gwent Documentation

Release 0.4.0

Andrew Kaiser

Jan 26, 2021

Contents:

1 gwent **1**
- 1.1 Features . . . 2
- 1.2 Getting Started . . . 2
- 1.3 README Figure and Data . . . 2
- 1.4 Publication . . . 3
- 1.5 Credits . . . 3

2 Installation **5**
- 2.1 Stable release . . . 5
- 2.2 From sources . . . 5

3 Usage **7**

4 Using `gwent` to Generate Characteristic Strain Curves **9**
- 4.1 Initialize different instruments . . . 10
- 4.2 Load ground instruments from files . . . 10
- 4.3 Load LISA Instruments from File . . . 12
- 4.4 Loading PTA Detection Curves and Upper Limits . . . 13
- 4.5 Generating PTAs with `gwent` . . . 15
- 4.6 Generating LISA designs with `gwent` . . . 18
- 4.7 Generating Ground Based Detector Designs with `gwent` . . . 20
- 4.8 Generating Binary Black Holes with `gwent` in the Frequency Domain . . . 25
- 4.9 Plots of Entire GW Band . . . 27

5 Using `gwent` to Generate Source Characteristic Strain Curves **31**
- 5.1 Initialize different instruments . . . 32
- 5.2 Generating Binary Black Holes with `gwent` in the Frequency Domain . . . 33
- 5.3 How to Get Information about BHB . . . 33
- 5.4 Plots of Example GW Band . . . 35
- 5.5 Calculating the SNR . . . 38
- 5.6 Generate Frequency Data from Given Time Domain . . . 39

6 Using `gwent` to Calculate Signal-to-Noise Ratios **41**
- 6.1 Fiducial Source Creation . . . 42
- 6.2 Create SNR Matrices and Samples for a Few Examples . . . 43
- 6.3 Instrument Creation Examples . . . 44
- 6.4 SNR Calculations . . . 46

7 gwent **61**
7.1 gwent package . . . 61

8 Contributing **65**
8.1 Types of Contributions . . . 65
8.2 Get Started! . . . 66
8.3 Pull Request Guidelines . . . 67
8.4 Deploying . . . 67

9 Credits **69**
9.1 Development Lead . . . 69
9.2 Contributors . . . 69

10 History **71**
10.1 0.4.0 (2021-1-8) . . . 71
10.2 0.3.0 (2020-10-5) . . . 71
10.3 0.2.1 (2020-10-5) . . . 71
10.4 0.2.0 (2020-4-29) . . . 72
10.5 0.1.16 (2020-1-19) . . . 72
10.6 0.1.15 (2020-1-18) . . . 72
10.7 0.1.14 (2020-1-7) . . . 72
10.8 0.1.13 (2019-10-28) . . . 72
10.9 0.1.12 (2019-10-8) . . . 72
10.10 0.1.11 (2019-09-19) . . . 72
10.11 0.1.10 (2019-09-14) . . . 73
10.12 0.1.0 (2019-09-04) . . . 73

11 Indices and tables **75**

Python Module Index **77**

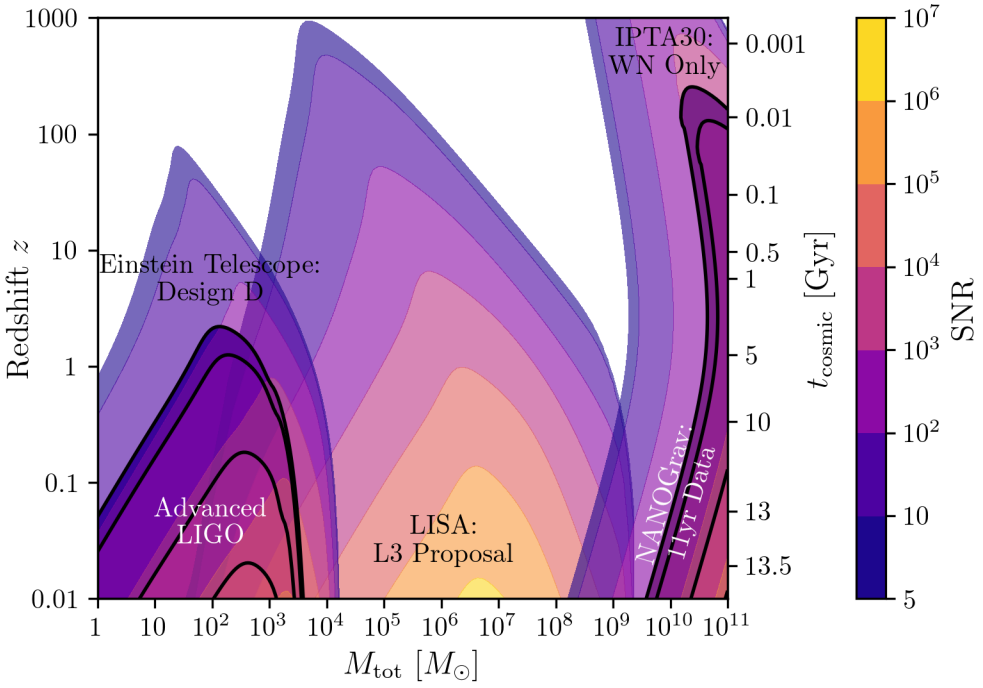
Index **79**

CHAPTER 1

gwent

Gravitational Wave dEtector desigN Toolkit.

Generates strain sensitivity curves and Waterfall plots for various gravitational wave detector designs.



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

1.1 Features

Calculates the sensitivity curves for various designs of pulsar timing arrays, space-based detectors, and ground-based detectors. This includes:

- NANOGrav
- SKA
- LISA
- aLIGO
- Voyager
- and more!

Calculates the strain from coalescing black hole binaries. It contains functionality for different source descriptions:

- Slowly-evolving sources, ie. BHBs early in their inspiral where they appear to not change in frequency.
- Rapidly-evolving sources, ie. BHBs in the final stages of coalescence.
  - Uses a fully Pythonic implementation of the phenomenological model `IMRPhenomD` to accurately represent the inspiral, merger, and ringdown of the BHB.

Calculates the matched-filtered signal-to-noise ratio (SNR) to help assess the detectability of any BHB source configuration by any represented gravitational wave detector.

- Includes robust plotting methods to represent these SNRs.

1.2 Getting Started

`gwent` is available on the Python Package Inventory, so the preferred method to install `gwent` is to install it with `pip`, as it will always install the most recent stable release.

```
$ pip install gwent
```

1.3 README Figure and Data

If you are looking for quick data, we conveniently place the figure above in the data folder on the Github repo. There you can also find the raw data used for this figure in `.npz` format. To load this data, simply use `np.load(filename)`, and the data can be accessed by the kwargs `'mass'`, `'redshift'`, and `'snr'`. E.g.,

```
import numpy as np
import gwent
from gwent.snrplot import Plot_SNR
loaded_file = np.load(filename)
Plot_SNR('M',load_file['mass'],'z',load_file['redshift'],load_file['snr'])
```

1.4 Publication

This work and methodology is available on arXiv. If you use `gwent`, please cite this work using the following:

```
@ARTICLE{Kaiser:2021,
  doi = {10.1088/1361-6382/abd4f6},
  url = {https://doi.org/10.1088/1361-6382/abd4f6},
  year = 2021,
  month = {jan},
  publisher = {{IOP} Publishing},
  volume = {38},
  number = {5},
  pages = {055009},
  author = {A R Kaiser and S T McWilliams},
  title = {Sensitivity of present and future detectors across the black-hole binary␣
↪gravitational wave spectrum},
  journal = {Classical and Quantum Gravity},
  eprint = {2010.02135},
  adsurl = {https://ui.adsabs.harvard.edu/abs/2020arXiv201002135K},
  adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

1.5 Credits

We utilize and include within the package a specific commit of `pygwinc` found at https://git.ligo.org/gwinc/pygwinc to create many of the ground-based gravitational wave detector sensitivity curves. At the time of creation, there is no `pygwinc` availability on PyPI, so we explicitly include the necessary portions of the code within.

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

CHAPTER 2

Installation

2.1 Stable release

To install gwent, run this command in your terminal:

```
$ pip install gwent
```

This is the preferred method to install gwent, 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.

2.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/ark0015/gwent
```

Or download the tarball:

```
$ curl  -OL https://github.com/ark0015/gwent/tarball/master
```

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

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

or use pip to install the requirements within gwent

```
$ pip install -r requirements.txt
```

CHAPTER 3

Usage

To use gwent in a project:

```
import gwent
```

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

CHAPTER 4

Using `gwent` to Generate Characteristic Strain Curves

Here we show examples of using the different classes in `gwent` for various detectors, both loading in from a file and generating with `gwent`, and binary black holes, both in the frequency and time domain.

First, we load important packages

```
import numpy as np

import matplotlib as mpl
import matplotlib.pyplot as plt

from cycler import cycler
from scipy.constants import golden_ratio

import astropy.constants as const
import astropy.units as u

import gwent
import gwent.detector as detector
import gwent.binary as binary

#Turn off warnings for tutorial
import warnings
warnings.filterwarnings('ignore')
```

Setting matplotlib and plotting preferences

```
def get_fig_size(width=7,scale=1.0):
    #width = 3.36 # 242 pt
    base_size = np.array([1, 1/scale/golden_ratio])
    fig_size = width * base_size
    return(fig_size)
mpl.rcParams['figure.dpi'] = 300
mpl.rcParams['figure.figsize'] = get_fig_size()
mpl.rcParams['text.usetex'] = True
```

(continues on next page)

(continued from previous page)

```
mpl.rc('font',**{'family':'serif','serif':['Times New Roman']})
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['axes.labelsize'] = 12
mpl.rcParams['xtick.labelsize'] = 12
mpl.rcParams['ytick.labelsize'] = 12
mpl.rcParams['legend.fontsize'] = 10
color_cycle_wong = ['#000000','#E69F00','#56B4E9','#009E73','#F0E442','#0072B2','
↪#D55E00','#CC79A7','#5a5a5a']
mpl.rcParams['axes.prop_cycle'] = cycler(color=color_cycle_wong)
```

We need to get the file directories to load in the instrument files.

```
load_directory = gwent.__path__[0] + '/LoadFiles'
```

4.1 Initialize different instruments

If loading a detector, the file should be frequency in the first column and either strain, effective strain noise spectral density, or amplitude spectral density in the second column.

For generating a detector, one must assign a value to each of the different instrument parameters (see the section on Declaring x and y variables and Sample Rates).

4.2 Load ground instruments from files

4.2.1 aLIGO

```
Ground_T_obs = 4*u.yr
```

```
#aLIGO
aLIGO_filedirectory = load_directory + '/InstrumentFiles/aLIGO/'
aLIGO_1_filename = 'aLIGODesign.txt'
aLIGO_2_filename = 'ZERO_DET_high_P.txt'

aLIGO_1_filelocation = aLIGO_filedirectory + aLIGO_1_filename
aLIGO_2_filelocation = aLIGO_filedirectory + aLIGO_2_filename

aLIGO_1 = detector.GroundBased('aLIGO 1',Ground_T_obs,load_location=aLIGO_1_
↪filelocation,I_type='A')
aLIGO_2 = detector.GroundBased('aLIGO 2',Ground_T_obs,load_location=aLIGO_2_
↪filelocation,I_type='A')
```

4.2.2 Einstein Telescope

```
#Einstein Telescope
ET_filedirectory = load_directory + '/InstrumentFiles/EinsteinTelescope/'
ET_B_filename = 'ET_B_data.txt'
ET_C_filename = 'ET_C_data.txt'
ET_D_filename = 'ET_D_data.txt'
```

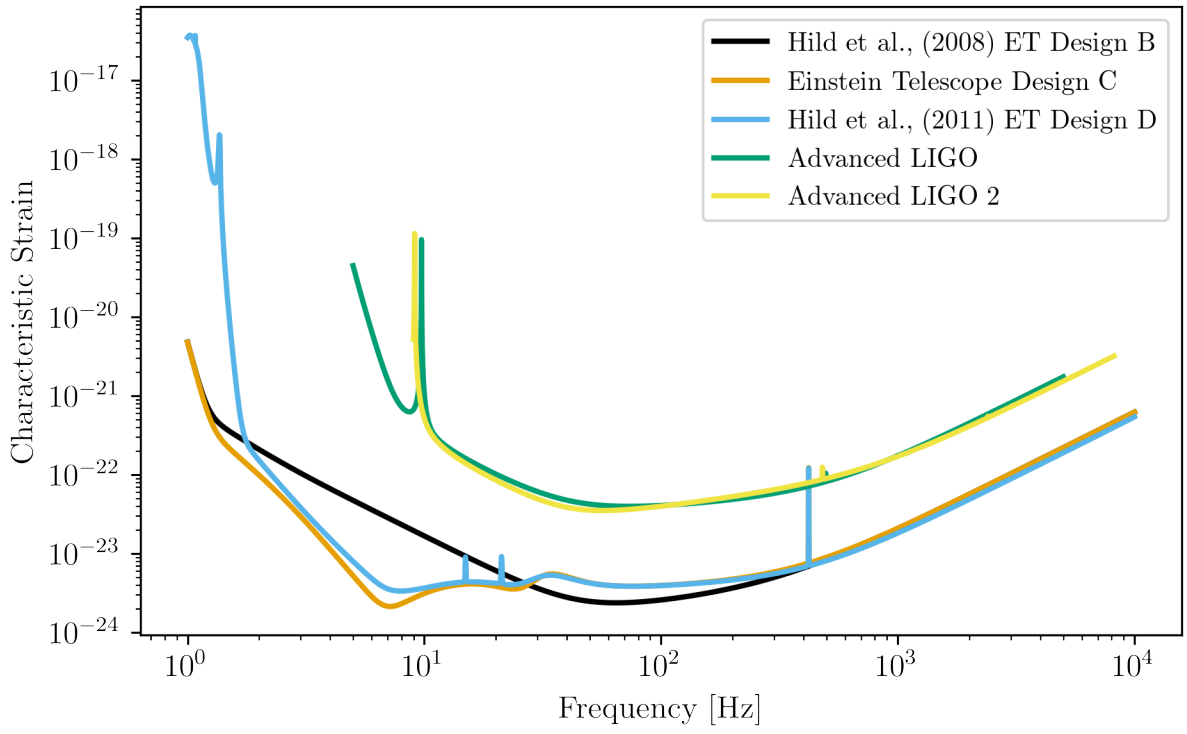
(continues on next page)

(continued from previous page)

```
ET_B_filelocation = ET_filedirectory + ET_B_filename
ET_C_filelocation = ET_filedirectory + ET_C_filename
ET_D_filelocation = ET_filedirectory + ET_D_filename

ET_B = detector.GroundBased('ET',Ground_T_obs,load_location=ET_B_filelocation,I_type=
↪'A')
ET_C = detector.GroundBased('ET',Ground_T_obs,load_location=ET_C_filelocation,I_type=
↪'A')
ET_D = detector.GroundBased('ET',Ground_T_obs,load_location=ET_D_filelocation,I_type=
↪'A')
```

Plots of Ground Detectors

```
fig = plt.figure()
plt.loglog(ET_B.fT,ET_B.h_n_f,label='Hild et al., (2008) ET Design B')
plt.loglog(ET_C.fT,ET_C.h_n_f,label='Einstein Telescope Design C')
plt.loglog(ET_D.fT,ET_D.h_n_f,label='Hild et al., (2011) ET Design D')
plt.loglog(aLIGO_1.fT,aLIGO_1.h_n_f,label='Advanced LIGO')
plt.loglog(aLIGO_2.fT,aLIGO_2.h_n_f,label='Advanced LIGO 2')
plt.xlabel(r'Frequency [Hz]')
plt.ylabel(r'Characteristic Strain')
plt.tick_params(axis = 'both',which = 'major')
plt.legend()
plt.show()
```



4.3 Load LISA Instruments from File

4.3.1 LISA Example 1

Modelled off of the Science Requirements document from https://lisa.nasa.gov/documentsReference.html.

```
SpaceBased_T_obs = 4*u.yr
```

```
LISA_Other_filedirectory = load_directory + '/InstrumentFiles/LISA_Other/'
LISA_ex1_filename = 'LISA_Allocation_S_h_tot.txt'
LISA_ex1_filelocation = LISA_Other_filedirectory + LISA_ex1_filename

#`I_type` should be Effective Noise Spectral Density
LISA_ex1 = detector.SpaceBased('LISA Example 1',SpaceBased_T_obs,load_location=LISA_
↪ex1_filelocation,I_type='E')
```

4.3.2 LISA Example 2

Modelled off of Robson,Cornish,and Liu 2018, LISA (https://arxiv.org/abs/1803.01944).

```
LISA_ex2_filedirectory = load_directory + '/InstrumentFiles/LISA_Other/'
LISA_ex2_filename = 'LISA_sensitivity.txt'
LISA_ex2_filelocation = LISA_ex2_filedirectory + LISA_ex2_filename

#`I_type` should be Effective Noise Spectral Density
LISA_ex2 = detector.SpaceBased('LISA Example 2',SpaceBased_T_obs,load_location=LISA_
↪ex2_filelocation,I_type='E')
```

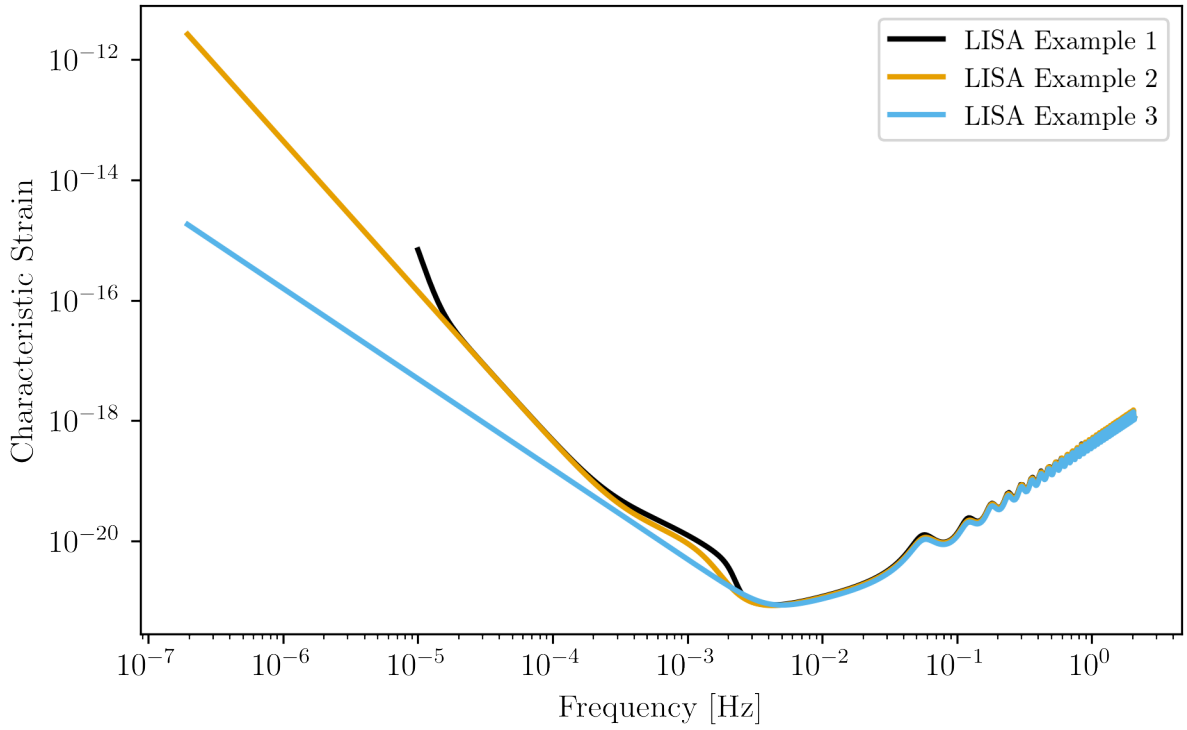
4.3.3 LISA Example 3

Generated by http://www.srl.caltech.edu/~shane/sensitivity/

```
LISA_ex3_filename = 'scg_6981.dat'
LISA_ex3_filelocation = LISA_Other_filedirectory + LISA_ex3_filename

#`I_type` should be Amplitude Spectral Density
LISA_ex3 = detector.SpaceBased('LISA Example 3',SpaceBased_T_obs,load_location=LISA_
↪ex3_filelocation,I_type='A')
```

