gravpy Documentation

Release 0.1.0

Daniel Williams

Jun 21, 2019

Contents

Gravpy is a sandbox for gravitational wave astrophysics: it's not intended to be used for heavy-weight gravitational wave data analysis, and it started life as a series of iPython notebooks written up while I was reading through papers when I was starting out in my PhD, but it's started to snowball.

Gravpy is currently capable of plotting sensitivity curves for ground-based interferometers, and for pulsar timing arrays, and their antenna responses. It's also capable of calculating the spectra of CBC merger events, and making plots of those.

It's early days and there are lots of other things to be done with the package!

CHAPTER 1

User Guide

1.1 gravpy

A sandbox for gravitational wave astronomy.

- Free software: ISC license
- Documentation: https://gravpy.readthedocs.org.

1.1.1 Features

- TODO

1.1.2 Credits

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

1.2 Installation

At the command line:

```
$ easy_install gravpy
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv gravpy
$ pip install gravpy
```

1.3 GW150914 Example

Let's have a quick look at what Gravpy can do. In this example we'll have a look at the sensitivities of some of the current and historical interferometers, and then have a look at how GW150914, the first detected gravitational wave event would have looked in the detectors.

Let's get started by importing astropy's units module, and numpy, which we'll need:

```
import astropy.units as u
import numpy as np
```

1.3.1 Simulating an interferometer

A number of approaches to detecting gravitational waves have been discussed in the literature, and have been constructed, ranging from pulsar timing arrays to Weber bars. We're going to simulate some interferometers, like the LIGO detectors which made the detection of GW150914. To do this, gravpy has a package for interferometers, with some "pre-made" interferometers.

```
import gravpy.interferometers as ifo
```

We can now simulate some interferometers. Let's start with Advanced LIGO.

```
aligo = ifo.AdvancedLIGO()
```

We can take a look at the sensitivity curve using the *~aligo.plot()* method:

1.3.2 Simulating an event

Now that we have an interferometer, let's have a look at an event. We'll simulate a compact binary merger, or a "CBC".

The first observed gravitational wave event was a CBC, a binary black hole merger. We can simulate that event using gravpy.

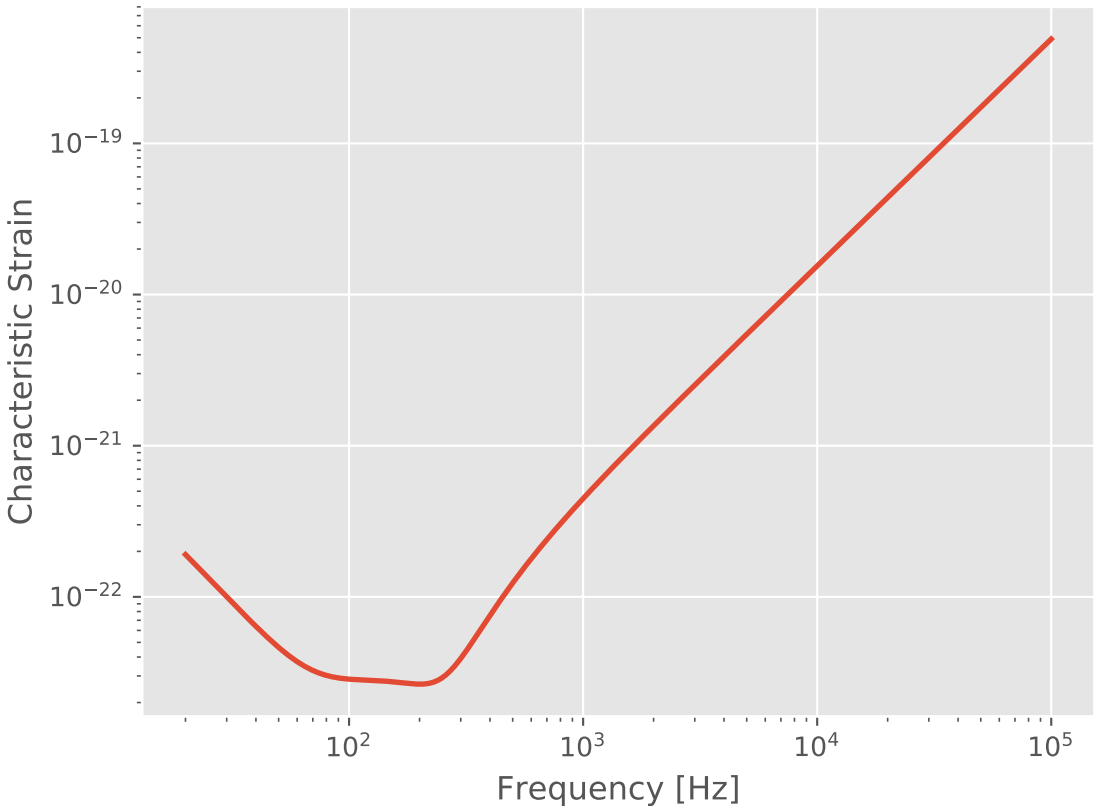
```
import gravpy.sources as sources
cbc = sources.CBC(frequencies=np.logspace(-4, 5, 1000) * u.hertz,
                  m1=32*u.solMass, m2=30*u.solMass, r=0.8*1e9*u.parsec)
```

