$$

Skymap

The *hasasia.Skymap* module contains the methods necessary for making sensitivity maps for pulsar timing array.

class hasasia.skymap.**SkySensitivity**(*spectra*, *theta_gw*, *phi_gw*, *pulsar_term=False*, *pol='gr'*, *iota=None*, *psi=None*)

Class to make sky maps for deterministic PTA gravitational wave signals. Calculated in terms of $\hat{n} = -\hat{k}$.

Parameters

theta_gw
[list, array] Gravitational wave source sky location colatitude at which to calculate sky map.

phi_gw
[list, array] Gravitational wave source sky location longitude at which to calculate sky map.

pulsar_term
[bool, str, optional [True, False, 'explicit']] Flag for including the pulsar term in sky map sensitivity. True includes an idealized factor of two from Equation (36) of [1]. The *'explicit'* flag turns on an explicit calculation of pulsar terms using pulsar distances. (This option takes considerably more computational resources.)

pol: str, optional ['gr','scalar-trans','scalar-long','vector-long']
Polarization of gravitational waves to be used in pulsar antenna patterns. Only one can be used at a time.

A_gwb(*h_div_A*, *SNR=1*)

Method to return a skymap of amplitudes needed to see signal the specified signal, given the specified SNR.

Parameters

h_div_A
[array] An array that represents the frequency dependence of a signal that has been divided by the amplitude. Must cover the same frequencies covered by the *Skymap.freqs*.

SNR
[float, optional] Desired signal-to-noise ratio.

Returns

An array representing the skymap of amplitudes needed to see the given signal.

property H_0

Hubble Constant. Assumed to be in units of km /(s Mpc) unless supplied as an *astropy.quantity*.

property NcalInvIJ

Inverse Noise Weighted Transmission Function that includes cross-correlation noise from GWB.

property Omega_gw

Energy Density sensitivity

SNR(*h0*, *iota=None*, *psi=None*)

Calculate the signal-to-noise ratio of a source given the strain amplitude. This is based on Equation (79) from Hazboun, et al., 2019 [1].

$$\rho(\hat{n}) = h_0 \sqrt{\frac{T_{\rm obs}}{S_{\rm eff}(f_0, \hat{k})}}$$

property S_SkyI

Per Pulsar Strain power sensitivity.

S_SkyI_full(*iota*, *psi*)

Per Pulsar Strain power sensitivity.

property S_eff

Strain power sensitivity.

S_eff_full(*iota*, *psi*)

Strain power sensitivity.

property S_eff_mean

Strain power sensitivity.

property h_c

Characteristic strain sensitivity

h_thresh(*SNR=1*, *iota=None*, *psi=None*)

Method to return a skymap of amplitudes needed to see a circular binary, given the specified SNR. This is based on Equation (80) from Hazboun, et al., 2019 [1].

$$h_0 = \rho(\hat{n}) \sqrt{\frac{S_{\rm eff}(f_0, \hat{k})}{T_{\rm obs}}}$$

Parameters

SNR
[float, optional] Desired signal-to-noise ratio.

Returns

An array representing the skymap of amplitudes needed to see the given signal with the SNR threshold specified.

sky_response_full(*iota*, *psi*)

Calculate the signal-to-noise ratio of a source given the strain amplitude. This is based on Equation (79) from Hazboun, et al., 2019 [1].

$$\rho(\hat{n}) = h_0 \sqrt{\frac{T_{\rm obs}}{S_{\rm eff}(f_0, \hat{k})}}$$

hasasia.skymap.**h_circ**(*M_c*, *D_L*, *f0*, *Tspan*, *f*)

Convenience function that returns the Fourier domain representation of a single circular super-massive binary black hole.

Parameters

M_c
[float [M_sun]] Chirp mass of a SMBHB.

D_L
[float [Mpc]] Luminosity distance to a SMBHB.

f0
[float [Hz]] Frequency of the SMBHB.

Tspan
[float [sec]] Timespan that the binary has been observed. Usually taken as the timespan of the data set.

f
[array [Hz]] Array of frequencies over which to model the Fourier domain signal.

Returns

hcw
[array [strain]] Array of strain values across the frequency range provided for a circular SMBHB.

Utils

hasasia.utils.**create_design_matrix**(*toas*, *RADEC=False*, *PROPER=False*, *PX=False*)

Return designmatrix for quadratic spindown model + optional astrometric parameters

Parameters

toas
[array] TOA measurements [s]

RADEC
[bool, optional] Includes RA/DEC fitting.