Plots of loaded LISA examples.

```
fig = plt.figure()
plt.loglog(LISA_ex1.fT,LISA_ex1.h_n_f,label=LISA_ex1.name)
plt.loglog(LISA_ex2.fT,LISA_ex2.h_n_f,label=LISA_ex2.name)
plt.loglog(LISA_ex3.fT,LISA_ex3.h_n_f,label=LISA_ex3.name)
plt.xlabel(r'Frequency [Hz]')
plt.ylabel(r'Characteristic Strain')
plt.tick_params(axis = 'both',which = 'major')
plt.legend()
plt.show()
```



4.4 Loading PTA Detection Curves and Upper Limits

4.4.1 Simulated NANOGrav Continuous Wave Detection Sensitivity

Samples from Mingarelli, et al. 2017 (https://arxiv.org/abs/1708.03491) of the Simulated NANOGrav Continuous Wave Detection Sensitivity.

```
NANOGrav_filedirectory = load_directory + '/InstrumentFiles/NANOGrav/StrainFiles/'
```

```
#NANOGrav continuous wave sensitivity
NANOGrav_background = 4e-16 # Unsubtracted GWB amplitude: 0,4e-16
NANOGrav_dp = 0.95 #Detection Probablility: 0.95,0.5
NANOGrav_fap = 0.0001 #False Alarm Probability: 0.05,0.003,0.001,0.0001
NANOGrav_Tobs = 15 #Observation years: 15,20,25

NANOGrav_filename = 'cw_simulation_Ared_' + str(NANOGrav_background) + '_dp_' +␣
↪str(NANOGrav_dp) \
                    + '_fap_' + str(NANOGrav_fap) + '_T_' + str(NANOGrav_Tobs) + '.txt
↪'
NANOGrav_filelocation = NANOGrav_filedirectory + NANOGrav_filename

NANOGrav_cw_GWB = detector.PTA('NANOGrav CW Detection w/ GWB',load_location=NANOGrav_
↪filelocation,I_type='h')
```

```
#NANOGrav continuous wave sensitivity
NANOGrav_background_2 = 0 # Unsubtracted GWB amplitude: 0,4e-16
NANOGrav_dp_2 = 0.95 #Detection Probablility: 0.95,0.5
NANOGrav_fap_2 = 0.0001 #False Alarm Probability: 0.05,0.003,0.001,0.0001
```

(continues on next page)

(continued from previous page)

```
NANOGrav_Tobs_2 = 15 #Observation years: 15,20,25

NANOGrav_filename_2 = 'cw_simulation_Ared_' + str(NANOGrav_background_2) + '_dp_' +␣
↪str(NANOGrav_dp_2) \
                    + '_fap_' + str(NANOGrav_fap_2) + '_T_' + str(NANOGrav_Tobs_2) +
↪'.txt'
NANOGrav_filelocation_2 = NANOGrav_filedirectory + NANOGrav_filename_2

NANOGrav_cw_no_GWB = detector.PTA('NANOGrav CW Detection no GWB',load_
↪location=NANOGrav_filelocation_2,I_type='h')
```

4.4.2 NANOGrav Continuous Wave 11yr Upper Limit

Sample from Aggarwal, et al. 2019 (https://arxiv.org/abs/1812.11585) of the NANOGrav 11yr continuous wave upper limit.

```
NANOGrav_cw_ul_file = NANOGrav_filedirectory + 'smoothed_11yr.txt'
NANOGrav_cw_ul = detector.PTA('NANOGrav CW Upper Limit',load_location=NANOGrav_cw_ul_
↪file,I_type='h')
```

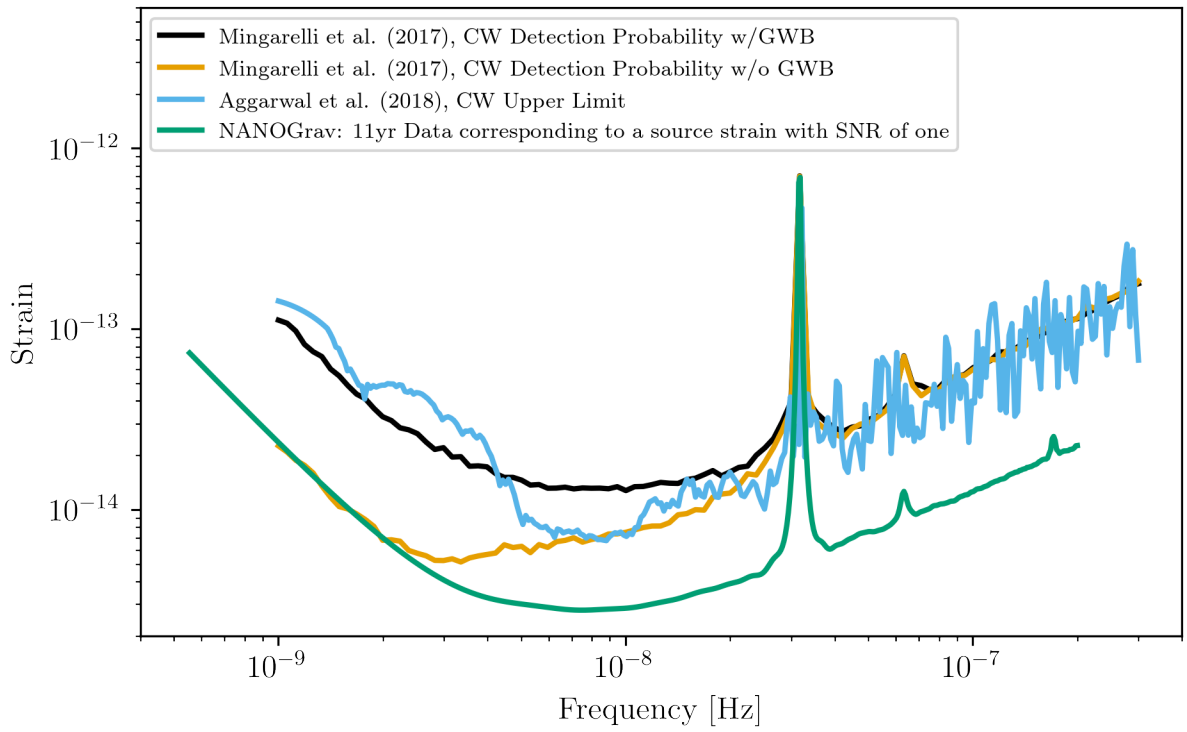
4.4.3 NANOGrav 11yr Characteristic Strain

Using real NANOGrav 11yr data put through `hasasia`. We need to initialize and fill the values used in the plots (i.e., `NANOGrav_11yr_hasasia.T_obs` isn't known until we set the values since we loaded it from a file.

```
NANOGrav_11yr_hasasia_file = NANOGrav_filedirectory + 'NANOGrav_11yr_S_eff.txt'
NANOGrav_11yr_hasasia = detector.PTA('NANOGrav 11yr',load_location=NANOGrav_11yr_
↪hasasia_file,I_type='E')
NANOGrav_11yr_hasasia.T_obs = 11.4*u.yr
```

Plots of the loaded PTAs

```
fig = plt.figure()
plt.loglog(NANOGrav_cw_GWB.fT,NANOGrav_cw_GWB.h_n_f,label = r'Mingarelli et al.␣
↪(2017), CW Detection Probability w/GWB')
plt.loglog(NANOGrav_cw_no_GWB.fT,NANOGrav_cw_no_GWB.h_n_f, label =r'Mingarelli et al.␣
↪(2017), CW Detection Probability w/o GWB')
plt.loglog(NANOGrav_cw_ul.fT,NANOGrav_cw_ul.h_n_f, label = r'Aggarwal et al. (2018),␣
↪CW Upper Limit')
plt.loglog(NANOGrav_11yr_hasasia.fT,np.sqrt(NANOGrav_11yr_hasasia.S_n_f/np.max(np.
↪unique(NANOGrav_11yr_hasasia.T_obs.to('s').value))),
           label = r'NANOGrav: 11yr Data  corresponding to a source strain with SNR␣
↪of one')

plt.tick_params(axis = 'both',which = 'major')
plt.ylim([2e-15,6e-12])
plt.xlim([4e-10,4e-7])
plt.xlabel(r'Frequency [Hz]')
plt.ylabel('Strain')
plt.legend(loc='upper left',fontsize=8)
plt.show()
```



4.5 Generating PTAs with `gwent`

Generated using the code `hasasia` (https://hasasia.readthedocs.io/en/latest/) via the methods of Hazboun, Romano, and Smith, 2019 (https://arxiv.org/abs/1907.04341)

4.5.1 SKA-esque Detector

Fiducial parameter estimates from Sesana, Vecchio, and Colacino, 2008 (https://arxiv.org/abs/0804.4476) section 7.1.

```
sigma_SKA = 10*u.ns.to('s')*u.s #sigma_rms timing residuals in nanoseconds to seconds
T_SKA = 15*u.yr #Observing time in years
N_p_SKA = 20 #Number of pulsars
cadence_SKA = 1/(u.wk.to('yr')*u.yr) #Avg observation cadence of 1 every week in␣
↪[number/yr]
```

SKA with White noise only

```
SKA_WN = detector.PTA('SKA, WN Only',N_p_SKA,T_obs=T_SKA,sigma=sigma_SKA,
↪cadence=cadence_SKA)
```

SKA with White and Varied Red Noise

```
SKA_WN_RN = detector.PTA('SKA, WN and RN',N_p_SKA,T_obs=T_SKA,sigma=sigma_SKA,
↪cadence=cadence_SKA,
                         rn_amp=[1e-16,1e-12],rn_alpha=[-1/2,1.25])
```

SKA with White Noise and a Stochastic Gravitational Wave Background

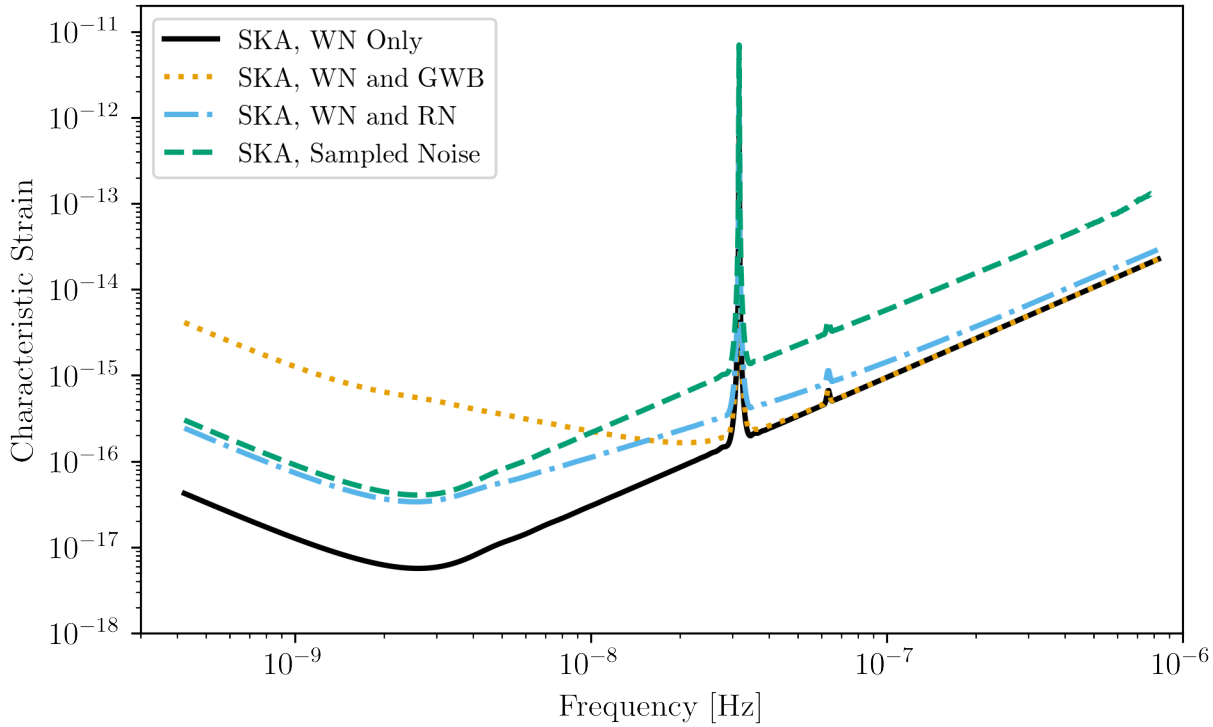
```
SKA_WN_GWB = detector.PTA('SKA, WN and GWB',N_p_SKA,T_obs=T_SKA,sigma=sigma_SKA,
↪cadence=cadence_SKA,
                          sb_amp=4e-16,sb_alpha=-2/3)
```

SKA with Sampled Noise for each pulsar, no GWB

```
SKA_Sampled_Noise = detector.PTA('SKA, Sampled Noise',N_p_SKA,cadence=[cadence_SKA,
↪cadence_SKA/4.],
                                 sigma=[sigma_SKA,10*sigma_SKA],T_obs=T_SKA,use_
↪11yr=True,use_rn=True)
```

Plots for Simulated SKA PTAs

```
fig = plt.figure()
plt.loglog(SKA_WN.fT,SKA_WN.h_n_f,label = SKA_WN.name)
plt.loglog(SKA_WN_GWB.fT,SKA_WN_GWB.h_n_f, linestyle=':',label = SKA_WN_GWB.name)
plt.loglog(SKA_WN_RN.fT,SKA_WN_RN.h_n_f, linestyle='-.',label = SKA_WN_RN.name)
plt.loglog(SKA_Sampled_Noise.fT,SKA_Sampled_Noise.h_n_f,linestyle='--',label=SKA_
↪Sampled_Noise.name)

plt.tick_params(axis = 'both',which = 'major')
plt.ylim([1e-18,2e-11])
plt.xlim([3e-10,1e-6])

plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain')
plt.legend(loc='upper left')
plt.show()
```



4.5.2 NANOGrav-esque Detector

Fiducial 11yr parameter estimates from Arzoumanian, et al., 2018 https://arxiv.org/abs/1801.01837

```
################################################
#NANOGrav calculation using 11.5yr parameters https://arxiv.org/abs/1801.01837
sigma_nano = 100*u.ns.to('s')*u.s #rms timing residuals in nanoseconds to seconds
T_nano = 11.4*u.yr #Observing time in years
N_p_nano = 34 #Number of pulsars
cadence_nano = 1/(2*u.wk.to('yr')*u.yr) #Avg observation cadence of 1 every 2 weeks␣
↪in number/year
```

NANOGrav with White Noise only

```
NANOGrav_WN = detector.PTA('NANOGrav, WN Only',N_p_nano,T_obs=T_nano,sigma=sigma_nano,
↪cadence=cadence_nano)
```

NANOGrav with White and Varied Red Noise

```
NANOGrav_WN_RN = detector.PTA('NANOGrav, WN and RN',N_p_nano,T_obs=T_nano,sigma=sigma_
↪nano,cadence=cadence_nano,
                              rn_amp=[1e-16,1e-12],rn_alpha=[-1/2,1.25])
```

NANOGrav with White Noise and a Stochastic Gravitational Wave Background

```
NANOGrav_WN_GWB = detector.PTA('NANOGrav, WN and GWB',N_p_nano,
                               T_obs=T_nano,sigma=sigma_nano,cadence=cadence_nano,sb_
↪amp=4e-16)
```

NANOGrav with Sampled Noise for each pulsar, no GWB

```
NANOGrav_Sampled_Noise = detector.PTA('NANOGrav, Sampled Noise',N_p_nano,use_
↪11yr=True,use_rn=True)
```

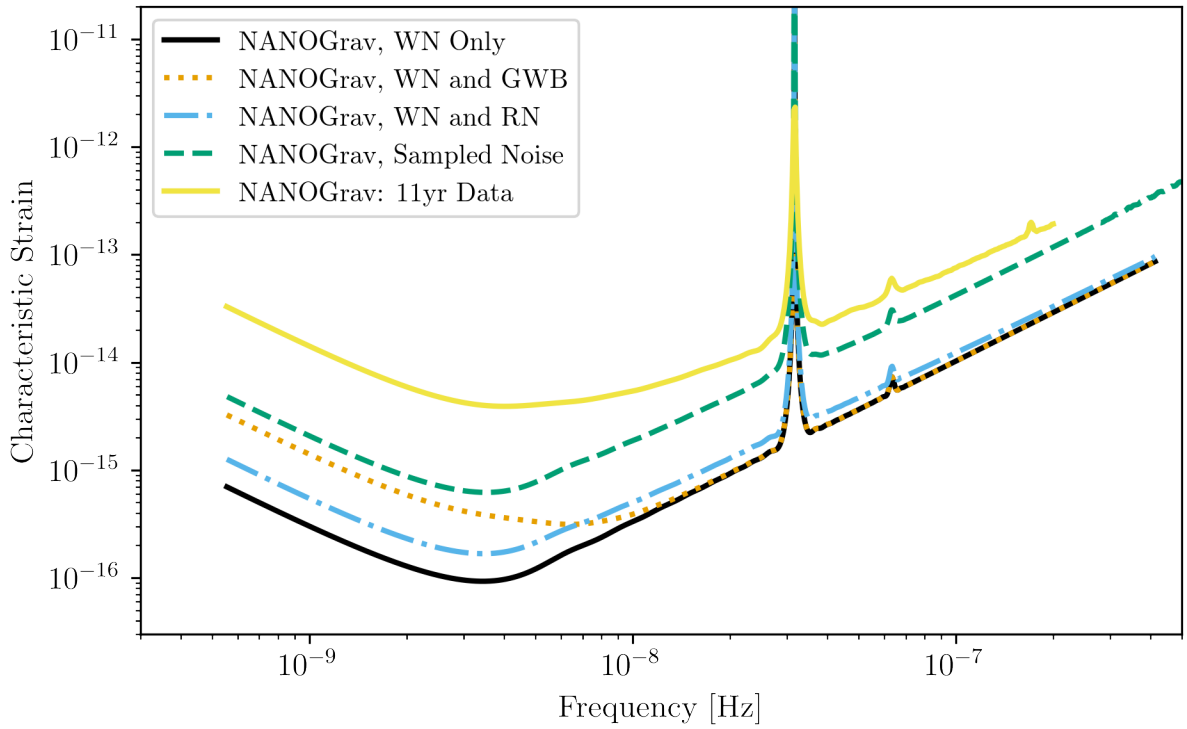
Plots for Simulated NANOGrav PTAs

```
fig = plt.figure()
plt.loglog(NANOGrav_WN.fT,NANOGrav_WN.h_n_f,
           label=NANOGrav_WN.name)
plt.loglog(NANOGrav_WN_GWB.fT,NANOGrav_WN_GWB.h_n_f,
           linestyle=':',label=NANOGrav_WN_GWB.name)
plt.loglog(NANOGrav_WN_RN.fT,NANOGrav_WN_RN.h_n_f,
           linestyle='-.',label=NANOGrav_WN_RN.name)
plt.loglog(NANOGrav_Sampled_Noise.fT,NANOGrav_Sampled_Noise.h_n_f,
           linestyle='--',label=NANOGrav_Sampled_Noise.name)

plt.loglog(NANOGrav_11yr_hasasia.fT,NANOGrav_11yr_hasasia.h_n_f,
           label = r'NANOGrav: 11yr Data')

plt.tick_params(axis = 'both',which = 'major')
plt.ylim([3e-17,2e-11])
plt.xlim([3e-10,5e-7])

plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain')
```