Let's have a look at the frequency behaviour of the event:

Now that we have a detector and an event we can find out the SNR of the signal in the detector.:

```
print cbc.snr(o1_aligo)
   112.363423673
```

That's quite a bit higher than the SNR of the observed event, so what gives? We simulated the design sensitivity of the aLIGO detectors, but the event was discovered in the first observing run, which was well below design sensitivity. We can fix this by simulating the detector with its "O1" configuration.


Characteristic Strain
10^{-19}
10^{-20}
10^{-21}
10^{-22}
10^{2}
10^{3}
10^{4}
10^{5}
Frequency [Hz]

```
o1_aligo = ifo.AdvancedLIGO(configuration='O1')
```

Let's have a look at this and the event on a plot.

```
import matplotlib.pyplot as plt
import gravpy.interferometers as ifo
import gravpy.sources as sources
import astropy.units as u
o1_aligo = ifo.AdvancedLIGO(configuration='O1')
cbc = sources.CBC(frequencies=np.logspace(-4, 5, 1000) * u.hertz,
                  m1=32*u.solMass, m2=30*u.solMass, r=0.8*1e9*u.parsec)

plt.style.use('ggplot')
f, ax = plt.subplots(1)
o1_aligo.plot(ax)
cbc.plot(ax)
```

The SNR looks better now:

```
>>> print cbc.snr(o1_aligo)
24.8134701645
```

How about other interferometers?

```
>>> geo = ifo.GEO()
>>> iligo = ifo.InitialLIGO()
>>> tama = ifo.TAMA()
>>> virgo = ifo.VIRGO()
>>> aligo = ifo.AdvancedLIGO()
>>> o1_aligo = ifo.AdvancedLIGO(configuration='O1')
>>> elisa = ifo.EvolvedLISA()
>>> print "{} \t\t {}".format('IFO', 'SNR')
>>> print "------------------------------"
>>> for inter in [aligo, o1_aligo, elisa, iligo, virgo, geo, tama]:
...     print "{} \t\t {}".format(inter.name, cbc.snr(inter))
IFO              SNR
------------------------------
aLIGO            112.363423673
aLIGO [O1]       24.8134701645
eLISA            109.12468906
Initial LIGO     6.37979047218
VIRGO            7.86000380341
GEO600           4.80002280092
TAMA             0.258152593608
```

So we can see that this event wouldn't have exceeded an SNR of 8 in any of the previous generation of detectors, but would have been loud in eLISA.