PROPER
[bool, optional] Includes proper motion fitting.

PX
[bool, optional] Includes parallax fitting.

Returns

M
[array] Design matrix for quadratic spin down + optional astrometry fit.

hasasia.utils.**fap**(*F*, *Npsrs=None*)

False alarm probability of the F-statistic Use None for the Fe statistic and the number of pulsars for the Fp stat.

Simulations

hasasia.sim.**sim_pta**(*timespan*, *cad*, *sigma*, *phi*, *theta*, *Npsrs=None*, *A_rn=None*, *alpha=None*, *freqs=None*, *uneven=False*, *A_gwb=None*, *fast=True*, *kwastro={'PROPER': True, 'PX': True, 'RADEC': True}*)

Make a simulated pulsar timing array. Using the available parameters, the function returns a list of pulsar objects encoding them.

Parameters

timespan
[float, array, list] Timespan of observations in [years].

cad
[float, array, list] Cadence of observations [number/yr].

sigma
[float, array, list] TOA RMS Error [sec]. Single float, Npsrs long array, or Npsrs x NTOA array excepted.

phi
[array, list] Pulsar's longitude in ecliptic coordinates.

theta
[array, list] Pulsar's colatitude in ecliptic coordinates.

Npsrs
[int, optional] Number of pulsars. Only needed if all pulsars have the same noise characteristics.

A_rn
[float, optional] Red noise amplitude to be injected for each pulsar.

alpha
[float, optional] Red noise spectral index to be injected for each pulsar.

freqs
[array, optional] Array of frequencies at which to calculate the red noise. Same array used for all pulsars.

uneven
[bool, optional] Option to have the toas be unevenly sampled.

fast
[bool, optional] Option to use the faster, less accurate PSD-to-correlation matrix calculation.

Returns

psrs
[list] List of *hasasia.Pulsar()* objects.

1.1.8 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/Hazboun6/hasasia/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

hasasia could always use more documentation, whether as part of the official hasasia docs, in docstrings, or even on the web in blog posts, articles, and such.

Once you change or add documentation within the *docs/* directory you can run *make html* to use *sphinx* to convert the *.rst* files into html. You can then open the documentation by pointing a web browser to *~/hasasia/docs/_build/html/index.html*. The packages needed to compile the documentation can be found in the *requirements_docs.txt* file.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Hazboun6/hasasia/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *hasasia* for local development.

1. Fork the *hasasia* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/hasasia.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv hasasia
$ cd hasasia/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 hasasia tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/Hazboun6/hasasia/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_hasasia
```

Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

1.1.9 Contributor Covenant Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at jeffrey.hazboun@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available here.

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

1.1.10 Credits

Development Lead

- Jeffrey S. Hazboun <jeffrey.hazboun@gmail.com>

Contributors

- Joseph D. Romano
- Tristan L. Smith

1.1.11 History

1.2.3 (2021-09-14) Remove pytest-runner from setup.py

1.2.2 (2021-09-14) Check Build status

1.2.1 (2021-09-14) Fix auto deploy.

1.2.0 (2021-09-14) Added new single source functionality to SkyMap.

1.1.2 (2020-09-12) Added test for the tutorials and a new version of make_corr.

1.1.1 (2020-06-11) Fix pulsar term functionality in SkySensitivity

1.1.0 (2020-06-11) Various extra functionality added, including non-GR polarizations and pulsar term flags.

1.0.5 (2019-10-21) JOSS/Zenodo Release with various changes from JOSS Refereeing and correct Zenodo *.json*

1.0.4 (2019-10-21) JOSS/Zenodo Release with various changes from JOSS Refereeing and correct Zenodo *.json*

1.0.3 (2019-10-21) JOSS/Zenodo Release with various changes from JOSS Refereeing and correct Zenodo *.json*

1.0.2 (2019-10-21) JOSS/Zenodo Release with various changes from JOSS Refereeing and correct Zenodo *.json*

1.0.1 (2019-10-21) JOSS/Zenodo Release with various changes from JOSS Refereeing

1.0.0 (2019-09-20) The Official Release.

0.1.7 (2019-08-30)

0.1.6 (2019-08-29)

0.1.5 (2019-08-13)

0.1.4 (2019-08-13)

0.1.3 (2019-08-13)

0.1.2 (2019-06-23)

0.1.1 (2019-06-23)

0.1.0 (2019-06-23)*

- First release on PyPI.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

P

Q

R

S