(continues on next page)

(continued from previous page)

```
plt.legend(loc='upper left')
plt.show()
```



4.6 Generating LISA designs with `gwent`

First we set a fiducial armlength and observation time-length

```
L = 2.5*u.Gm  #armlength in Gm
L = L.to('m')
LISA_T_obs = 4*u.yr
```

4.6.1 LISA Proposal 1

Values taken from the ESA L3 proposal, Amaro-Seaone, et al., 2017 (https://arxiv.org/abs/1702.00786)

```
f_acc_break_low = .4*u.mHz.to('Hz')*u.Hz
f_acc_break_high = 8.*u.mHz.to('Hz')*u.Hz
f_IMS_break = 2.*u.mHz.to('Hz')*u.Hz
A_acc = 3e-15*u.m/u.s/u.s
A_IMS = 10e-12*u.m

Background = False

LISA_prop1 = detector.SpaceBased('LISA',\
```

(continues on next page)

(continued from previous page)

```
                              LISA_T_obs,L,A_acc,f_acc_break_low,f_acc_break_high,A_IMS,f_
↪IMS_break,\
                              Background=Background)
```

4.6.2 LISA Proposal 1 with Galactic Binary Background

Values taken from the ESA L3 proposal, Amaro-Seaone, et al., 2017 (https://arxiv.org/abs/1702.00786)

```
f_acc_break_low = .4*u.mHz.to('Hz')*u.Hz
f_acc_break_high = 8.*u.mHz.to('Hz')*u.Hz
f_IMS_break = 2.*u.mHz.to('Hz')*u.Hz
A_acc = 3e-15*u.m/u.s/u.s
A_IMS = 10e-12*u.m

Background = True

LISA_prop1_w_background = detector.SpaceBased('LISA w/Background',\
                              LISA_T_obs,L,A_acc,f_acc_break_low,f_acc_break_high,A_IMS,f_
↪IMS_break,\
                              Background=Background)
```

4.6.3 LISA Proposal 2

Values from Robson, Cornish, and Liu 2019 https://arxiv.org/abs/1803.01944 using the Transfer Function Approximation within. (Note the factor of 2 change from summing 2 independent low-frequency data channels assumed in the paper.)

```
f_acc_break_low = .4*u.mHz.to('Hz')*u.Hz
f_acc_break_high = 8.*u.mHz.to('Hz')*u.Hz
f_IMS_break = 2.*u.mHz.to('Hz')*u.Hz
A_acc = 3e-15*u.m/u.s/u.s
A_IMS = 1.5e-11*u.m
Background = False

LISA_prop2 = detector.SpaceBased('LISA Approximate',
                                 LISA_T_obs,L,A_acc,f_acc_break_low,f_acc_break_high,
↪A_IMS,f_IMS_break,
                                 Background=Background,T_type='A')
```

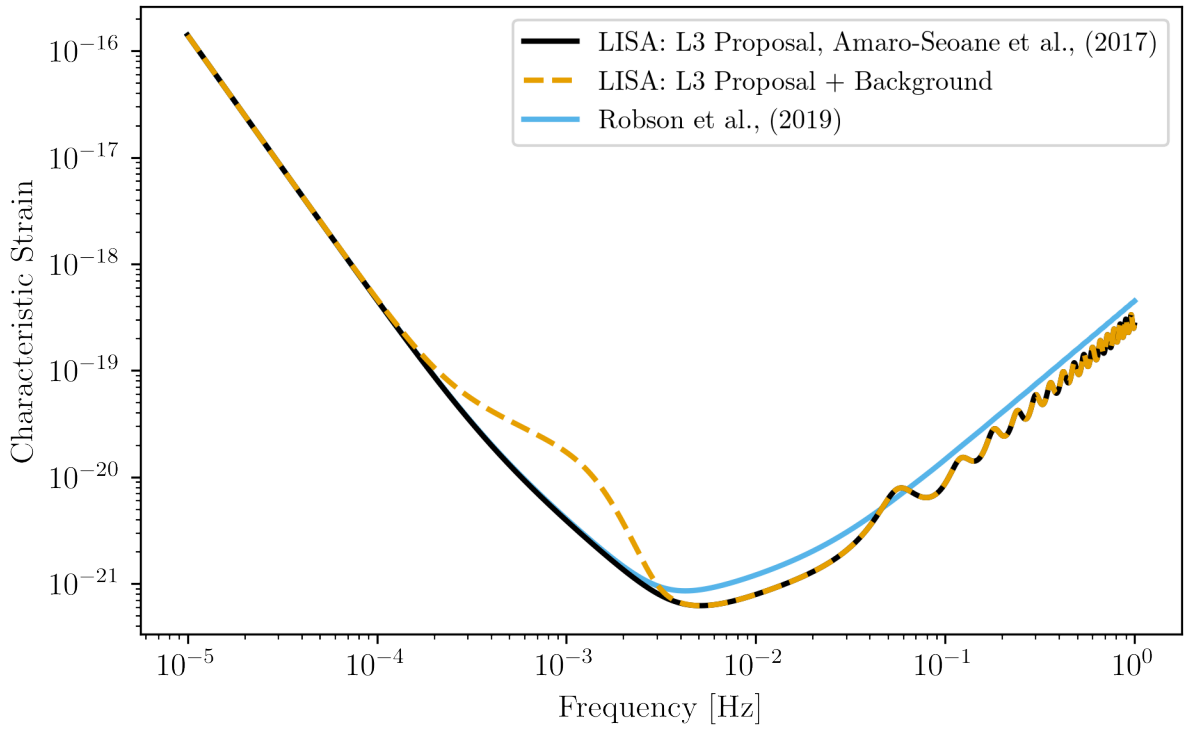
Plots of Generated LISA Detectors

```
fig = plt.figure()
plt.loglog(LISA_prop1.fT,LISA_prop1.h_n_f,label=r'LISA: L3 Proposal, Amaro-Seoane et␣
↪al., (2017)')
plt.loglog(LISA_prop1_w_background.fT,LISA_prop1_w_background.h_n_f,label=r'LISA: L3␣
↪Proposal + Background',
           linestyle='--')
plt.loglog(LISA_prop2.fT,LISA_prop2.h_n_f,label=r'Robson et al., (2019)',zorder=-1)
plt.xlabel(r'Frequency [Hz]')
plt.ylabel(r'Characteristic Strain')
plt.tick_params(axis = 'both',which = 'major')
```

(continues on next page)

(continued from previous page)

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



4.7 Generating Ground Based Detector Designs with `gwent`

First we set a fiducial observation time-length

```
Ground_T_obs = 4*u.yr
```

4.7.1 aLIGO

```
aLIGO_prop1 = detector.GroundBased('aLIGO',Ground_T_obs,f_low=min(aLIGO_1.fT),f_
↪high=max(aLIGO_1.fT))
```

If one wanted to change the parameters from the fiducial values, you could set up a new noise dictionary, then initialize that instument with the new values. It also works for updating the current instrument values.

```
noise_dict = {'Infrastructure':
                {'Length':2500},
              'Materials':
                {'Substrate':{'Temp':500}}}
aLIGO_prop2 = detector.GroundBased('aLIGO prop 2',Ground_T_obs,noise_dict=noise_dict)
```

4.7.2 A+

```
Aplus_prop1 = detector.GroundBased('Aplus',Ground_T_obs,f_low=min(aLIGO_1.fT),f_
↪high=max(aLIGO_1.fT))
```

If you want to see what the current instrument parameters are, and what you can vary, you can use the `instrument.Get_Noise_Dict()`. To access each parameter, you must make a noise dictionary like above that matches the depth of the parameter you wish to change.

```
Aplus_prop1.Get_Noise_Dict()
```

```
Infrastructure Parameters:
     Length : 3995
     Temp : 290
     ResidualGas Subparameters:
          pressure : 4e-07
          mass : 3.35e-27
          polarizability : 7.8e-31
TCS Parameters:
     s_cc : 7.024
     s_cs : 7.321
     s_ss : 7.631
     SRCloss : 0.0
Seismic Parameters:
     Site : LHO
     KneeFrequency : 10
     LowFrequencyLevel : 1e-09
     Gamma : 0.8
     Rho : 1800.0
     Beta : 0.8
     Omicron : 1
     TestMassHeight : 1.5
     RayleighWaveSpeed : 250
Suspension Parameters:
     Type : Quad
     FiberType : Tapered
     BreakStress : 750000000.0
     Temp : 290
     Silica Subparameters:
          Rho : 2200.0
          C : 772
          K : 1.38
          Alpha : 3.9e-07
          dlnEdT : 0.000152
          Phi : 4.1e-10
          Y : 72000000000.0
          Dissdepth : 0.015
     C70Steel Subparameters:
          Rho : 7800
          C : 486
          K : 49
          Alpha : 1.2e-05
          dlnEdT : -0.00025
          Phi : 0.0002
          Y : 212000000000.0
     MaragingSteel Subparameters:
          Rho : 7800
```

(continues on next page)

(continued from previous page)

```
          C : 460
          K : 20
          Alpha : 1.1e-05
          dlnEdT : 0
          Phi : 0.0001
          Y : 187000000000.0
     Silicon Subparameters:
          Rho : 2329
          C : 300
          K : 700
          Alpha : 1e-10
          dlnEdT : -2e-05
          Phi : 2e-09
          Y : 155800000000.0
          Dissdepth : 0.0015
     Stage : array of shape 4
     Ribbon Subparameters:
          Thickness : 0.000115
          Width : 0.00115
     Fiber Subparameters:
          Radius : 0.000205
          EndRadius : 0.0004
          EndLength : 0.045
     VHCoupling Subparameters:
          theta : 0.0006263620827519167
     hForce : array of shape (1000,)
     vForce : array of shape (1000,)
     hForce_singlylossy : array of shape (4, 1000)
     vForce_singlylossy : array of shape (4, 1000)
     hTable : array of shape (1000,)
     vTable : array of shape (1000,)
Materials Parameters:
     MassRadius : 0.17
     MassThickness : 0.2
     Coating Subparameters:
          Yhighn : 124000000000.0
          Sigmahighn : 0.28
          CVhighn : 2100000.0
          Alphahighn : 3.6e-06
          Betahighn : 1.4e-05
          ThermalDiffusivityhighn : 33
          Indexhighn : 2.06539
          Phihighn : 9e-05
          Phihighn_slope : 0.1
          Ylown : 72000000000.0
          Sigmalown : 0.17
          CVlown : 1641200.0
          Alphalown : 5.1e-07
          Betalown : 8e-06
          ThermalDiffusivitylown : 1.38
          Indexlown : 1.45
          Philown : 1.25e-05
          Philown_slope : 0.4
     Substrate Subparameters:
          Temp : 295
          c2 : 7.6e-12
          MechanicalLossExponent : 0.77
```

(continues on next page)

(continued from previous page)

```
          Alphas : 5.2e-12
          MirrorY : 72700000000.0
          MirrorSigma : 0.167
          MassDensity : 2200.0
          MassAlpha : 3.9e-07
          MassCM : 739
          MassKappa : 1.38
          RefractiveIndex : 1.45
     MirrorVolume : 0.01815840553774901
     MirrorMass : 39.948492183047826
Laser Parameters:
     Wavelength : 1.064e-06
     Power : 125
Optics Parameters:
     Type : SignalRecycled
     PhotoDetectorEfficiency : 0.9
     Loss : 3.75e-05
     BSLoss : 0.0005
     coupling : 1.0
     SubstrateAbsorption : 5e-05
     pcrit : 10
     Quadrature Subparameters:
          dc : 1.5707963
     ITM Subparameters:
          Transmittance : 0.014
          CoatingThicknessLown : 0.308
          CoatingThicknessCap : 0.5
          CoatingAbsorption : 5e-07
          Thickness : 0.2
          CoatLayerOpticalThickness : array of shape (16,)
          BeamRadius : 0.05549089680470938
     ETM Subparameters:
          Transmittance : 5e-06
          CoatingThicknessLown : 0.27
          CoatingThicknessCap : 0.5
          CoatLayerOpticalThickness : array of shape (38,)
          BeamRadius : 0.06203311014519086
     PRM Subparameters:
          Transmittance : 0.03
     SRM Subparameters:
          Transmittance : 0.325
          CavityLength : 55
          Tunephase : 0.0
     Curvature Subparameters:
          ITM : 1970
          ETM : 2192
Squeezer Parameters:
     Type : Freq Dependent
     AmplitudedB : 12
     InjectionLoss : 0.05
     SQZAngle : 0
     LOAngleRMS : 0.03
     FilterCavity Subparameters:
          L : 300
          Te : 1e-06
          Lrt : 6e-05
          Rot : 0
```

(continues on next page)

(continued from previous page)

```
          fdetune : -45.78
          Ti : 0.0012
gwinc Parameters:
     PRfixed : True
     pbs : 5351.309810308315
     parm : 750599.8555500002
     finesse : 446.4074818600061
     prfactor : 42.81047848246652
     gITM : -1.0279187817258881
     gETM : -0.822536496350365
     BeamWaist : 0.011750961823848846
     BeamRayleighRange : 407.7134846079674
     BeamWaistToITM : 1881.657523510972
     BeamWaistToETM : 2113.3424764890283
     dhdl_sqr : array of shape (1000,)
     sinc_sqr : array of shape (1000,)

Number of Variables:  150
```

4.7.3 Voyager

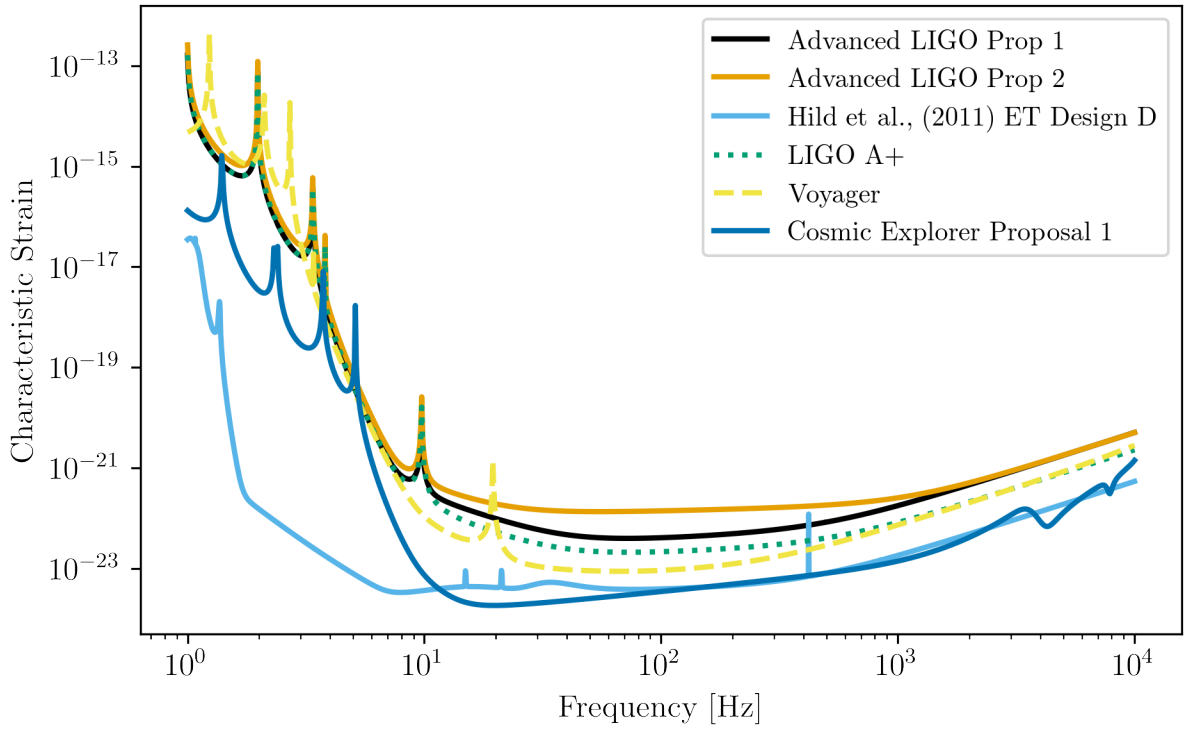
```
Voyager_prop1 = detector.GroundBased('Voyager',Ground_T_obs)
```

4.7.4 Cosmic Explorer

```
CE1_prop1 = detector.GroundBased('CE1',Ground_T_obs)
```

Plots of Generated Ground Based Detectors

```
fig = plt.figure()
plt.loglog(aLIGO_prop1.fT,aLIGO_prop1.h_n_f,label='Advanced LIGO Prop 1')
plt.loglog(aLIGO_prop2.fT,aLIGO_prop2.h_n_f,label='Advanced LIGO Prop 2')

plt.loglog(ET_D.fT,ET_D.h_n_f,label='Hild et al., (2011) ET Design D')
plt.loglog(Aplus_prop1.fT,Aplus_prop1.h_n_f,label='LIGO A+',
           linestyle=':')
plt.loglog(Voyager_prop1.fT,Voyager_prop1.h_n_f,label='Voyager',
           linestyle='--')
plt.loglog(CE1_prop1.fT,CE1_prop1.h_n_f,label='Cosmic Explorer Proposal 1')
plt.xlabel(r'Frequency [Hz]')
plt.ylabel(r'Characteristic Strain')
plt.tick_params(axis = 'both',which = 'major')
plt.legend()
plt.show()
```



4.8 Generating Binary Black Holes with `gwent` in the Frequency Domain

We start with BBH parameters that exemplify the range of IMRPhenomD's waveforms from Khan, et al. 2016 https://arxiv.org/abs/1508.07253 and Husa, et al. 2016 https://arxiv.org/abs/1508.07250

For more information see the tutorial on source strains.

```
M = [1e6,65.0,1e10]
q = [1.0,18.0,1.0]
x1 = [0.5,0.0,-0.95]
x2 = [0.2,0.0,-0.95]
z = [3.0,0.093,20.0]
```

Uses the first parameter values and the `LISA_prop1` detector model for the observation time with the precessing phenomenological `lalsuite` waveform IMRPhenomPv3.

```
lalsuite_kwargs = {"S1x": 0.5, "S1y": 0., "S1z": x1[0],
                   "S2x": -0.2, "S2y": 0.5, "S2z": x2[0],
                   "inclination":np.pi/2}
source_1 = binary.BBHFrequencyDomain(M[0],q[0],z[0],instrument=LISA_prop1,
                                     approximant='IMRPhenomPv3',lalsuite_
↪kwargs=lalsuite_kwargs)
```

Uses the first parameter values and the `aLIGO` detector model for the observation time.

```
source_2 = binary.BBHFrequencyDomain(M[1],q[1],z[1],x1[1],x2[1],instrument=aLIGO_1)
```

Uses the first parameter values and the `SKA_WN` detector model for the observation time.

```
source_3 = binary.BBHFrequencyDomain(M[2],q[2],z[2],x1[2],x2[2],instrument=SKA_WN)
```

To display the time it takes for each source to evolve, we find several markers in time: 200 years prior to merger, `T_obs` until merger, and one year until merger. In each call, we assume the time to merger is in the observer frame (i.e., `in_frame = 'observer'`)

```
t_year = u.yr.to('s')*u.s
t_200_year = 200.*t_year
```

```
#Source 1
source_1_t_200_year_f = binary.Get_Source_Freq(source_1,t_200_year,
                                               in_frame='observer',out_frame='observer
↪')
idx1 = np.abs(source_1.f-source_1_t_200_year_f).argmin()
source_1_t_200_year_h = binary.Get_Char_Strain(source_1)[idx1]

source_1_t_year_f = binary.Get_Source_Freq(source_1,t_year,
                                           in_frame='observer',out_frame='observer')
idx2 = np.abs(source_1.f-source_1_t_year_f).argmin()
source_1_t_year_h = binary.Get_Char_Strain(source_1)[idx2]

source_1_t_T_obs_f = binary.Get_Source_Freq(source_1,source_1.instrument.T_obs,
                                            in_frame='observer',out_frame='observer')
idx3 = np.abs(source_1.f-source_1_t_T_obs_f).argmin()
source_1_t_T_obs_h = binary.Get_Char_Strain(source_1)[idx3]
```

```
#Source 2
source_2_t_200_year_f = binary.Get_Source_Freq(source_2,t_200_year,
                                               in_frame='observer',out_frame='observer
↪')
idx4 = np.abs(source_2.f-source_2_t_200_year_f).argmin()
source_2_t_200_year_h = binary.Get_Char_Strain(source_2)[idx4]

source_2_t_year_f = binary.Get_Source_Freq(source_2,t_year,
                                           in_frame='observer',out_frame='observer')
idx5 = np.abs(source_2.f-source_2_t_year_f).argmin()
source_2_t_year_h = binary.Get_Char_Strain(source_2)[idx5]

source_2_t_T_obs_f = binary.Get_Source_Freq(source_2,source_2.instrument.T_obs,
                                            in_frame='observer',out_frame='observer')
idx6 = np.abs(source_2.f-source_2_t_T_obs_f).argmin()
source_2_t_T_obs_h = binary.Get_Char_Strain(source_2)[idx6]
```

```
#Source 3
source_3_t_200_year_f = binary.Get_Source_Freq(source_3,t_200_year,
                                               in_frame='observer',out_frame='observer
↪')
idx7 = np.abs(source_3.f-source_3_t_200_year_f).argmin()
source_3_t_200_year_h = binary.Get_Char_Strain(source_3)[idx7]

source_3_t_year_f = binary.Get_Source_Freq(source_3,t_year,
                                           in_frame='observer',out_frame='observer')
idx8 = np.abs(source_3.f-source_3_t_year_f).argmin()
source_3_t_year_h = binary.Get_Char_Strain(source_3)[idx8]

source_3_t_T_obs_f = binary.Get_Source_Freq(source_3,np.unique(np.max(source_3.
↪instrument.T_obs)),
```

(continues on next page)

(continued from previous page)

```
                                                in_frame='observer',out_frame='observer')
idx9 = np.abs(source_3.f-source_3_t_T_obs_f).argmin()
source_3_t_T_obs_h = binary.Get_Char_Strain(source_3)[idx9]
```

4.9 Plots of Entire GW Band

Displays only generated detectors: WN only PTAs, ESA L3 proposal LISA, aLIGO, and Einstein Telescope.