1.4 Usage

To use gravpy in a project:

```
import gravpy
```

The gravpy package can currently produce sensitivity curves for interferometers.

CHAPTER 2

Developer Guide

2.1 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:

2.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/transientlunatic/gravpy/issues.

If you are reporting a bug, please include:

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

Fix Bugs

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

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

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

Submit Feedback

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

2.1.2 Get Started!

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

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

```
$ git clone git@github.com:transientlunatic/gravpy.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 gravpy
$ cd gravpy/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

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

```
$ flake8 gravpy tests
$ python setup.py test
$ tox
```

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

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

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

7. Submit a pull request through the GitHub website.

2.1.3 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 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/transientlunatic/gravpy/pull_requests and make sure that the tests pass for all supported Python versions.

2.1.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_gravpy
```

2.2 Credits

2.2.1 Development Lead

- Daniel Williams <d.williams.2@research.gla.ac.uk>

2.2.2 Contributors

None yet. Why not be the first?

2.3 gravpy

2.3.1 gravpy package

Subpackages

Submodules

gravpy.general module

gravpy.general.**snr**(*signal*, *detector*)

Calculate the SNR of a signal in a given detector, assuming that it has been detected with an optimal filter. See e.g. arxiv.org/abs/1408.0740

Parameters

signal [Source] A Source object which describes the source producing the signal, e.g. a CBC.

detector [Detector] A Detector object describing the instrument making the observation e.g. aLIGO.

Returns

SNR [float] The signal-to-noise ratio of the signal in the detector.

gravpy.gravpy module

gravpy.interferometers module

class gravpy.interferometers.**AdvancedLIGO**(*frequencies=None*, *configuration=None*, *obs_time=None*)

Bases: gravpy.interferometers.Interferometer

The aLIGO Interferometer

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self[, frequencies])	Calculate the one-sided power spectral desnity for a detector.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

S0 = <Quantity 1.e-49 1 / Hz>

configurations = {'O1': ['data/aligo_freqVector.txt', 'data/o1_data50Mpc_step1.txt']}

detector_tensor = <Quantity [[4., 0., 0.], [0., -4., 0.], [0., 0., 0.]] km>

f0 = <Quantity 215. Hz>

frequency_range = <Quantity [30., 4000.] Hz>

fs = <Quantity 20. Hz>

length = <Quantity 4. km>

name = 'aLIGO'

noise_spectrum(*self*, *x*)

Return the default noise spectrum for the interferometer.

Parameters

x [float] The rescaled frequency at which the fit should be evaluated.

Returns

float The sensitivity of the interferometer at the frequency provided.

xhat = array([1, 0, 0])

yhat = array([0, 1, 0])

zhat = array([0, 0, 1])

class gravpy.interferometers.**BDecigo**(*frequencies=None*, *configuration=None*, *obs_time=None*)

Bases: gravpy.interferometers.Interferometer

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self, frequencies)	Calculate the one-sided power spectral desnity for a detector.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

S0 = <Quantity 4.04e-46 1 / Hz>

frequencies = <Quantity [1.00000000e-02, 1.00092155e-02, 1.00184395e-02, ..., 9.981594

frequency_range = <Quantity [1.e-02, 1.e+02] Hz>

psd(*self*, *frequencies*)

Calculate the one-sided power spectral desnity for a detector. If a particular configuration is specified then the results will be returned for a spline fit to that configuration's curve, if available.

Parameters

frequencies [ndarray] An array of frequencies where the PSD should be evaluated.

configuration [str] The configuration of the detector for which the curve should be returned.

class gravpy.interferometers.**BigBangObservatory**(*frequencies=None*, *configuration=None*, *obs_time=None*)

Bases: gravpy.interferometers.Interferometer

The Big Bang Observatory.

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.

Continued on next page

Table 3 – continued from previous page

noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self, frequencies)	The power spectrum density of the detector, taken from equation 6 of arxiv:1101.3940.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

frequencies = <Quantity [1.00000000e-03, 1.00115207e-03, 1.00230547e-03, ..., 9.976998

frequency_range = <Quantity [1.e-03, 1.e+02] Hz>

psd(*self*, *frequencies*)

The power spectrum density of the detector, taken from equation 6 of arxiv:1101.3940.

class gravpy.interferometers.**Decigo**(*frequencies=None*, *configuration=None*, *obs_time=None*)

Bases: gravpy.interferometers.Interferometer

The full, original Decigo noise curve, from arxiv:1101.3940.

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self, frequencies)	The power spectrum density of the detector, taken from equation 5 of arxiv:1101.3940.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

fp = <Quantity 7.36 Hz>

frequencies = <Quantity [1.00000000e-02, 1.00092155e-02, 1.00184395e-02, ..., 9.981594

frequency_range = <Quantity [1.e-02, 1.e+02] Hz>

psd(*self*, *frequencies*)

The power spectrum density of the detector, taken from equation 5 of arxiv:1101.3940.

class gravpy.interferometers.**Detector**

Bases: object

This is the base class for all types of detectors, and contains the conversion methods between the various different ways of expressing the noise levels (sensitivity) of any detector.

Methods

energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
plot(self[, axis])	Plot the noise curve for this detector.

srpsd

energy_density(*self*, *frequencies=None*)

Produce the sensitivity curve of the detector in terms of the energy density.

Parameters

frequencies [ndarray] An array of frequencies, in units of Hz

Returns

energy_density [ndarray] An array of the dimensionless energy density of the sensitivity of the detector.

noise_amplitude(*self*, *frequencies=None*)

The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.

Parameters

frequencies [ndarray] An array of frequencies, in units of Hz

Returns

noise_amplitude [ndarray] An array of the noise amplitudes correcsponding to the input frequency values

plot(*self*, *axis=None*, ***kwargs*)

Plot the noise curve for this detector.

srpsd(*self*, *frequencies=None*)

class gravpy.interferometers.**EvolvedLISA**(*frequencies=None*, *configuration=None*, *obs_time=None*)

Bases: gravpy.interferometers.Interferometer

The eLISA Interferometer

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self, frequencies)	Calculate the one-sided power spectral desnity for a detector.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

L = <Quantity 1.e+09 m>

frequencies = <Quantity [1.00000000e-06, 1.00138264e-06, 1.00276720e-06, ..., 9.972404

fs = <Quantity 3.e-05 Hz>

name = 'eLISA'

psd(*self*, *frequencies*)

Calculate the one-sided power spectral desnity for a detector. If a particular configuration is specified then the results will be returned for a spline fit to that configuration's curve, if available.

Parameters

frequencies [ndarray] An array of frequencies where the PSD should be evaluated.

configuration [str] The configuration of the detector for which the curve should be returned.

class gravpy.interferometers.**GEO**(*frequencies=None*, *configuration=None*, *obs_time=None*)

Bases: gravpy.interferometers.Interferometer

The GEO600 Interferometer

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self[, frequencies])	Calculate the one-sided power spectral desnity for a detector.

Continued on next page

Table 7 – continued from previous page

skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

```
S0 = <Quantity 1.e-46 1 / Hz>
f0 = <Quantity 150. Hz>
fs = <Quantity 40. Hz>
name = 'GEO600'
```

noise_spectrum(*self*, *x*)

Return the default noise spectrum for the interferometer.

Parameters

x [float] The rescaled frequency at which the fit should be evaluated.

Returns

float The sensitivity of the interferometer at the frequency provided.

class gravpy.interferometers.**InitialLIGO**(*frequencies=None*, *configuration=None*, *obs_time=None*)

Bases: gravpy.interferometers.Interferometer

The iLIGO Interferometer

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self[, frequencies])	Calculate the one-sided power spectral desnity for a detector.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

```
S0 = <Quantity 9.e-46 1 / Hz>
f0 = <Quantity 150. Hz>
fs = <Quantity 40. Hz>
name = 'Initial LIGO'
```

noise_spectrum(*self*, *x*)
Return the default noise spectrum for the interferometer.