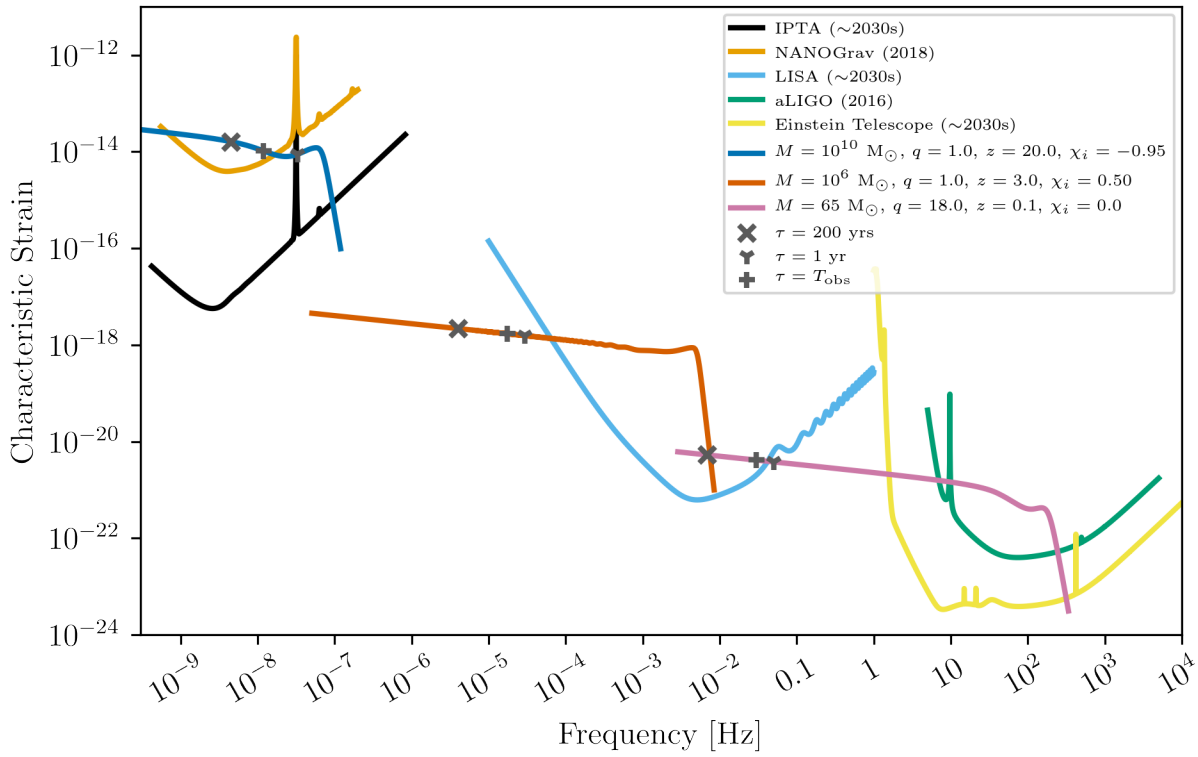
Displays three sources' waveform along with their monochromatic strain if they were observed by the initialized instrument at the detector's most sensitive frequency throughout its observing run (from left to right: `SKA_WN`,`LISA_prop1`,`ET`).

```
fig,ax = plt.subplots()
zord = 10.

ax.loglog(SKA_WN.fT,SKA_WN.h_n_f,label = r'IPTA ($\sim$2030s)')
ax.loglog(NANOGrav_11yr_hasasia.fT,NANOGrav_11yr_hasasia.h_n_f,label = 'NANOGrav␣
↪(2018)')
ax.loglog(LISA_prop1.fT,LISA_prop1.h_n_f,label = 'LISA ($\sim$2030s)')
ax.loglog(aLIGO_1.fT,aLIGO_1.h_n_f,label = 'aLIGO (2016)')
ax.loglog(ET_D.fT,ET_D.h_n_f,label = 'Einstein Telescope ($\sim$2030s)')

ax.loglog(source_3.f,binary.Get_Char_Strain(source_3),
          label = r'$M = 10^{%.0f}$ $\mathrm{M}_{\odot}$, $q = %.1f$, $z = %.1f$,
↪$\chi_{i} = %.2f$' %(np.log10(M[2]),q[2],z[2],x1[2]))

ax.loglog(source_1.f,binary.Get_Char_Strain(source_1),
          label = r'$M = 10^{%.0f}$ $\mathrm{M}_{\odot}$, $q = %.1f$, $z = %.1f$,
↪$\chi_{i} = %.2f$' %(np.log10(M[0]),q[0],z[0],x1[0]))

ax.loglog(source_2.f,binary.Get_Char_Strain(source_2),
          label = r'$M = %.0f$ $\mathrm{M}_{\odot}$, $q = %.1f$, $z = %.1f$, $\chi_{i}
↪ = %.1f$' %(M[1],q[1],z[1],x1[1]))

#Source 1
ax.scatter(source_1_t_200_year_f.value,source_1_t_200_year_h,color='C8',zorder=zord,
↪marker='x',
           label=r'$\tau = %.0f$ yrs' %t_200_year.to('yr').value)
ax.scatter(source_1_t_year_f.value,source_1_t_year_h,color='C8',zorder=zord,marker='1
↪',
           label=r'$\tau = %.0f$ yr' %t_year.to('yr').value)
ax.scatter(source_1_t_T_obs_f.value,source_1_t_T_obs_h,color='C8',zorder=zord,marker=
↪'+',
           label=r'$\tau = T_{\mathrm{obs}}$')
#Source 2
ax.scatter(source_2_t_200_year_f.value,source_2_t_200_year_h,color='C8',zorder=zord,
↪marker='x')
ax.scatter(source_2_t_year_f.value,source_2_t_year_h,color='C8',zorder=zord,marker='1
↪')
ax.scatter(source_2_t_T_obs_f.value,source_2_t_T_obs_h,color='C8',zorder=zord,marker=
↪'+')
#Source 3
ax.scatter(source_3_t_200_year_f.value,source_3_t_200_year_h,color='C8',zorder=zord,
↪marker='x')
```

(continues on next page)

(continued from previous page)

```
ax.scatter(source_3_t_year_f.value,source_3_t_year_h,color='C8',zorder=zord,marker='1
↪')
ax.scatter(source_3_t_T_obs_f.value,source_3_t_T_obs_h,color='C8',zorder=zord,marker=
↪'+')

xlabel_min = -10
xlabel_max = 4
xlabels = np.arange(xlabel_min,xlabel_max+1)
#xlabels = xlabels[1::2]

ax.set_xticks(10.**xlabels)
print_xlabels = []
for x in xlabels:
    if abs(x) > 1:
        print_xlabels.append(r'$10^{%i}$' %x)
    elif x == -1:
        print_xlabels.append(r'$%.1f$' %10.**x)
    else:
        print_xlabels.append(r'$%.0f$' %10.**x)
ax.set_xticklabels([label for label in print_xlabels],rotation=30)

ax.set_xlim([3e-10, 1e4])
ax.set_ylim([1e-24, 1e-11])

ax.set_xlabel('Frequency [Hz]')
ax.set_ylabel('Characteristic Strain')
ax.legend(loc='upper right',fontsize=6)
plt.show()
```



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

CHAPTER 5

Using `gwent` to Generate Source Characteristic Strain Curves

Here we show examples of using the different classes in `gwent` for various black holes binaries (BHBs), both in the frequency and time domain.

First, we load important packages

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

from cycler import cycler
from scipy.constants import golden_ratio
import astropy.units as u

import gwent
import gwent.detector as detector
import gwent.binary as binary
import gwent.snr as snr

#Turn off warnings for tutorial
import warnings
warnings.filterwarnings('ignore')
```

Setting matplotlib and plotting preferences

```
def get_fig_size(width=7,scale=1.0):
    #width = 3.36 # 242 pt
    base_size = np.array([1, 1/scale/golden_ratio])
    fig_size = width * base_size
    return(fig_size)
mpl.rcParams['figure.dpi'] = 300
mpl.rcParams['figure.figsize'] = get_fig_size()
mpl.rcParams['text.usetex'] = True
mpl.rc('font',**{'family':'serif','serif':['Times New Roman']})
mpl.rcParams['lines.linewidth'] = 2
```

(continues on next page)

(continued from previous page)

```
mpl.rcParams['axes.labelsize'] = 12
mpl.rcParams['xtick.labelsize'] = 12
mpl.rcParams['ytick.labelsize'] = 12
mpl.rcParams['legend.fontsize'] = 10
color_cycle_wong = ['#000000','#E69F00','#56B4E9','#009E73','#F0E442','#0072B2','
↪#D55E00','#CC79A7']
mpl.rcParams['axes.prop_cycle'] = cycler(color=color_cycle_wong)
```

We need to get the file directories to load in the instrument files.

```
load_directory = gwent.__path__[0] + '/LoadFiles'
```

5.1 Initialize different instruments

To compare BHB strains and assess their detectability, we load in a few example detectors. For more information about loading instruments, see the tutorial on detectors.

5.1.1 NANOGrav 11yr Characteristic Strain

Using real NANOGrav 11yr data put through `hasasia`

```
NANOGrav_filedirectory = load_directory + '/InstrumentFiles/NANOGrav/StrainFiles/'
NANOGrav_11yr_hasasia_file = NANOGrav_filedirectory + 'NANOGrav_11yr_S_eff.txt'
NANOGrav_11yr_hasasia = detector.PTA('NANOGrav 11yr',load_location=NANOGrav_11yr_
↪hasasia_file,I_type='E')
NANOGrav_11yr_hasasia.T_obs = 11.4*u.yr
```

5.1.2 LISA Proposal 1

Values taken from the ESA L3 proposal, Amaro-Seaone, et al., 2017 (https://arxiv.org/abs/1702.00786)

```
L = 2.5*u.Gm  #armlength in Gm
L = L.to('m')
LISA_T_obs = 4*u.yr

f_acc_break_low = .4*u.mHz.to('Hz')*u.Hz
f_acc_break_high = 8.*u.mHz.to('Hz')*u.Hz
f_IMS_break = 2.*u.mHz.to('Hz')*u.Hz
A_acc = 3e-15*u.m/u.s/u.s
A_IMS = 10e-12*u.m

Background = False

LISA_prop1 = detector.SpaceBased('LISA',
                                 LISA_T_obs,L,A_acc,f_acc_break_low,
                                 f_acc_break_high,A_IMS,f_IMS_break,
                                 Background=Background)
```

5.1.3 aLIGO

```
Ground_T_obs = 4*u.yr
#aLIGO
aLIGO_filedirectory = load_directory + '/InstrumentFiles/aLIGO/'
aLIGO_1_filename = 'aLIGODesign.txt'

aLIGO_1_filelocation = aLIGO_filedirectory + aLIGO_1_filename

aLIGO_1 = detector.GroundBased('aLIGO 1',Ground_T_obs,load_location=aLIGO_1_
↪filelocation,I_type='A')
```

5.2 Generating Binary Black Holes with `gwent` in the Frequency Domain

We start with BHB parameters that exemplify the range of IMRPhenomD's waveforms from Khan, et al. 2016 https://arxiv.org/abs/1508.07253 and Husa, et al. 2016 https://arxiv.org/abs/1508.07250

```
M = [1e6,65.0,1e10]
q = [1.0,18.0,1.0]
x1 = [0.5,0.0,-0.95]
x2 = [0.3,0.0,-0.95]
z = [3.0,0.093,20.0]
```

Uses the first parameter values that lie in the `LISA_prop1` detector band with the precessing phenomenological `lalsuite` waveform IMRPhenomPv3.

```
lalsuite_kwargs = {"S1x": 0.5, "S1y": 0., "S1z": x1[0],
                   "S2x": -0.2, "S2y": 0.5, "S2z": x2[0],
                   "inclination":np.pi/2}
source_1 = binary.BBHFrequencyDomain(M[0],q[0],z[0],approximant='IMRPhenomPv3',
↪lalsuite_kwargs=lalsuite_kwargs)
```

Uses the second parameter values that lie in the `aLIGO` detector band.

```
source_2 = binary.BBHFrequencyDomain(M[1],q[1],z[1],x1[1],x2[1])
```

Uses the third parameter values that lie in the `NANOGrav_11yr_hasasia` detector band.

```
source_3 = binary.BBHFrequencyDomain(M[2],q[2],z[2],x1[2],x2[2])
```

5.3 How to Get Information about BHB

5.3.1 Find out source 1's frequency given some time from merger.

```
print("Source frequency 10 years prior to merger in Observer frame: ",
      binary.Get_Source_Freq(source_1,10*u.yr,in_frame='observer',out_frame='source'))
print("Source frequency 10 years prior to merger in Source frame: ",
      binary.Get_Source_Freq(source_1,10*u.yr,in_frame='source',out_frame='source'))
print("Observed frequency 10 years prior to merger in Observer frame: ",
```

(continues on next page)

(continued from previous page)

```
      binary.Get_Source_Freq(source_1,10*u.yr,in_frame='observer',out_frame='observer
↪'))
print("Observed frequency 10 years prior to merger in Source frame: ",
      binary.Get_Source_Freq(source_1,10*u.yr,in_frame='source',out_frame='observer'))
```

```
Source frequency 10 years prior to merger in Observer frame:  4.9371229709723884e-05␣
↪1 / s
Source frequency 10 years prior to merger in Source frame:  2.9356308823618684e-05 1 /
↪ s
Observed frequency 10 years prior to merger in Observer frame:  1.2342807427430971e-
↪05 1 / s
Observed frequency 10 years prior to merger in Source frame:  7.339077205904671e-06 1␣
↪/ s
```

5.3.2 Find out source 2's time to merger from a given frequency.

```
print("Source time from merger for BHB with GW frequency of 1/minute (~17mHz) in the␣
↪Observer frame: ",
      binary.Get_Time_From_Merger(source_2,freq=1/u.minute,in_frame='observer',out_
↪frame='source').to('yr'))
print("Source time from merger for BHB with GW frequency of 1/minute (~17mHz) in the␣
↪Source frame: ",
      binary.Get_Time_From_Merger(source_2,freq=1/u.minute,in_frame='source',out_
↪frame='source').to('yr'))
print("Observed ime from merger for BHB with GW frequency of 1/minute (~17mHz) in the␣
↪Observer frame: ",
      binary.Get_Time_From_Merger(source_2,freq=1/u.minute,in_frame='observer',out_
↪frame='observer').to('yr'))
print("Observed time from merger for BHB with GW frequency of 1/minute (~17mHz) in␣
↪the Source frame: ",
      binary.Get_Time_From_Merger(source_2,freq=1/u.minute,in_frame='source',out_
↪frame='observer').to('yr'))
```

```
Source time from merger for BHB with GW frequency of 1/minute (~17mHz) in the␣
↪Observer frame:  17.032270309184415 yr
Source time from merger for BHB with GW frequency of 1/minute (~17mHz) in the Source␣
↪frame:  21.590347849144273 yr
Observed ime from merger for BHB with GW frequency of 1/minute (~17mHz) in the␣
↪Observer frame:  18.61627144793857 yr
Observed time from merger for BHB with GW frequency of 1/minute (~17mHz) in the␣
↪Source frame:  23.59825019911469 yr
```

5.3.3 Find out source 3's observed frequency given some evolved time.

And whether the source is monochromatic or chirping for the evolved time in the observer frame.

```
#First we have to give the source some initial frequency
source_3.f_gw = 8*u.nHz

binary.Check_Freq_Evol(source_3,T_evol=5*u.yr,T_evol_frame='observer')
print("Observed frequency after 5 years of evolution in Observer frame: ",
      source_3.f_T_obs)
```

(continues on next page)

(continued from previous page)

```
print("Does the source change a resolvable amount after evolving for 5 years in the␣
↪Observer frame?: ",
      source_3.ismono)
print("\n")
binary.Check_Freq_Evol(source_3,T_evol=5*u.yr,T_evol_frame='source')
print("Observed frequency after 5 years of evolution in Source frame: ",
      source_3.f_T_obs)
print("Does the source change a resolvable amount after evolving for 5 years in the␣
↪Source frame?: ",
      source_3.ismono)
```

```
Observed frequency after 5 years of evolution in Observer frame:  1.7955629558729957e-
↪08 1 / s
Does the source change a resolvable amount after evolving for 5 years in the Observer␣
↪frame?:  True


Observed frequency after 5 years of evolution in Source frame:  5.732821260078733e-09␣
↪1 / s
Does the source change a resolvable amount after evolving for 5 years in the Source␣
↪frame?:  False
```

We can set the instrument that "observes" the source. If you orginally assign the source an instrument (which we show in a bit), the initial frequency (`f_gw`) is set to the instrument's most sensitive frequency

```
source_3.instrument = NANOGrav_11yr_hasasia
binary.Check_Freq_Evol(source_3)
print(f"Observed frequency after {np.max(source_3.instrument.T_obs)} years of␣
↪evolution in Observer frame: ",
      source_3.f_T_obs)
```

```
Observed frequency after 11.4 yr years of evolution in Observer frame:  1.
↪3181810661218933e-08 1 / s
```

5.4 Plots of Example GW Band

Displays only generated detectors: WN only PTAs, ESA L3 proposal LISA, aLIGO, and Einstein Telescope.

5.4.1 Chirping Sources

Displays two sources' waveform throughout its observing run (from left to right: `NANOGrav_11yr_hasasia`,`LISA_prop1`,`ET`). Since the default frame for each source is the observer frame, we get the observed frequency of each source `T_obs` before merger.

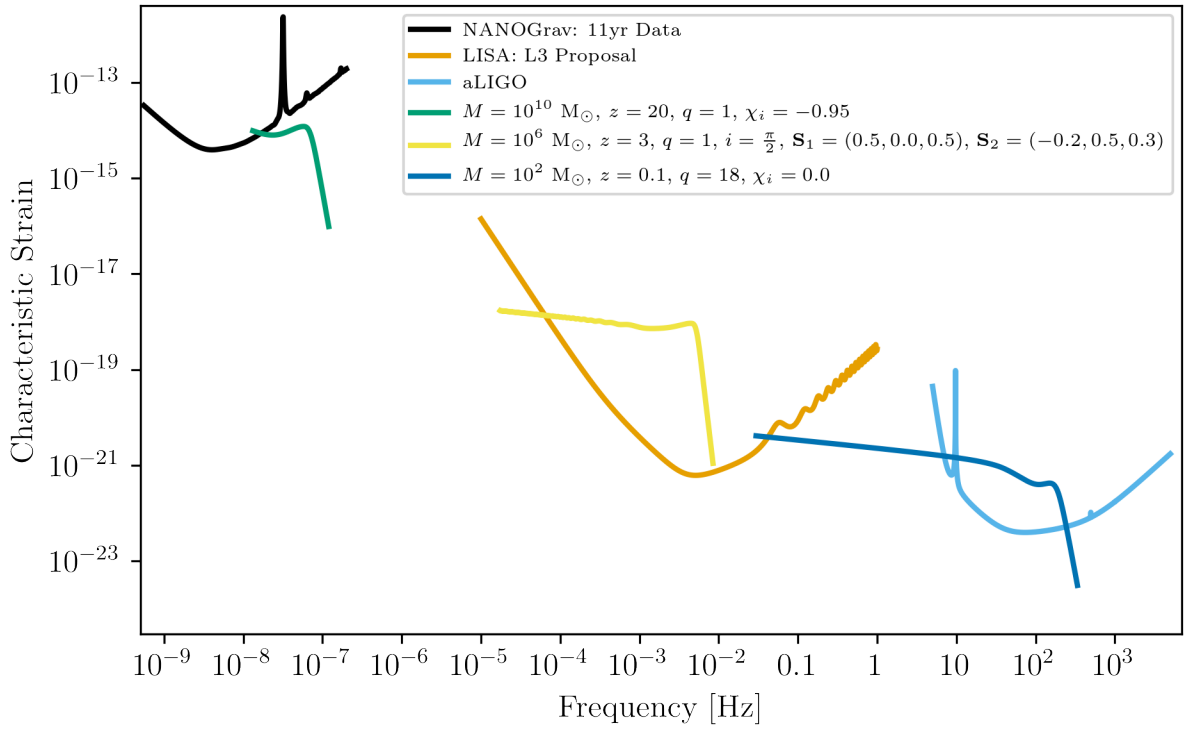
```
source_1_t_T_obs_f = binary.Get_Source_Freq(source_1,LISA_prop1.T_obs,in_frame=
↪"observer",out_frame="observer")
source_1_idx = np.abs(source_1.f-source_1_t_T_obs_f).argmin()

source_2_t_T_obs_f = binary.Get_Source_Freq(source_2,aLIGO_1.T_obs,in_frame="observer
↪",out_frame="observer")
source_2_idx = np.abs(source_2.f-source_2_t_T_obs_f).argmin()
```

(continues on next page)

(continued from previous page)

```
source_3_t_T_obs_f = binary.Get_Source_Freq(source_3,NANOGrav_11yr_hasasia.T_obs,in_
↪frame="observer",out_frame="observer")
source_3_idx = np.abs(source_3.f-source_3_t_T_obs_f).argmin()
```

```
plt.figure(figsize=get_fig_size())

plt.loglog(NANOGrav_11yr_hasasia.fT,NANOGrav_11yr_hasasia.h_n_f,label='NANOGrav: 11yr␣
↪Data')
plt.loglog(LISA_prop1.fT,LISA_prop1.h_n_f,label='LISA: L3 Proposal')
plt.loglog(aLIGO_1.fT,aLIGO_1.h_n_f,label='aLIGO')

plt.loglog(source_3.f[source_3_idx:],binary.Get_Char_Strain(source_3)[source_3_idx:],
          label=r'$M = 10^{%.0f}$ $\mathrm{M}_{\odot}$, $z = %.0f$, $q = %.0f$, $\chi_
↪{i} = %.2f$'
              %(np.log10(source_3.M.value),source_3.z,source_3.q,source_3.chi1))
plt.loglog(source_1.f[source_1_idx:],binary.Get_Char_Strain(source_1)[source_1_idx:],
          label=r'$M = 10^{%.0f}$ $\mathrm{M}_{\odot}$, $z = %.0f$, $q = %.0f$, $i =␣
↪\frac{\pi}{2}$,'\
           %(np.log10(source_1.M.value),source_1.z,source_1.q) +\
           r' $\textbf{S}_{1} = (%.1f,%.1f,%.1f)$, $\textbf{S}_{2} = (%.1f,%.1f,%.1f)$
↪'\
           %(source_1.lalsuite_kwargs['S1x'],source_1.lalsuite_kwargs['S1y'],source_1.
↪lalsuite_kwargs['S1z'],
             source_1.lalsuite_kwargs['S2x'],source_1.lalsuite_kwargs['S2y'],source_1.
↪lalsuite_kwargs['S2z']))
plt.loglog(source_2.f[source_2_idx:],binary.Get_Char_Strain(source_2)[source_2_idx:],
          label=r'$M = 10^{%.0f}$ $\mathrm{M}_{\odot}$, $z = %.1f$, $q = %.0f$, $\chi_
↪{i} = %.1f$'
              %(np.log10(source_2.M.value),source_2.z,source_2.q,source_2.chi1))

xlabel_min = -10
xlabel_mplt = 5
xlabels = np.arange(xlabel_min,xlabel_mplt+1)
xlabels = xlabels[1::]

print_xlabels = []
for x in xlabels:
    if abs(x) > 1:
        print_xlabels.append(r'$10^{%i}$' %x)
    elif x == -1:
        print_xlabels.append(r'$%.1f$' %10.**x)
    else:
        print_xlabels.append(r'$%.0f$' %10.**x)
plt.xticks(10.**xlabels,print_xlabels)

plt.xlim([5e-10, 7e3])
plt.ylim([3e-25, 4e-12])

plt.xlabel('Frequency [Hz]')
plt.ylabel('Characteristic Strain')
plt.legend(fontsize=7)
plt.show()
```



5.4.2 Monochromatic Sources

Displays a comparison between two monochromatic strain sources, one equal mass, the other at a mass ratio of 18. The initial frequency is set by the NANOGrav 11yr at the detector's most sensitive frequency. The NANOGrav 11yr data in this plot corresponds to a source strain (h_0) with SNR of one; note that this is not characteristic strain.

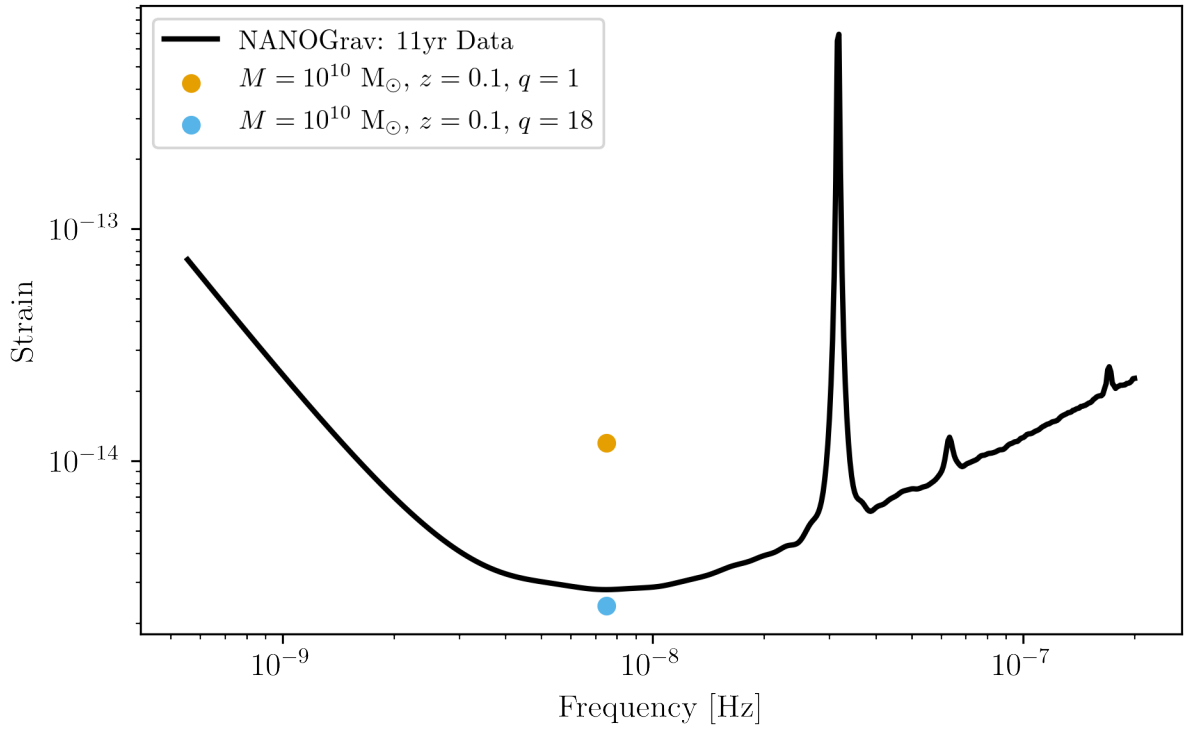
```
source_4 = binary.BBHFrequencyDomain(1e10,1.0,0.1,0.0,0.0,instrument=NANOGrav_11yr_
↪hasasia)
source_5 = binary.BBHFrequencyDomain(1e10,18.0,0.1,0.0,0.0,instrument=NANOGrav_11yr_
↪hasasia)
```

```
plt.figure(figsize=get_fig_size())

plt.loglog(NANOGrav_11yr_hasasia.fT,
           np.sqrt(NANOGrav_11yr_hasasia.S_n_f/np.max(np.unique(NANOGrav_11yr_hasasia.
↪T_obs.to('s').value))),
           label=r'NANOGrav: 11yr Data')
plt.scatter(source_4.f_gw,
            source_4.h_gw,
            color='C1',
            label=r'$M = 10^{%.0f}$ $\mathrm{M}_{\odot}$, $z = %.1f$, $q = %.0f$'
              %(np.log10(source_4.M.value),source_4.z,source_4.q))
plt.scatter(source_5.f_gw,
            source_5.h_gw,
            color='C2',
            label=r'$M = 10^{%.0f}$ $\mathrm{M}_{\odot}$, $z = %.1f$, $q = %.0f$'
              %(np.log10(source_5.M.value),source_5.z,source_5.q))

plt.xlabel('Frequency [Hz]')
```

(continues on next page)

(continued from previous page)

```
plt.ylabel('Strain')
plt.legend(loc='upper left')
plt.show()
```



5.5 Calculating the SNR

For the two sources displayed in the plot above, we will calculate the SNRs for monochromatic and chirping versions.

5.5.1 Source 4: Monochromatic Case

Response in LISA data First we set the source frequency. If you assign an instrument and not a frequency, `gwent` does this step internally and sets `f_gw` to the instruments optimal frequency (like we have done above too).

```
snr.Calc_Mono_SNR(source_4,NANOGrav_11yr_hasasia).to('')
```

2.7270486

One can also change the inclination of the source for calculating the monochromatic SNR.

```
snr.Calc_Mono_SNR(source_4,NANOGrav_11yr_hasasia,inc=np.pi/2).to('')
```

1.5244665

5.5.2 Source 2: Chirping Case

Response in aLIGO data

To set the start frequency of integration, you need to set the amount of time the instrument observes the source. This is done automatically for the given instrument.

```
snr.Calc_Chirp_SNR(source_2,aLIGO_1)
```

```
19.29898065079436
```

5.5.3 Source 1: Chirping Case

Response in LISA data

```
snr.Calc_Chirp_SNR(source_1,LISA_prop1)
```

```
1038.9835482056897
```

Other ways this can be done is by setting the instrument's observation time or by using `binary.Check_Freq_Evol` and setting the optional `T_evol` parameter to the new observation time.

You can see in tis case, we have to drastically shorten the observed time to visibly change the SNR because the source waveform is so close to merger at the edge of LISA's frequency band.

```
binary.Check_Freq_Evol(source_1,T_evol=1*u.hr)
snr.Calc_Chirp_SNR(source_1,LISA_prop1)
```

```
1035.346096569876
```

5.6 Generate Frequency Data from Given Time Domain

Uses waveforms that are the difference between Effective One Body waveforms subtracted from Numerical Relativity waveforms for different harmonics.

This method and use is fairly untested, so proceed with caution and feel free to help out!

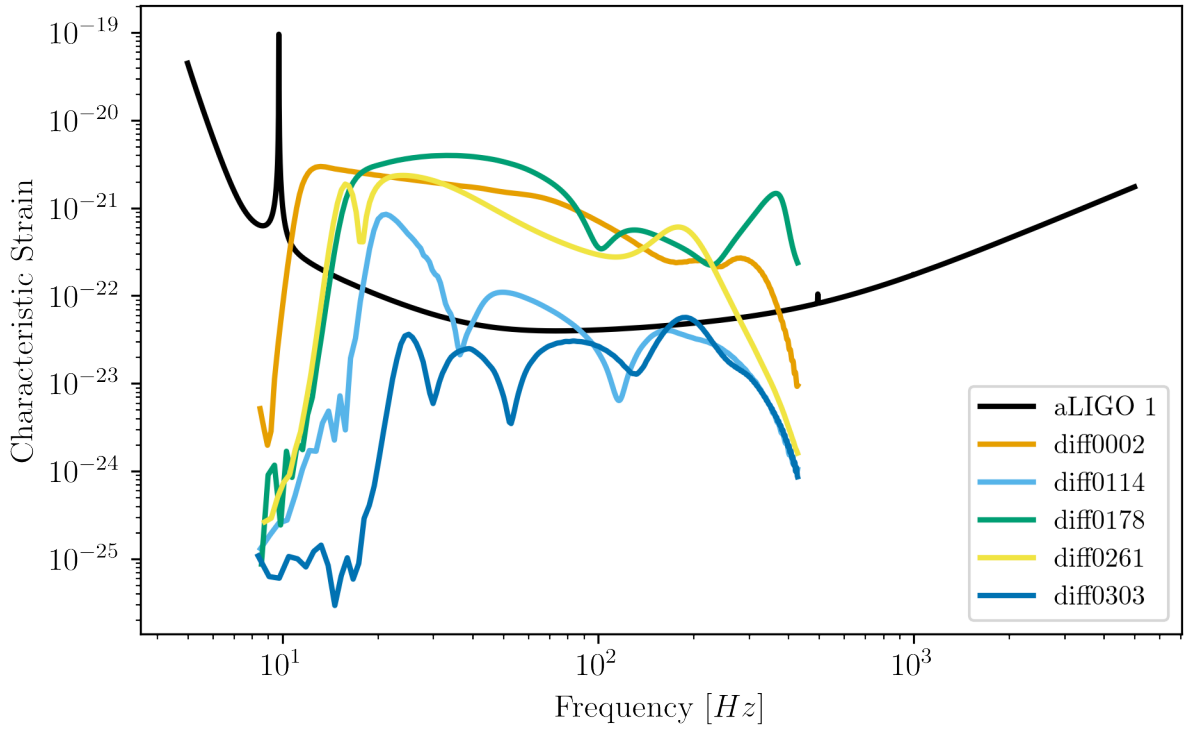
```
EOBdiff_filedirectory = load_directory + '/DiffStrain/EOBdiff/'
diff0002 = binary.BBHTimeDomain(M[1],q[0],z[1],load_location=EOBdiff_filedirectory+
↪'diff0002.dat')
diff0114 = binary.BBHTimeDomain(M[1],q[0],z[1],load_location=EOBdiff_filedirectory+
↪'diff0114.dat')
diff0178 = binary.BBHTimeDomain(M[1],q[0],z[1],load_location=EOBdiff_filedirectory+
↪'diff0178.dat')
diff0261 = binary.BBHTimeDomain(M[1],q[0],z[1],load_location=EOBdiff_filedirectory+
↪'diff0261.dat')
diff0303 = binary.BBHTimeDomain(M[1],q[0],z[1],load_location=EOBdiff_filedirectory+
↪'diff0303.dat')
```

```
fig,ax = plt.subplots()
plt.loglog(aLIGO_1.fT,aLIGO_1.h_n_f,label = aLIGO_1.name)
plt.loglog(diff0002.f,binary.Get_Char_Strain(diff0002),label = 'diff0002')
plt.loglog(diff0114.f,binary.Get_Char_Strain(diff0114),label = 'diff0114')
plt.loglog(diff0178.f,binary.Get_Char_Strain(diff0178),label = 'diff0178')
plt.loglog(diff0261.f,binary.Get_Char_Strain(diff0261),label = 'diff0261')
plt.loglog(diff0303.f,binary.Get_Char_Strain(diff0303),label = 'diff0303')
```

(continues on next page)

(continued from previous page)

```
plt.xlabel(r'Frequency $[Hz]$')
plt.ylabel('Characteristic Strain')
plt.legend()
plt.show()
```



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

CHAPTER 6

Using `gwent` to Calculate Signal-to-Noise Ratios

Here we present a tutorial on how to use `gwent` to calculate SNRs for the instrument models currently implemented (LISA, PTAs, aLIGO, and Einstein Telescope) with the signal being an array of coalescing Binary Black Holes.

First, we import important modules.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.constants import golden_ratio

import astropy.constants as const
import time
import astropy.units as u

import gwent
import gwent.binary as binary
import gwent.detector as detector
import gwent.snr as snr
from gwent.snrplot import Plot_SNR

#Turn off warnings for tutorial
import warnings
warnings.filterwarnings('ignore')
```

Setting matplotlib preferences and adding a pretty plot function for fig sizes

```
def get_fig_size(width=7,scale=2.0):
    #width = 3.36 # 242 pt
    base_size = np.array([1, 1/scale/golden_ratio])
    fig_size = width * base_size
    return(fig_size)
mpl.rcParams['figure.dpi'] = 300
#mpl.rcParams['figure.figsize'] = get_fig_size()
mpl.rcParams['text.usetex'] = True
```

(continues on next page)

(continued from previous page)

```
mpl.rc('font',**{'family':'serif','serif':['Times New Roman']})
mpl.rcParams['lines.linewidth'] = 1.3
mpl.rcParams['axes.labelsize'] = 12
mpl.rcParams['xtick.labelsize'] = 10
mpl.rcParams['ytick.labelsize'] = 10
mpl.rcParams['legend.fontsize'] = 8
```

We need to get the file directories to load in the instrument files.

```
load_directory = gwent.__path__[0] + '/LoadFiles/InstrumentFiles/'
```

6.1 Fiducial Source Creation

To run `snr.Get_SNR_Matrix`, you need to instantiate a source. Since we need to reinitialize a few times, we just put it here as a function.

This is an example for reasonable mass ranges for the particular detector mass regime and the variable ranges limited by the waveform calibration region.