Parameters

x [float] The rescaled frequency at which the fit should be evaluated.

Returns

float The sensitivity of the interferometer at the frequency provided.

class gravpy.interferometers.**Interferometer**(*frequencies=None*, *configuration=None*, *obs_time=None*)
Bases: gravpy.interferometers.Detector

The base class to describe an interferometer.

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self[, frequencies])	Calculate the one-sided power spectral desnity for a detector.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd	

S0 = <Quantity 1.e-46 1 / Hz>

antenna_pattern(*self*, *theta*, *phi*, *psi*)
Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.

Parameters

theta [float] The altitude angle.

phi [float] The azimuthal angle.

psi [float or list] The polarisation angle. If psi is a list of two angles the returned antenna patterns will be the integrated response between those two polsarisation angles.

Returns

F+ [float] The antenna response to the '+' polarisation state.

Fx [float] The antenna response to the 'x' polsarisation state.

|F| [float] The combined antenna response (sqrt(F+^2 + Fx^2)).

detector_tensor = <Quantity [[4., 0., 0.], [0., -4., 0.], [0., 0., 0.]] km>

f0 = <Quantity 150. Hz>

frequencies = <Quantity [1.00000000e+01, 1.00230582e+01, 1.00461695e+01, ..., 9.954042

fs = <Quantity 40. Hz>

length = <Quantity 4. km>

name = 'Generic Interferometer'

noise_spectrum(*self*, *x*)
Return the default noise spectrum for the interferometer.

Parameters

x [float] The rescaled frequency at which the fit should be evaluated.

Returns

float The sensitivity of the interferometer at the frequency provided.

psd(*self*, *frequencies=None*)
Calculate the one-sided power spectral desnity for a detector. If a particular configuration is specified then the results will be returned for a spline fit to that configuration's curve, if available.

Parameters

frequencies [ndarray] An array of frequencies where the PSD should be evaluated.

configuration [str] The configuration of the detector for which the curve should be returned.

skymap(*self, nx=200, ny=100, psi=[0, 3.141592653589793]*)
Produce a skymap of the antenna repsonse of the interferometer.

Parameters

nx [int] The number of locations along the horizontal axis to produce the map at defaults to 200.

ny [int] The number of locations along the vertical axis to produce the map at defaults to 100

psi [float or list] The polarisation angle to produce the map at. If a list is given then the integrated response is given between those angles.

Returns

x [ndarray] The x values for the map

y: ndarray The y values for the map

antennap [ndarray] The values of the sensitivity in the + polarisation

antennax [ndarray] The values of the sensitivity in the x polarisation

antennac [ndarray] The values of the combined polarisation sensitivities

xhat = array([1, 0, 0])

yhat = array([0, 1, 0])

zhat = array([0, 0, 1])

class gravpy.interferometers.**LISA**(*frequencies=None*, *configuration=None*, *obs_time=None*)
Bases: gravpy.interferometers.Interferometer

The LISA Interferometer in its mission-accepted state, as of 2018

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
confusion_noise(self, frequencies[, ...])	The noise created by unresolvable galactic binaries at low frequencies.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
metrology_noise(self, frequencies)	Calculate the noise due to the single-link optical metrology, from arxiv:1803.01944.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self, frequencies)	Calculate the one-sided power spectral desnity for a detector.
single_mass_noise(self, frequencies)	The acceleration noise for a single test mass.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

L = <Quantity 2.5e+09 m>

confusion_noise(*self*, *frequencies*, *observation_time=0.5*)
The noise created by unresolvable galactic binaries at low frequencies.

frequencies = <Quantity [1.00000000e-05, 1.00115207e-05, 1.00230547e-05, ..., 9.976998

fs = <Quantity 3.e-05 Hz>

fstar = <Quantity 0.01908 Hz>

metrology_noise(*self*, *frequencies*)
Calculate the noise due to the single-link optical metrology, from arxiv:1803.01944.

name = 'eLISA'

psd(*self*, *frequencies*)
Calculate the one-sided power spectral desnity for a detector. If a particular configuration is specified then the results will be returned for a spline fit to that configuration's curve, if available.