The source parameters must be set (ie. M,q,z,chi1,chi2), but one also needs to set the minima and maxima of the selected SNR axes variables. This takes the form of `source_param = [fiducial_value,minimum, maximum]`

```
def Initialize_Source(instrument,approximant='pyPhenomD',lalsuite_kwargs={}):
    """Initializes a source binary based on the instrument type and returns the source

    Parameters
    ----------
    instrument : object
        Instance of a gravitational wave detector class
    approximant : str, optional
        the approximant used to calculate the frequency domain waveform of the source.
        Can either be the python implementation of IMRPhenomD ('pyPhenomD', the␣
↪default) given below,
        or a waveform modelled in LIGO's lalsuite's lalsimulation package.
    lalsuite_kwargs: dict, optional
        More specific user-defined kwargs for the different lalsuite waveforms
    """

    #q = m2/m1 reduced mass
    q = 1.0
    q_min = 1.0
    q_max = 18.0
    q_list = [q,q_min,q_max]

    #Chi = S_i*L/m_i**2, spins of each mass i
    chi1 = 0.0 #spin of m1
    chi2 = 0.0 #spin of m2
    chi_min = -0.85 #Limits of PhenomD for unaligned spins
    chi_max = 0.85
    chi1_list = [chi1,chi_min,chi_max]
    chi2_list = [chi2,chi_min,chi_max]

    #Redshift
```

(continues on next page)

(continued from previous page)

```
    z_min = 1e-2
    z_max = 1e3

    if isinstance(instrument,detector.GroundBased):
        #Total source mass
        M_ground_source = [10.,1.,1e4]
        #Redshift
        z_ground_source = [0.1,z_min,z_max]

        source = binary.BBHFrequencyDomain(M_ground_source,
                                           q_list,
                                           z_ground_source,
                                           chi1_list,
                                           chi2_list,
                                           approximant=approximant,
                                           lalsuite_kwargs=lalsuite_kwargs)
    elif isinstance(instrument,detector.SpaceBased):
        M_space_source = [1e6,1.,1e10]
        z_space_source = [1.0,z_min,z_max]
        source = binary.BBHFrequencyDomain(M_space_source,
                                           q_list,
                                           z_space_source,
                                           chi1_list,
                                           chi2_list,
                                           approximant=approximant,
                                           lalsuite_kwargs=lalsuite_kwargs)
    elif isinstance(instrument,detector.PTA):
        M_pta_source = [1e9,1e8,1e11]
        z_pta_source = [0.1,z_min,z_max]
        source = binary.BBHFrequencyDomain(M_pta_source,
                                           q_list,
                                           z_pta_source,
                                           chi1_list,
                                           chi2_list,
                                           approximant=approximant,
                                           lalsuite_kwargs=lalsuite_kwargs)
    return source
```

6.2 Create SNR Matrices and Samples for a Few Examples

The variables for either axis in the SNR calculation can be:

- **GLOBAL:**
  - `T_obs` - Detector Observation Time
- **SOURCE:**
  - `M` - Mass (Solar Units)
  - `q` - Mass Ratio
  - `chi1` - Dimensionless Spin of Black Hole 1
  - `chi2` - Dimensionless Spin of Black Hole 2
  - `z` - Redshift

- **GroundBased ONLY:**
  - Any single valued variable in list of params given by: `instrument_GroundBased.Get_Noise_Dict()`
  - To make variable in SNR, declare the main variable, then the subparameter variable as a string e.g. `var_x = Infrastructure Length`, the case matters.
- **SpaceBased ONLY:**
  - `L` - Detector Armlength
  - `A_acc` - Detector Acceleration Noise
  - `A_IFO` - Detector Optical Metrology Noise
  - `f_acc_break_low` - The Low Acceleration Noise Break Frequency
  - `f_acc_break_high` - The High Acceleration Noise Break Frequency
  - `f_IFO_break` - The Optical Metrology Noise Break Frequency
- **PTA ONLY:**
  - `n_p` - Number of Pulsars
  - `sigma` - Root-Mean-Squared Timing Error
  - `cadence` - Observation Cadence

6.3 Instrument Creation Examples

For each instrument one wants to investigate, you have to assign the fiducial noise and detector values. We do the same reinitialization game here as the source, so each of these are functions.

These examples only assign ranges of calculation for quick variable calculations, but one only needs to set the minima and maxima if they wish to use other selected SNR axes variables.

If loading a detector, the file should be frequency in the first column and either strain, effective strain noise spectral density, or amplitude spectral density in the second column.

The strain tutorial goes into more detail on initializing detectors, so if you get lost, look there!

6.3.1 Ground Based Detectors

```
def Initialize_aLIGO():
    #Observing time in years
    T_obs_ground_list = [4*u.yr,1*u.yr,10*u.yr]
    #aLIGO
    noise_dict_aLIGO = {'Infrastructure':
                   {'Length':[3995,2250,4160]},
                   'Laser':
                   {'Power':[125,10,1e3]},
                   'Seismic':
                   {'Gamma':[0.8,0.1,1.0]}}
    aLIGO = detector.GroundBased('aLIGO',T_obs_ground_list,noise_dict=noise_dict_
↪aLIGO)

    return aLIGO
```

6.3.2 Space Based Detectors

```
def Initialize_LISA():
    #Values taken from the ESA L3 proposal, Amaro-Seaone, et al., 2017 (https://arxiv.
↪org/abs/1702.00786)
    T_obs_space_list = [4*u.yr,1*u.yr,10*u.yr]

    #armlength in meters
    L = 2.5e9*u.m
    L_min = 1.0e7*u.m
    L_max = 1.0e11*u.m
    L_list = [L,L_min,L_max]

    #Acceleration Noise Amplitude
    A_acc = 3e-15*u.m/u.s/u.s
    A_acc_min = 1e-16*u.m/u.s/u.s
    A_acc_max = 1e-14*u.m/u.s/u.s
    A_acc_list = [A_acc,A_acc_min,A_acc_max]

    #The Low Acceleration Noise Break Frequency
    f_acc_break_low = .4*u.mHz.to('Hz')*u.Hz
    f_acc_break_low_min = .1*u.mHz.to('Hz')*u.Hz
    f_acc_break_low_max = 1.0*u.mHz.to('Hz')*u.Hz
    f_acc_break_low_list = [f_acc_break_low,f_acc_break_low_min,f_acc_break_low_max]

    #The High Acceleration Noise Break Frequency
    f_acc_break_high = 8.*u.mHz.to('Hz')*u.Hz
    f_acc_break_high_min = 1.*u.mHz.to('Hz')*u.Hz
    f_acc_break_high_max = 10.*u.mHz.to('Hz')*u.Hz
    f_acc_break_high_list = [f_acc_break_high,f_acc_break_high_min,f_acc_break_high_
↪max]

    #The Optical Metrology Noise Break Frequency
    f_IFO_break = 2.*u.mHz.to('Hz')*u.Hz
    f_IFO_break_min = 1.*u.mHz.to('Hz')*u.Hz
    f_IFO_break_max = 10.*u.mHz.to('Hz')*u.Hz
    f_IFO_break_list = [f_IFO_break,f_IFO_break_min,f_IFO_break_max]

    #Detector Optical Metrology Noise
    A_IFO = 10e-12*u.m
    A_IFO_min = 1.0e-13*u.m
    A_IFO_max = 1.0e-10*u.m
    A_IFO_list = [A_IFO,A_IFO_min,A_IFO_max]

    #Unresolved Galactic WD Background
    Background = False

    #Numerical Transfer Function
    T_type = 'N'

    LISA_prop1 = detector.SpaceBased('LISA_prop1',
                                     T_obs_space_list,L_list,A_acc_list,
                                     f_acc_break_low_list,f_acc_break_high_list,
                                     A_IFO_list,f_IFO_break_list,
                                     Background=Background,T_type=T_type)
    return LISA_prop1
```

6.3.3 PTA Detectors

```
def Initialize_NANOGrav():
    #NANOGrav calculation using 11.5yr parameters https://arxiv.org/abs/1801.01837
    #Observing time in years
    T_obs_ptas_list = [11.42*u.yr,5*u.yr,30*u.yr]
    #rms timing residuals in seconds
    sigma = 100*u.ns.to('s')*u.s
    sigma_min = 100*u.ns.to('s')*u.s
    sigma_max = 500*u.ns.to('s')*u.s
    sigma_list = [sigma,sigma_min,sigma_max]
    #Number of pulsars
    n_p = 34
    n_p_min = 18
    n_p_max = 200
    n_p_list = [n_p,n_p_min,n_p_max]
    #Avg observation cadence of 1 every 2 weeks in num/year
    cadence = 1/(2*u.wk.to('yr')*u.yr)
    cadence_min = 2/u.yr
    cadence_max = 1/(u.wk.to('yr')*u.yr)
    cadence_list = [cadence,cadence_min,cadence_max]

    #NANOGrav 11.4 yr WN only
    NANOGrav_WN = detector.PTA('NANOGrav_WN',n_p_list,T_obs=T_obs_ptas_list,
↪sigma=sigma_list,cadence=cadence_list)
    return NANOGrav_WN
```

6.4 SNR Calculations

To actually sample the parameter space, one needs to declare x and y variables that correspond to the variables inside the relavant instrument and/or model for the SNR Calculation.

You will also need to assign Sample Rates for each, this will directly determine how long a calculation will take. I have kept all curves under 100 for paper figures, so I would recommend nothing over that, but I won't tell you what to do!

```
#Number of SNRMatrix rows
sampleRate_y = 100
#Number of SNRMatrix columns
sampleRate_x = 100
```

We now use `Get_SNR_Matrix` with the variables given and the data range to sample the space either logrithmically or linearly based on the selection of variables. It computes the SNR for each value, then returns the variable ranges used to calculate the SNR for each matrix, then returns the SNRs with size of the `sampleRate_x`X`sampleRate_y`

6.4.1 aLIGO

Varying Source Parameters

Here we calculate the SNR for three source parameters `chi1`,`q`, and `z` using `Get_SNR_Matrix`. For ease of the example, we just do them all at once.

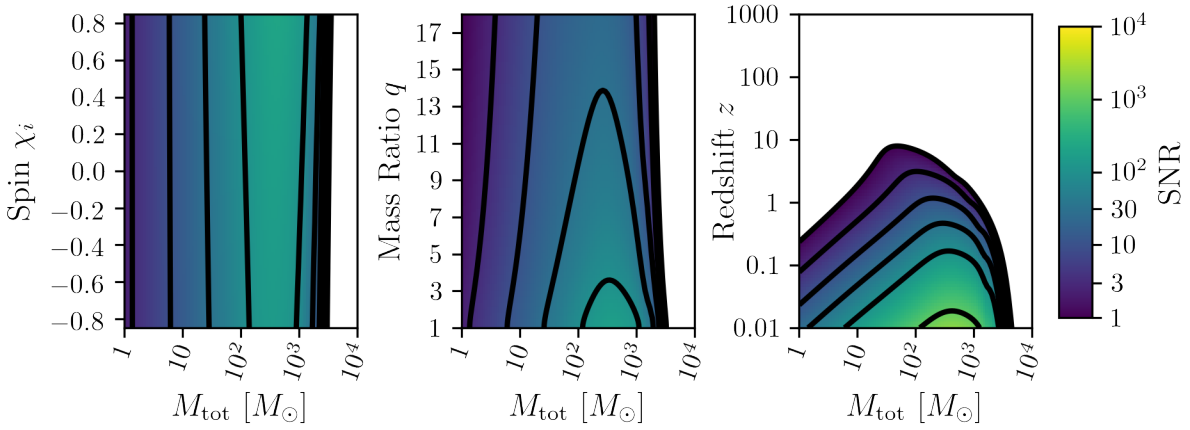
```
#Variable on y-axis
var_ys = ['chi1','q','z']
#Variable on x-axis
var_x = 'M'
instrument = Initialize_aLIGO()
sample_x_array = []
sample_y_array = []
SNR_array = []
for var_y in var_ys:
    source = Initialize_Source(instrument)
    start = time.time()
    [sample_x,sample_y,SNRMatrix] = snr.Get_SNR_Matrix(source,instrument,
                                                       var_x,sampleRate_x,
                                                       var_y,sampleRate_y)
    end = time.time()
    sample_x_array.append(sample_x)
    sample_y_array.append(sample_y)
    SNR_array.append(SNRMatrix)

    print('Model: ',instrument.name + '_' + var_x + '_vs_' + var_y,',',' done. t = :
↪',end-start)
```

```
Model:  aLIGO_M_vs_chi1 ,  done. t = :  31.16754937171936
Model:  aLIGO_M_vs_q ,  done. t = :  31.795299530029297
Model:  aLIGO_M_vs_z ,  done. t = :  23.489653825759888
```

Plotting SNRs

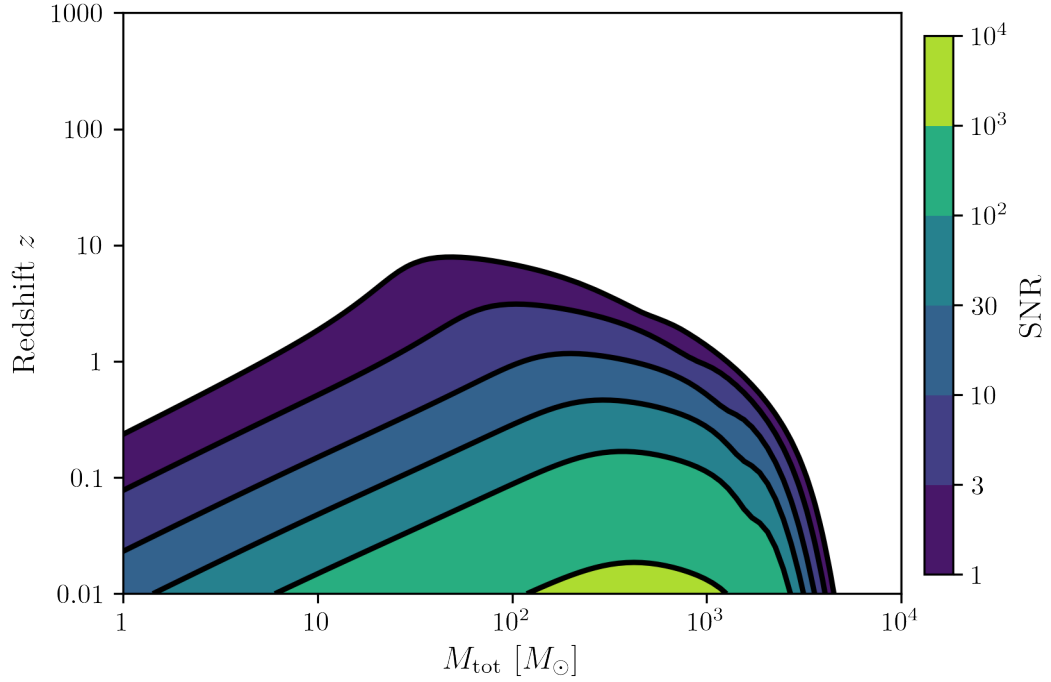
This is just an example of plotting the above SNRs using the `Plot_SNR` function. The function can take a *ton* of parameters, but for simple plots most of them are unneccessary.

```
fig, axes = plt.subplots(1,3,figsize=get_fig_size())
loglevelMax=4.0
hspace = .1
wspace = .45
for i,ax in enumerate(axes):
    if i == (len(axes)-1):
        Plot_SNR('M',sample_x_array[i],var_ys[i],
                 sample_y_array[i],SNR_array[i],
                 fig=fig,ax=ax,display=True,display_cbar=True,
                 logLevels_max=loglevelMax,
                 hspace=hspace,wspace=wspace,
                 xticklabels_kwargs={'rotation':70,'y':0.02},
                 ylabels_kwargs={'labelpad':-5})
    else:
        Plot_SNR('M',sample_x_array[i],var_ys[i],
                 sample_y_array[i],SNR_array[i],
                 fig=fig,ax=ax,display=False,display_cbar=False,
                 logLevels_max=loglevelMax,
                 hspace=hspace,wspace=wspace,xticklabels_kwargs={'rotation':70,'y':0.
↪02})
    i += 1
```



A simple example for just one figure.

```
Plot_SNR('M',sample_x_array[-1],'z',sample_y_array[-1],SNR_array[-1],smooth_
↪contours=False)
```



Varying Instrument Parameters

This is very similar to the previous example, but with varying the instrument parameters `Infrastructure Length`, `Seismic Gamma`, and `Laser Power`vs. `M`.

One thing to note is that we moved the instrument initialization inside the for loop this time since we don't want the parameters to stay at the max value from the previous run.

```
#Variable on y-axis
var_ys = ['Infrastructure Length','Seismic Gamma','Laser Power']
```

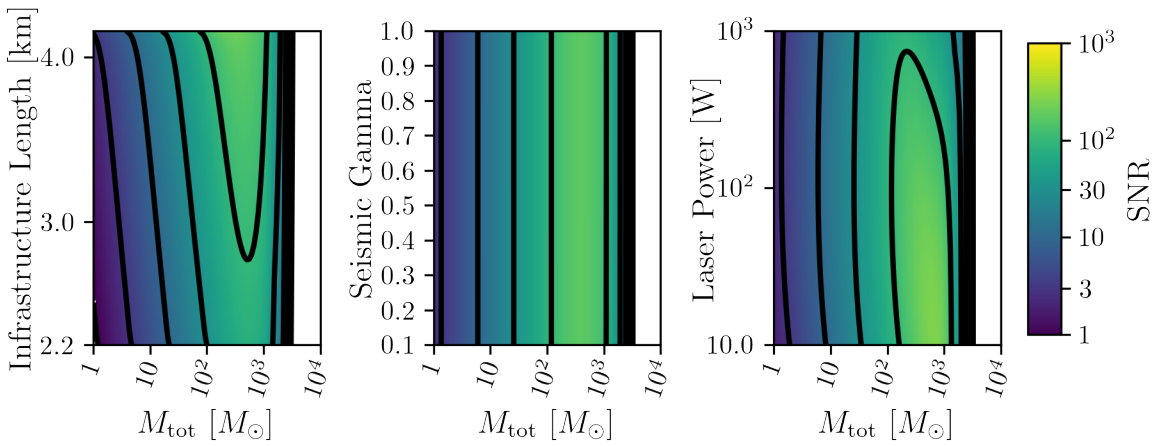
(continues on next page)

(continued from previous page)

```
#Variable on x-axis
var_x = 'M'
sample_x_array = []
sample_y_array = []
SNR_array = []
for var_y in var_ys:
    instrument = Initialize_aLIGO()
    source = Initialize_Source(instrument)
    start = time.time()
    [sample_x,sample_y,SNRMatrix] = snr.Get_SNR_Matrix(source,instrument,
                                                       var_x,sampleRate_x,
                                                       var_y,sampleRate_y)
    end = time.time()
    sample_x_array.append(sample_x)
    sample_y_array.append(sample_y)
    SNR_array.append(SNRMatrix)

    print('Model: ',instrument.name + '_' + var_x + '_vs_' + var_y,',',' done. t = :
↪',end-start)
```

```
Model:  aLIGO_M_vs_Infrastructure Length ,  done. t = :  24.8679461479187
Model:  aLIGO_M_vs_Seismic Gamma ,  done. t = :  23.985658407211304
Model:  aLIGO_M_vs_Laser Power ,  done. t = :  24.44096326828003
```

```
figsize = get_fig_size()
fig, axes = plt.subplots(1,3,figsize=figsize)
loglevelMax=3.0

wspace = .5

for i,ax in enumerate(axes):
    if i == (len(axes))-1:
        Plot_SNR('M',sample_x_array[i],var_ys[i],
                 sample_y_array[i],SNR_array[i],
                 fig=fig,ax=ax,display=True,display_cbar=True,
                 logLevels_max=loglevelMax,
                 hspace=hspace,wspace=wspace,
                 xticklabels_kwargs={'rotation':70,'y':0.02},ylabels_kwargs={'labelpad
↪':-5})
    else:
        Plot_SNR('M',sample_x_array[i],var_ys[i],
                 sample_y_array[i],SNR_array[i],
                 fig=fig,ax=ax,display=False,display_cbar=False,
                 logLevels_max=loglevelMax,
                 xticklabels_kwargs={'rotation':70,'y':0.02},
                 ylabels_kwargs={'labelpad':1})
```