Parameters

frequencies [ndarray] An array of frequencies where the PSD should be evaluated.

configuration [str] The configuration of the detector for which the curve should be returned.

single_mass_noise(*self*, *frequencies*)
The acceleration noise for a single test mass.

class gravpy.interferometers.**TAMA**(*frequencies=None*, *configuration=None*, *obs_time=None*)
Bases: gravpy.interferometers.Interferometer

The TAMA Interferometer

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self[, frequencies])	Calculate the one-sided power spectral desnity for a detector.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd

S0 = <Quantity 7.5e-46 1 / Hz>

f0 = <Quantity 400. Hz>

fs = <Quantity 75. Hz>

name = 'TAMA'

noise_spectrum (*self*, *x*)

Return the default noise spectrum for the interferometer.

Parameters

x [float] The rescaled frequency at which the fit should be evaluated.

Returns

float The sensitivity of the interferometer at the frequency provided.

class gravpy.interferometers.**TimingArray**

Bases: gravpy.interferometers.Detector

A class to represent a pulsar timing array.

Methods

energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
plot(self[, axis])	Plot the noise curve for this detector.

Pn	
Sn	
noise_spectrum	
psd	
srpsd	

Pn (*self*, *frequencies*)

Sn (*self*, *frequencies*)

T = <Quantity 15. yr>

dt = <Quantity 14. d>

frequencies = <Quantity [1.00000000e-10, 1.00926219e-10, 1.01861017e-10, 1.02804473e-1

n = 20

name = 'Generic PTA'

noise_spectrum (*self*, *frequencies*)

psd (*self*, *frequencies*)

sigma = <Quantity 100. ns>

zeta_sum = 4.74

class gravpy.interferometers.**Virgo** (*frequencies=None*, *configuration=None*, *obs_time=None*)

Bases: gravpy.interferometers.Interferometer

The Virgo Interferometer

Methods

antenna_pattern(self, theta, phi, psi)	Produce the antenna pattern for a detector, given its detector tensor, and a set of angles.
energy_density(self[, frequencies])	Produce the sensitivity curve of the detector in terms of the energy density.
noise_amplitude(self[, frequencies])	The noise amplitude for a detector is defined as $h^2_n(f) = f S_n(f)$ and is designed to incorporate the effect of integrating an inspiralling signal.
noise_spectrum(self, x)	Return the default noise spectrum for the interferometer.
plot(self[, axis])	Plot the noise curve for this detector.
psd(self[, frequencies])	Calculate the one-sided power spectral desnity for a detector.
skymap(self[, nx, ny, psi])	Produce a skymap of the antenna repsonse of the interferometer.

srpsd	

S0 = <Quantity 3.2e-46 1 / Hz>

f0 = <Quantity 500. Hz>

fs = <Quantity 20. Hz>

name = 'Virgo'

noise_spectrum(*self*, *x*)

Return the default noise spectrum for the interferometer.

Parameters

x [float] The rescaled frequency at which the fit should be evaluated.

Returns

float The sensitivity of the interferometer at the frequency provided.

gravpy.interferometers.**rot_x**(*theta*)

gravpy.interferometers.**rot_y**(*psi*)

gravpy.interferometers.**rot_z**(*phi*)

gravpy.sources module

gravpy.timingarray module

Module contents

2.4 History

2.4.1 0.1.0 (2016-03-15)

- First release on PyPI.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index