6.4.2 LISA SNR

We now just for examples repeat the above few SNR calculations for LISA parameters.

```
#Variable on y-axis
var_ys = ['chi1','q','z']
#Variable on x-axis
var_x = 'M'
instrument = Initialize_LISA()
sample_x_array = []
sample_y_array = []
SNR_array = []
for var_y in var_ys:
    source = Initialize_Source(instrument)
    start = time.time()
    [sample_x,sample_y,SNRMatrix] = snr.Get_SNR_Matrix(source,instrument,
                                                       var_x,sampleRate_x,
                                                       var_y,sampleRate_y)
    end = time.time()
    sample_x_array.append(sample_x)
    sample_y_array.append(sample_y)
    SNR_array.append(SNRMatrix)

    print('Model: ',instrument.name + '_' + var_x + '_vs_' + var_y,',',' done. t = :
↪',end-start)
```

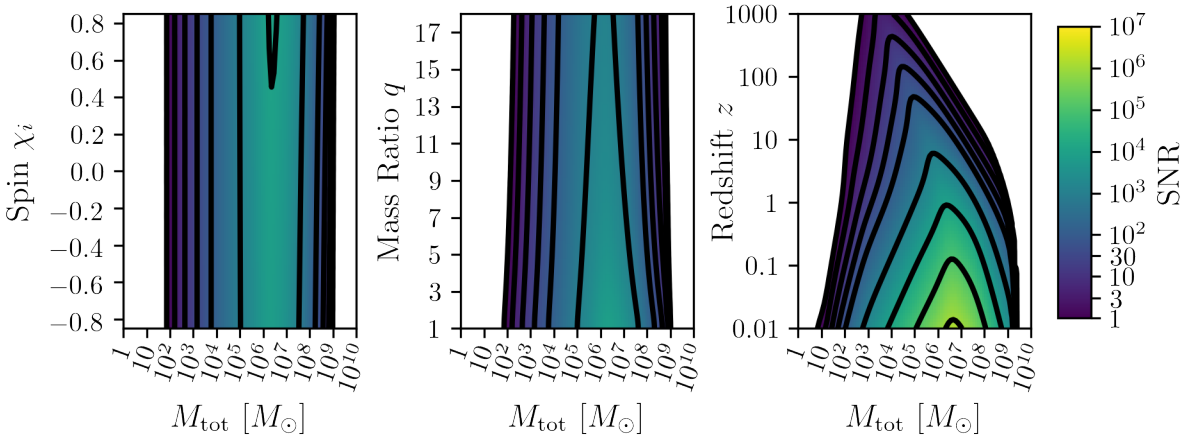
```
Model:  LISA_prop1_M_vs_chi1 ,  done. t = :  43.22341322898865
Model:  LISA_prop1_M_vs_q ,  done. t = :  43.869982957839966
Model:  LISA_prop1_M_vs_z ,  done. t = :  33.89306306838989
```

```
figsize = get_fig_size()
fig, axes = plt.subplots(1,3,figsize=figsize)
loglevelMax=7.0
hspace = .1
wspace = .45

for i,ax in enumerate(axes):
    if i == (len(axes))-1:
        Plot_SNR('M',sample_x_array[i],var_ys[i],
```

(continues on next page)

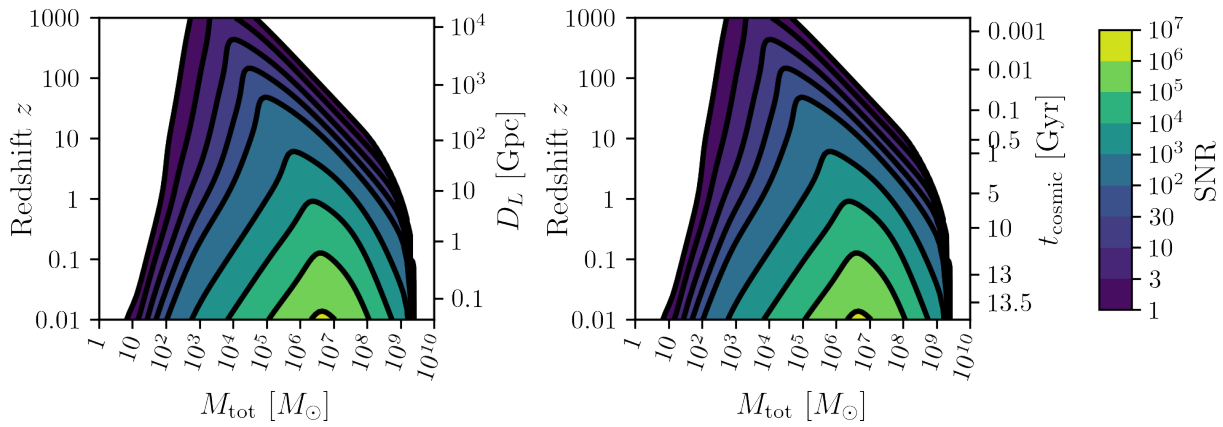
(continued from previous page)

```
                sample_y_array[i],SNR_array[i],
                fig=fig,ax=ax,display=True,display_cbar=True,
                logLevels_max=loglevelMax,
                hspace=hspace,wspace=wspace,
                xticklabels_kwargs={'rotation':70,'y':0.02},
                ylabels_kwargs={'labelpad':-5})
    else:
        Plot_SNR('M',sample_x_array[i],var_ys[i],
                sample_y_array[i],SNR_array[i],
                fig=fig,ax=ax,display=False,display_cbar=False,
                logLevels_max=loglevelMax,
                hspace=hspace,wspace=wspace,xticklabels_kwargs={'rotation':70,'y':0.
↪02})
```



Another included feature is the ability to add luminosity distance or lookback times onto the right hand axes of the redshift vs. total mass plots.

```
figsize = get_fig_size()
fig, axes = plt.subplots(1,2,figsize=figsize)
wspace = 0.6
Plot_SNR('M',sample_x_array[-1],'z',sample_y_array[-1],SNR_array[-1],fig=fig,
↪ax=axes[0],
         display=False,display_cbar=False,dl_axis=True,smooth_contours=False,
         xticklabels_kwargs={'rotation':70,'y':0.02},
         ylabels_kwargs={'labelpad':-3})
Plot_SNR('M',sample_x_array[-1],'z',sample_y_array[-1],SNR_array[-1],fig=fig,
↪ax=axes[1],
         lb_axis=True,wspace=wspace,smooth_contours=False,
         xticklabels_kwargs={'rotation':70,'y':0.02},
         ylabels_kwargs={'labelpad':-3})
```



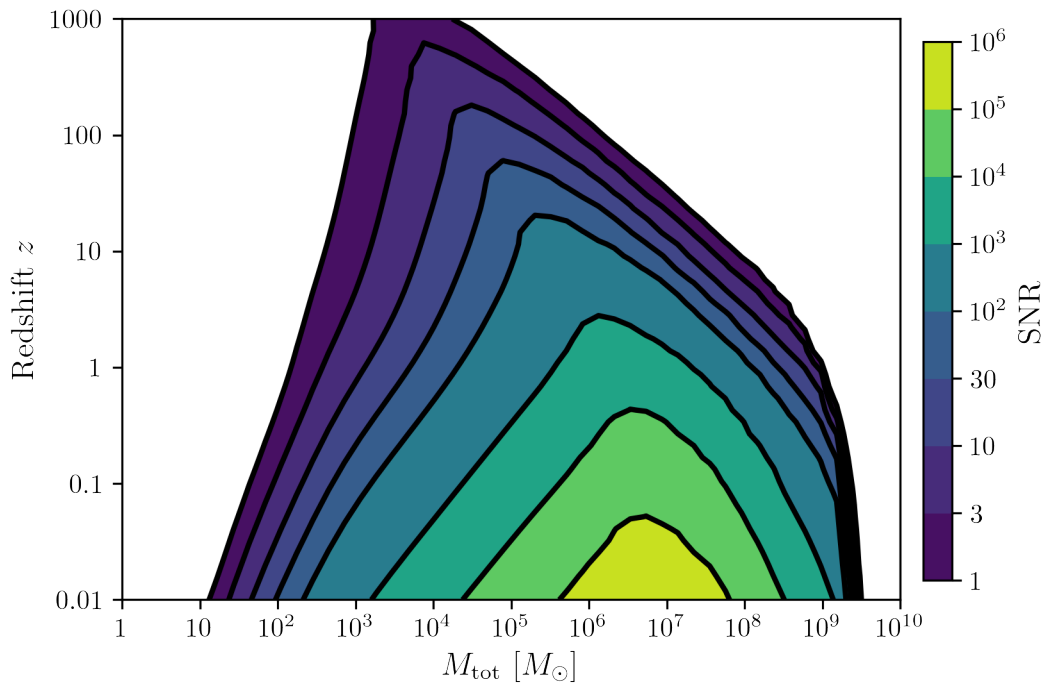
Changing Waveform Models

Thanks to swig wrapping, we can access the frequency domain waveforms within `lalsuite` (specifically any found here). User beware though, this is *mostly* untested. Depending on the waveform model, the SNR calculation could take much longer. To select between waveforms, simply change the `approximant` either in the source initialization, or inside the dictionary of extra parameters passed to `lalsuite`.

```
#Number of SNRMatrix rows
sampleRate_y = 50
#Number of SNRMatrix columns
sampleRate_x = 50
```

```
#Variable on y-axis
var_y = 'z'
#Variable on x-axis
var_x = 'M'

lalsuite_kwargs = {"S1x": 0.5, "S1y": 0, "S1z": 0.2,
                   "S2x": -0.2, "S2y": 0.5, "S2z": 0.1,
                   "inclination":np.pi/2}
instrument = Initialize_LISA()
source = Initialize_Source(instrument,approximant='IMRPhenomPv3',lalsuite_
↪kwargs=lalsuite_kwargs)
start = time.time()
[sample_x,sample_y,SNRMatrix] = snr.Get_SNR_Matrix(source,instrument,
                                                   var_x,sampleRate_x,
                                                   var_y,sampleRate_y)
end = time.time()

print('Model: ',instrument.name + '_' + var_x + '_vs_' + var_y,',',' done. t = : ',
↪end-start)
Plot_SNR(var_x,sample_x,var_y,sample_y,SNRMatrix,smooth_contours=False)
```

```
Model:  LISA_prop1_M_vs_z ,  done. t = :  431.48117208480835
```

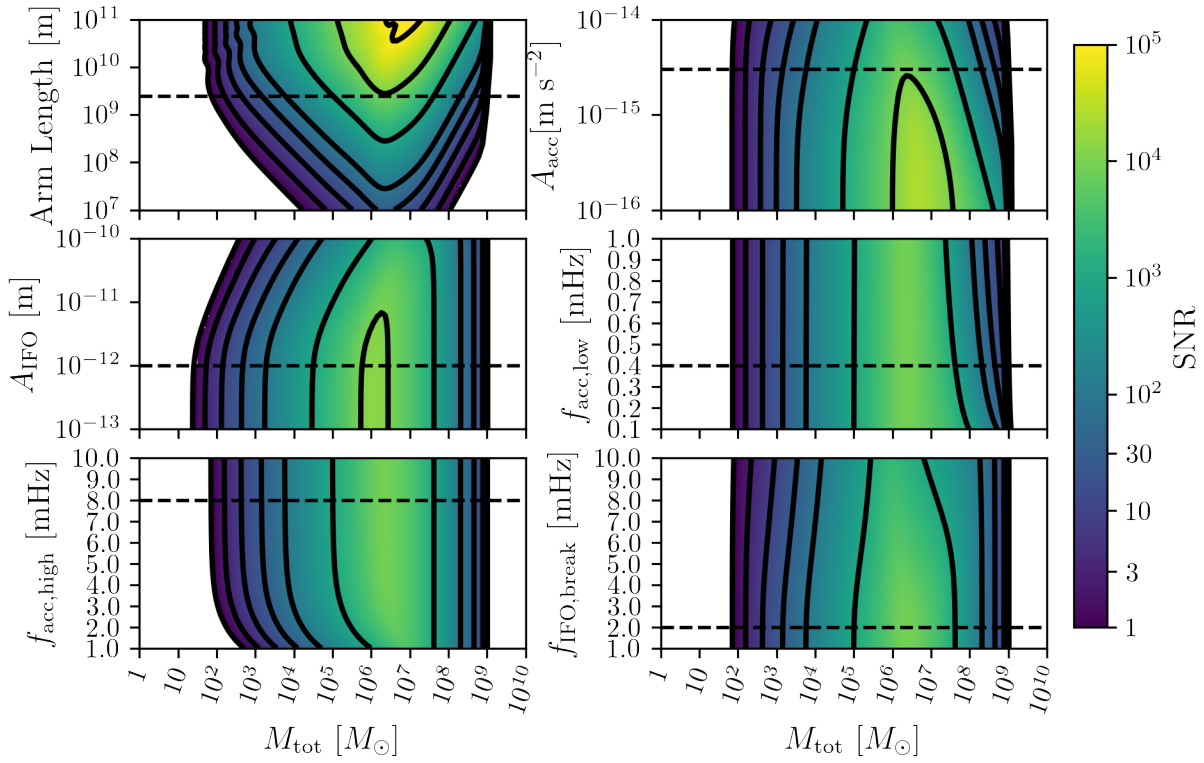



```
sampleRate_y = 100
sampleRate_x = 100
```

```
#Variable on y-axis
var_ys = ['L','A_acc','A_IFO','f_acc_break_low','f_acc_break_high','f_IFO_break']
#Variable on x-axis
var_x = 'M'
sample_x_array = []
sample_y_array = []
SNR_array = []
for var_y in var_ys:
    instrument = Initialize_LISA()
    source = Initialize_Source(instrument)
    start = time.time()
    [sample_x,sample_y,SNRMatrix] = snr.Get_SNR_Matrix(source,instrument,
                                                       var_x,sampleRate_x,
                                                       var_y,sampleRate_y)

    end = time.time()
    sample_x_array.append(sample_x)
    sample_y_array.append(sample_y)
    SNR_array.append(SNRMatrix)

    print('Model: ',instrument.name + '_' + var_x + '_vs_' + var_y,',',' done. t = :
↪',end-start)
```

```
Model:  LISA_prop1_M_vs_L ,  done. t = :  32.94680953025818
Model:  LISA_prop1_M_vs_A_acc ,  done. t = :  33.771636962890625
Model:  LISA_prop1_M_vs_A_IFO ,  done. t = :  33.917126417160034
Model:  LISA_prop1_M_vs_f_acc_break_low ,  done. t = :  34.29063296318054
Model:  LISA_prop1_M_vs_f_acc_break_high ,  done. t = :  34.608739614486694
Model:  LISA_prop1_M_vs_f_IFO_break ,  done. t = :  34.29920196533203
```

```
figsize = get_fig_size(scale=1.0)
fig, axes = plt.subplots(3,2,figsize=figsize)

#Can add lines on plot
fiducial_lines = [2.5e9,3e-15,1e-12,0.4*u.mHz.to('Hz'),8*u.mHz.to('Hz'),2*u.mHz.to('Hz
↪')]
loglevelMax=5.0
hspace = .15
wspace = .35

ii = 0
for i in range(np.shape(axes)[0]):
    for j in range(np.shape(axes)[1]):
        if ii == (np.shape(axes)[0]*np.shape(axes)[1])-1:
            Plot_SNR('M',sample_x_array[ii],var_ys[ii],
                     sample_y_array[ii],SNR_array[ii],
                     fig=fig,ax=axes[i,j],display=True,display_cbar=True,
                     logLevels_max=loglevelMax,y_axis_line=fiducial_lines[ii],
                     hspace=hspace,wspace=wspace,
                     xticklabels_kwargs={'rotation':70,'y':0.02},
                     ylabels_kwargs={'labelpad':5})
        elif ii == (np.shape(axes)[0]*np.shape(axes)[1])-2:
            Plot_SNR('M',sample_x_array[ii],var_ys[ii],
                     sample_y_array[ii],SNR_array[ii],
                     fig=fig,ax=axes[i,j],display=False,display_cbar=False,
                     logLevels_max=loglevelMax,y_axis_line=fiducial_lines[ii],
                     hspace=hspace,wspace=wspace,
                     xticklabels_kwargs={'rotation':70,'y':0.02},
                     ylabels_kwargs={'labelpad':5})
        else:
            Plot_SNR('M',sample_x_array[ii],var_ys[ii],
                     sample_y_array[ii],SNR_array[ii],
                     fig=fig,ax=axes[i,j],display=False,display_cbar=False,
                     logLevels_max=loglevelMax,y_axis_line=fiducial_lines[ii],
                     x_axis_label = False,
                     hspace=hspace,wspace=wspace,
                     xticklabels_kwargs={'rotation':70,'y':0.02},
                     ylabels_kwargs={'labelpad':5})
        ii += 1
```



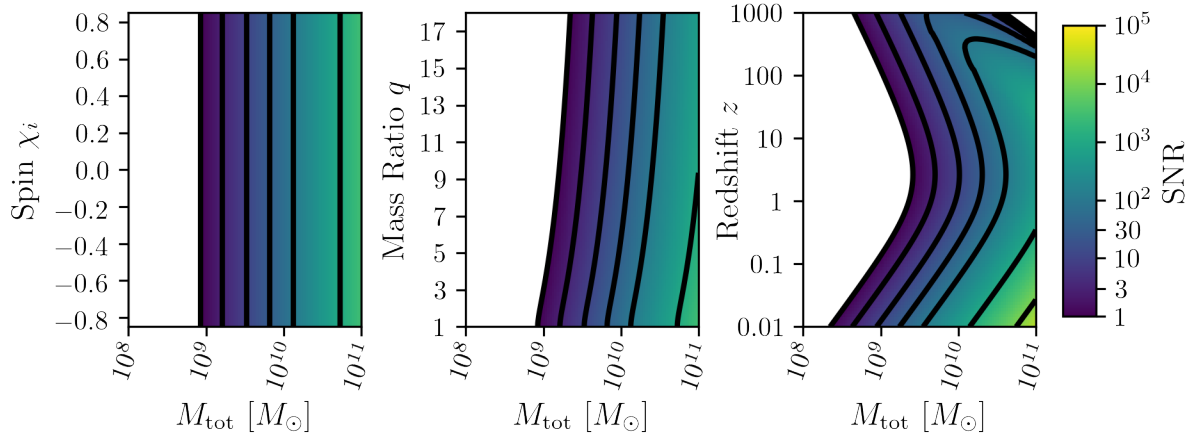
6.4.3 PTA SNRs

Same as the rest, just for example purposes!

```
#Variable on y-axis
var_ys = ['chi1','q','z']
#Variable on x-axis
var_x = 'M'
instrument = Initialize_NANOGrav()
sample_x_array = []
sample_y_array = []
SNR_array = []
for var_y in var_ys:
    source = Initialize_Source(instrument)
    start = time.time()
    [sample_x,sample_y,SNRMatrix] = snr.Get_SNR_Matrix(source,instrument,
                                                       var_x,sampleRate_x,
                                                       var_y,sampleRate_y)

    end = time.time()
    sample_x_array.append(sample_x)
    sample_y_array.append(sample_y)
    SNR_array.append(SNRMatrix)

    print('Model: ',instrument.name + '_' + var_x + '_vs_' + var_y,',',' done. t = :
↪',end-start)
```

```
Model:  NANOGrav_WN_M_vs_chi1 ,  done. t = :  24.73322606086731
Model:  NANOGrav_WN_M_vs_q ,  done. t = :  21.385414361953735
Model:  NANOGrav_WN_M_vs_z ,  done. t = :  22.074593544006348
```

```
figsize = get_fig_size()
fig, axes = plt.subplots(1,3,figsize=figsize)
loglevelMax=5.0
hspace = .1
wspace = .45

for i,ax in enumerate(axes):
    if i == (len(axes))-1:
        Plot_SNR('M',sample_x_array[i],var_ys[i],
                 sample_y_array[i],SNR_array[i],
                 fig=fig,ax=ax,display=True,display_cbar=True,
                 logLevels_max=loglevelMax,
                 hspace=hspace,wspace=wspace,
                 xticklabels_kwargs={'rotation':70,'y':0.02},
                 ylabels_kwargs={'labelpad':-5})
    else:
        Plot_SNR('M',sample_x_array[i],var_ys[i],
                 sample_y_array[i],SNR_array[i],
                 fig=fig,ax=ax,display=False,display_cbar=False,
                 logLevels_max=loglevelMax,
                 hspace=hspace,wspace=wspace,xticklabels_kwargs={'rotation':70,'y':0.
↪02})
```



There is also functionality to plot two different plots together for eazy comparison. Here we compare the methods in the previous calculation to a source with an inclination of $\pi/2$ and one using the alternate method of calculating the monochromatic strain with the phenomenological waveform amplitude.

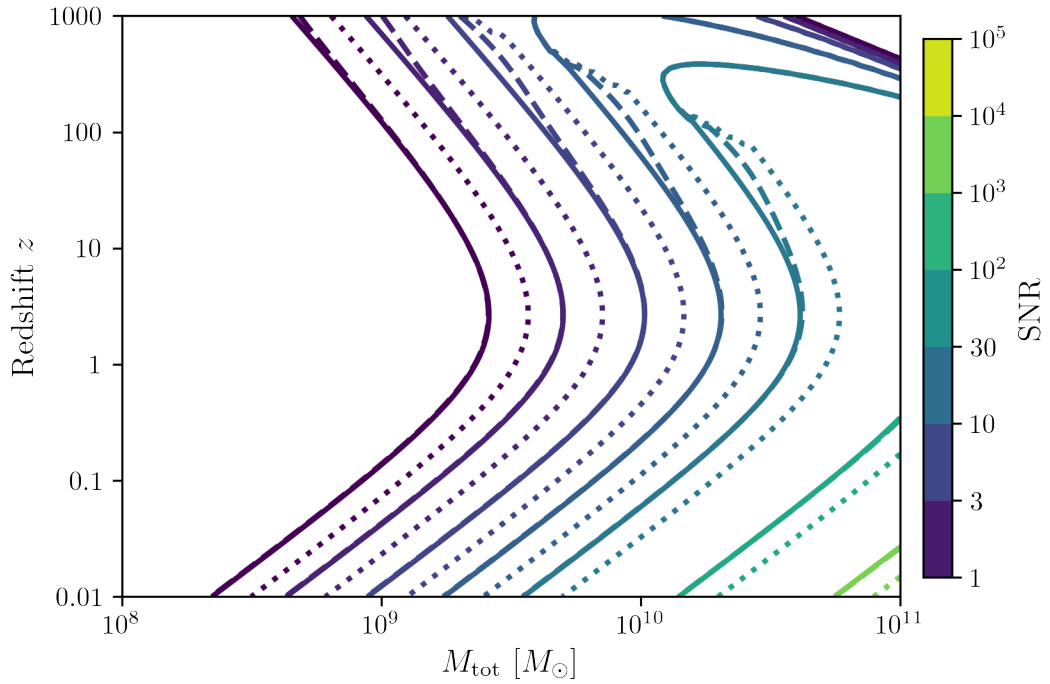
```
instrument = Initialize_NANOGrav()
inc = np.pi/2
source = Initialize_Source(instrument)
[sample_x_inclined,sample_y_inclined,SNRMatrix_inclined] = snr.Get_SNR_Matrix(source,
↪instrument,
                                                                              'M',
↪sampleRate_x,
                                                                              'z',
↪sampleRate_y,
                                                                              inc=inc)
[sample_x_PN_method,sample_y_PN_method,SNRMatrix_PN_method] = snr.Get_SNR_
↪Matrix(source,instrument,
                                                                                 'M',
↪sampleRate_x,
```

(continues on next page)

(continued from previous page)

```
                                                                    'z',
↪sampleRate_y,method='PN')
```

```
fig,ax = plt.subplots()
Plot_SNR('M',sample_x_array[-1],'z',sample_y_array[-1],SNR_array[-1],
         display=False,display_cbar=False,fig=fig,ax=ax,
         contour_kwargs={'cmap':'viridis'},cfill=False)
Plot_SNR('M',sample_x_inclined,'z',sample_y_inclined,SNRMatrix_inclined,
         display=False,display_cbar=False,fig=fig,ax=ax,
         contour_kwargs={'cmap':'viridis','linestyles':':'},cfill=False)
Plot_SNR('M',sample_x_PN_method,'z',sample_y_PN_method,SNRMatrix_PN_method,fig=fig,
↪ax=ax,
         contour_kwargs={'cmap':'viridis','linestyles':'--'},cfill=False)
```



These can take a long time if you vary the instrument parameters. Be careful with your sample rates!

```
sampleRate_x = 50
sampleRate_y = 50
```

```
#Variable on y-axis
var_ys = ['n_p','sigma','cadence','T_obs']
#Variable on x-axis
var_x = 'M'
sample_x_array = []
sample_y_array = []
SNR_array = []
for var_y in var_ys:
    instrument = Initialize_NANOGrav()
    source = Initialize_Source(instrument)
    start = time.time()
    [sample_x,sample_y,SNRMatrix] = snr.Get_SNR_Matrix(source,instrument,
```

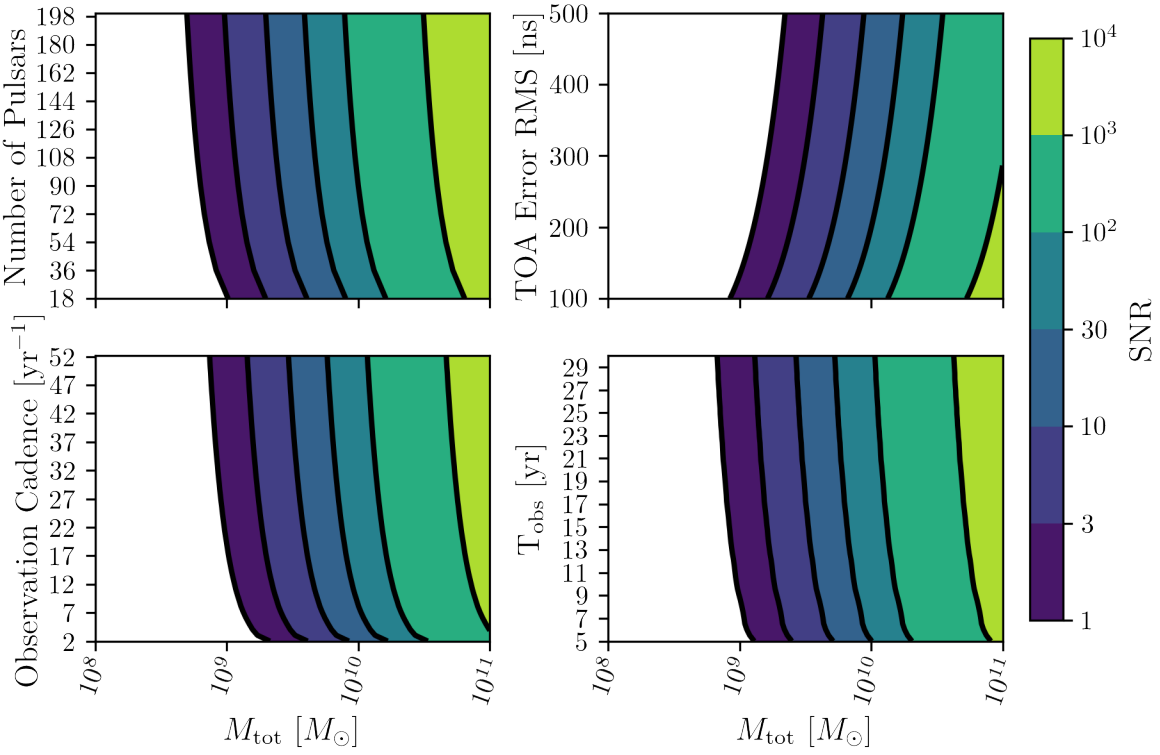
(continues on next page)

(continued from previous page)

```
                                                    var_x,sampleRate_x,
                                                    var_y,sampleRate_y)
    end = time.time()
    sample_x_array.append(sample_x)
    sample_y_array.append(sample_y)
    SNR_array.append(SNRMatrix)

    print('Model: ',instrument.name + '_' + var_x + '_vs_' + var_y,',',' done. t = :
↪',end-start)
```

```
Model:  NANOGrav_WN_M_vs_n_p ,  done. t = :  134.68967270851135
Model:  NANOGrav_WN_M_vs_sigma ,  done. t = :  171.6554970741272
Model:  NANOGrav_WN_M_vs_cadence ,  done. t = :  172.50515484809875
Model:  NANOGrav_WN_M_vs_T_obs ,  done. t = :  261.6520907878876
```

```
figsize = get_fig_size(scale=1.0)
fig, axes = plt.subplots(2,2,figsize=figsize)

loglevelMax=4.0
hspace = .2
wspace = .3
smooth = False

ii = 0
for i in range(np.shape(axes)[0]):
    for j in range(np.shape(axes)[1]):
        if ii == (np.shape(axes)[0]*np.shape(axes)[1])-1:
            Plot_SNR('M',sample_x_array[ii],var_ys[ii],
                     sample_y_array[ii],SNR_array[ii],
                     fig=fig,ax=axes[i,j],
                     logLevels_max=loglevelMax,
                     hspace=hspace,wspace=wspace,
                     smooth_contours=smooth,
                     xticklabels_kwargs={'rotation':70,'y':0.02},
                     ylabels_kwargs={'labelpad':5})
        elif ii == (np.shape(axes)[0]*np.shape(axes)[1])-2:
            Plot_SNR('M',sample_x_array[ii],var_ys[ii],
                     sample_y_array[ii],SNR_array[ii],
                     fig=fig,ax=axes[i,j],display=False,display_cbar=False,
                     logLevels_max=loglevelMax,
                     hspace=hspace,wspace=wspace,
                     smooth_contours=smooth,
                     xticklabels_kwargs={'rotation':70,'y':0.02},
                     ylabels_kwargs={'labelpad':2})
        else:
            Plot_SNR('M',sample_x_array[ii],var_ys[ii],
                     sample_y_array[ii],SNR_array[ii],
                     fig=fig,ax=axes[i,j],display=False,display_cbar=False,x_axis_
↪label=False,
                     logLevels_max=loglevelMax,
                     hspace=hspace,wspace=wspace,
                     smooth_contours=smooth,
                     xticklabels_kwargs={'rotation':70,'y':0.02},
                     ylabels_kwargs={'labelpad':2})
        ii += 1
```


Number of Pulsars
198
180
162
144
126
108
90
72
54
36
18
TOA Error RMS [ns]
500
400
300
200
100
SNR
10^4
10^3
10^2
30
10
3
1
Observation Cadence [yr^{-1}]
52
47
42
37
32
27
22
17
12
7
2
T_{obs} [yr]
29
27
25
23
21
19
17
15
13
11
9
7
5
10^8
10^9
10^{10}
10^{11}
M_{tot} $[M_\odot]$

CHAPTER 7

gwent

7.1 gwent package

7.1.1 Submodules

7.1.2 gwent.binary module

7.1.3 gwent.detector module

7.1.4 gwent.snr module

7.1.5 gwent.snrplot module

gwent.snrplot.**Get_Axes_Labels**(*ax*, *var_axis*, *var*, *var_scale*, *orig_labels*, *line_val*, *label_kwargs*, *tick_label_kwargs*, *line_kwargs*)

Gives paper plot labels for given axis

Parameters

ax: object The current axes object

var_axis: str The axis to change labels and ticks, can either be `'y'` or `'x'`

var: str The variable to label

orig_labels: list,np.ndarray The original labels for the particular axis, may be updated depending on parameter

line_val: int,float Value of line plotted on `var_axis` if not None. Assumed to be non-log10 value

label_kwargs: dict The dictionary adjusting the particular axis' label kwargs

tick_label_kwargs: dict The dictionary adjusting the particular axis' tick label kwargs

line_kwargs: dict The dictionary associated with the line displayed on `var_axis`

gwent.snrplot.**Plot_SNR**(*var_x*, *sample_x*, *var_y*, *sample_y*, *SNRMatrix*, *fig=None*, *ax=None*, *display=True*, *return_plt=False*, *dl_axis=False*, *lb_axis=False*, *smooth_contours=True*, *cfill=True*, *display_cbar=True*, *x_axis_label=True*, *y_axis_label=True*, *x_axis_line=None*, *y_axis_line=None*, *logLevels_min=-1.0*, *logLevels_max=0.0*, *hspace=0.15*, *wspace=0.1*, *contour_kwargs={}*, *contourf_kwargs={}*, *xticklabels_kwargs={}*, *xlabels_kwargs={}*, *xline_kwargs={}*, *yticklabels_kwargs={}*, *ylabels_kwargs={}*, *yline_kwargs={}*)

Plots the SNR contours from calcSNR

Parameters

var_x: str x-axis variable

sample_x: array samples at which `SNRMatrix` was calculated corresponding to the x-axis variable

var_y: str y-axis variable

sample_y: array samples at which `SNRMatrix` was calculated corresponding to the y-axis variable

SNRMatrix: array-like the matrix at which the SNR was calculated corresponding to the particular x and y-axis variable choices

fig: object, optional matplotlib figure object on which to collate the individual plots

ax: object, optional matplotlib axes object on which to plot the individual plot

display: bool, optional Option to turn off display if saving multiple plots to a file

return_plt: bool, optional Option to return `fig` and `ax`

dl_axis: bool, optional Option to turn on the right hand side labels of luminosity distance

lb_axis: bool, optional Option to turn on the right hand side labels of lookback time

smooth_contours: bool, optional Option to have contours appear smooth instead of tiered (depending on sample size the edges appear boxey).

cfill: bool, optional Option to use filled contours or not, default is `True`

display_cbar: bool, optional Option to display the colorbar on the axes object

x_axis_label: bool, optional Option to display the x axis label

y_axis_label: bool, optional Option to display the y axis label

x_axis_line: int,float, optional Option to display a line on the x axis if not None

y_axis_line: int,float, optional Option to display a line on the y axis if not None

logLevels_min: float, optional Sets the minimum log level of the colorbar, default is -1.0 which set the minimum to the log minimum of the given `SNRMatrix`

logLevels_max: float, optional Sets the maximum log level of the colorbar, default is 0.0, which sets the maximum to the log maximum value of the given `SNRMatrix`

hspace: float, optional Sets the vertical space between axes objects, default is 0.15

wspace: float, optional Sets the horizontal space between axes objects, default is 0.1

contour_kwargs: dict, optional Sets additional kwargs taken by contour in matplotlib

contourf_kwargs: dict, optional Sets additional kwargs taken by contourf in matplotlib

xticklabels_kwargs: dict, optional Sets additional kwargs taken by xticklabel in matplotlib

xlabels_kwargs=: dict, optional Sets additional kwargs taken by xlabel in matplotlib

xline_kwargs: dict, optional Sets additional kwargs taken by ax.axvline in matplotlib

yticklabels_kwargs: dict, optional Sets additional kwargs taken by yticklabel in matplotlib

ylabels_kwargs: dict, optional Sets additional kwargs taken by ylabel in matplotlib

yline_kwargs: dict, optional Sets additional kwargs taken by ax.axhline in matplotlib

7.1.6 gwent.utils module

gwent.utils.**DictError_2**()

gwent.utils.**DictError_3**()

gwent.utils.**DictError_Full**()

gwent.utils.**Get_Var_Dict**(*obj*, *value*)

Updates and initializes variable dictionaries used to keep track of current values and variable minima and maxima.

Parameters

obj: object Instance of class with parameter variables

value: array-like value(s) that are assigned into dictionary

Notes

value contains the variable name in the first index the next is the current value of the variable the last two are optional and contain the variable min and max

Examples

`obj.var_dict = ['M',value]` where obj is in this case an instance of a BinaryBlackHole

gwent.utils.**LenError_1**()

gwent.utils.**LenError_2**()

gwent.utils.**make_quant**(*param*, *default_unit*)

Convenience function to intialize a parameter as an astropy quantity.

Parameters

param: float, or Astropy Quantity Parameter to initialize

default_unit: str Astropy unit string, sets as default for param.

Returns

an astropy quantity

Notes

Taken from <https://github.com/Hazboun6/hasasia/blob/master/hasasia/sensitivity.py#L834>

Examples

self.f0 = make_quant(f0,'MHz')

7.1.7 gwent.waveform module

7.1.8 Module contents

Top-level package for gwent.

CHAPTER 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:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/ark0015/gwent/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.

8.1.2 Fix Bugs

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

8.1.3 Implement Features

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

8.1.4 Write Documentation

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

8.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ark0015/gwent/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 :)

8.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/gwent.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 gwent
$ cd gwent/
$ pip install -r requirements_dev.txt
$ pip install -r requirements.txt
$ pip install -e .
```

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:

```
$ make lint
$ make 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.

8.3 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 3.6+, and for PyPy. Check github actions and make sure that the tests pass for all supported Python versions.

8.4 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
```

CHAPTER 9

Credits

9.1 Development Lead

- Andrew Kaiser <andrewkaiser70@gmail.com>

9.2 Contributors

- Jeffrey S. Hazboun
- Sean T. McWilliams

CHAPTER 10

History

10.1 0.4.0 (2021-1-8)

- Adding functionality to use lalsuite waveforms
- Changing locations of multiple functions from BBHFrequencyDomain functions to general binary functions
- Moved several functions from binary to waveform
- Fixed several bugs with SNR calculations
- Clarified certain constants
- Added JOSS paper draft
- Expanded testing and switched from TravisCI to Github Actions

10.2 0.3.0 (2020-10-5)

- Turning pygwinc install into vendor inside gwent

10.3 0.2.1 (2020-10-5)

- Update to ground-based SNR calculation
- Minor changes to snrplot functionality
- Changed fitting for WD background
- Misc bug fixing

10.4 0.2.0 (2020-4-29)

- Major Changes to PTA detector setup
- Major Changes to snrplot
- Overhaul of tutorials
- Minor Changes to binary and snr functions
- Other Minor Changes sprinkled throughout

10.5 0.1.16 (2020-1-19)

- Removing install of pygwinc in setup.py
- pygwinc must now be manually installed

10.6 0.1.15 (2020-1-18)

- Including easy install of pygwinc

10.7 0.1.14 (2020-1-7)

- Major addition of pygwinc
- Minor fixes to snr sampling and plotting

10.8 0.1.13 (2019-10-28)

- Removing SNR Files in LoadFiles
- Fixing Error in PTA Initialization that Ignored User Input

10.9 0.1.12 (2019-10-8)

- Adding New Files for NANOGrav
- Updating Loading from files for Detectors

10.10 0.1.11 (2019-09-19)

- Fixing bugs and removing empty functionality

10.11 0.1.10 (2019-09-14)

- Removed Python 2.7 support

10.12 0.1.0 (2019-09-04)

- First release on PyPI.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

gwent, 64
gwent.snrplot, 61
gwent.utils, 63

h

hasasia, 7

Index

D

DictError_2() (*in module gwent.utils*), 63
DictError_3() (*in module gwent.utils*), 63
DictError_Full() (*in module gwent.utils*), 63

G

Get_Axes_Labels() (*in module gwent.snrplot*), 61
Get_Var_Dict() (*in module gwent.utils*), 63
gwent (*module*), 64
gwent.snrplot (*module*), 61
gwent.utils (*module*), 63

H

hasasia (*module*), 7, 28, 40

L

LenError_1() (*in module gwent.utils*), 63
LenError_2() (*in module gwent.utils*), 63

M

make_quant() (*in module gwent.utils*), 63

P

Plot_SNR() (*in module gwent.snrplot*), 61