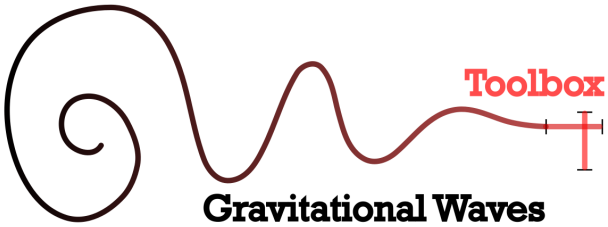

Gravitational Waves Universe Toolbox
Python Package User Manual
Version: 1.0

Shu-Xu Yi

Radboud University, Nijmegen, the Netherlands

March 31, 2021

Contents

1 Introduction

The Gravitational Waves Universe Toolbox (GWToolbox) python package are the bundle of python codes, on which the GWToolbox website is based. It can be installed and imported, as a more flexible way, comparing to the website interface, to simulate observation on various of GW sources, to explore the properties of detectors and pulsar timing array, and to incorporate the results with your codes of further searches and publications. In this manual, I will first give a instruction on the installation of the package in section II. And in section III, I will give tutorials on the usage of the tools in the toolbox.

2 Installation

The source codes can be downloaded from the repository:

The package is composed of three modules, namely the ground-based detectors (and their targets), the space-borne detectors (and their targets) and pulsar timing arrays (PTA). See illustration in figure 2.1. These three modules work independently, and have different dependencies on other packages and libraries. That means failed dependencies met in one module will not influence the usage of another module.

The three modules depend on some common python packages: `numpy`, `scipy`, `multiprocessing`, `pandas`, `astropy`.

I will introduce the dependencies indiviudally for the three modules in the following subsections. After the dependencies installed, the GWToolbox package can be directly imported, after you set the environment variable:

```
export PYTHONPATH=folder_contain_gwtoolbox/gwtoolbox
```

or, `cd` to `folder_contain_gwtoolbox/gwtoolbox`, and run

```
python setup.py install
```

this will install the python packages into your default path for python packages.

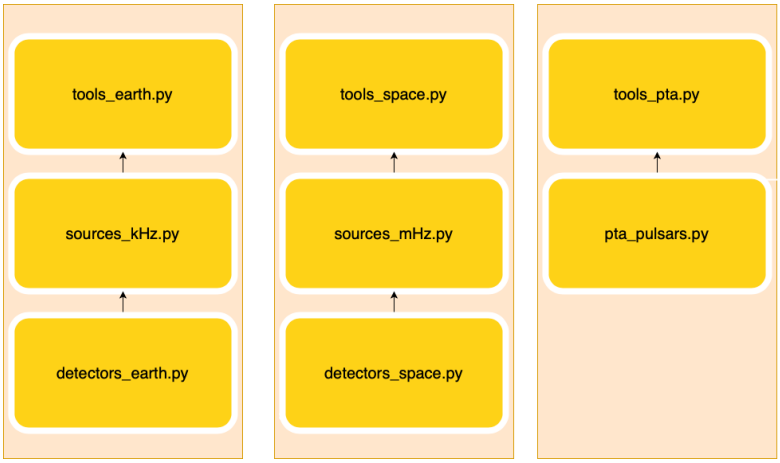

Figure 2.1: Three modules of the package

2.1 Ground-based detectors module

The functionality of simulating the noise properties of customised LIGO-like detectors is realized with `PyKat`[1]. It is a python wrapper for of the detector simulation tool `FINESSE`[2]. Make sure the environment variables `FINESSE_DIR` and `KATINI` are set correctly towards the folder containing executable `kat`.

2.2 Space-borne detectors

This module depends on the codes inside the Mock LISA data challenge (MLDC). The relevant parts of MLDC is already included in the GWToolbox:

`/gwtoolbox/gwtoolbox/MLDC-master`,

`cd` to the directory `/MLDC-master/Packages` and run:

```
python setup_lisaxml2.py install
```

then `cd` to the directory `/MLDC-master/Packages/common` and run:

```
python setup.py install
```

to install common packages needed by `MLDC`. Then to install packages that simulate the waveform of sources respectively: `cd` to the directory `/MLDC-master/Waveforms/MBH_IMR/IMRPhenomD` and run:

```
make
```

to install tools to simulate waveforms of supermassive black hole binary mergers. `cd` to the directory `/MLDC-master/Waveforms/fastGB` and run:

```
python setup.py install
```

to install tools to simulate waveforms of Galactic compact binaries.

The simulation on the waveform of EMRIs depends on an other tool `EMRI_Kludge_Suite`[3]. In the original `EMRI_Kludge_Suite`, the arm-length of LISA is hard-coded. I revised the source code to enable a different arm-length. The revised `EMRI_Kludge_Suite` is included in the GWToolbox: `/gwtoolbox/gwtoolbox/EMRI_Kludge_Suite`, `cd` to this directory, and run:

```
make
```

[1] https://pypi.org/project/PyKat/
[2] http://www.gwoptics.org/finesse/
[3] https://github.com/alvincjk/EMRI_Kludge_Suite

to build the binaries. The GSL and FFTW libraries are required for compilation. After this step, you will need to install the python wrapper. Make sure that in the setup.py, the variables `gsl` and `gslcblas` are set correctly to the path of the libraries, and make sure that your environment variable `LD_LIBRARY_PATH` is set properly so that the required libraries can be found. Then run:

```
python setup.py install
```

2.3 Pulsar Timing Arrays

This module has no special dependencies.

3 Usage

3.1 Ground-based detectors

3.1.1 Simulate Observations

Import the needed packages with:

```
from gwtoolbox.tools_earth import set_cosmology, Tools
```

You can set your cosmology model with:

```
cosmos=set_cosmology(cosmosID)
```

where the arguement `cosmosID` can take values:

- `'WMAP5'` The cosmology model implied by WMAP data release 2005
- `'WMAP7'` The cosmology model implied by WMAP data release 2007
- `'WMAP9'` The cosmology model implied by WMAP data release 2009
- `'Planck13'` The cosmology model implied by Planck data release 2013
- `'Planck15'` The cosmology model implied by Planck data release 2015

You can also define your own Λ-CDM cosmology model with:

```
cosmos=set_cosmology(H0=H0, Om0=Om0, Tcmb=Tcmb)
```

where H0 is the current Hubble constant (km/s/Mpc), Om0 is the current matter fraction, and Tcmb is the current CMB temperature (K). Then you can define your toolset with:

```
Toolset=Tools(detector_type=detector_type, event_type=event_type,
        population=population,
        cosmos=cosmos,
        det_setup=det_setup,
        new_theta=new_theta)
```

where the argument `detector_type` defines the detector that you want to simulate, it takes values:

- `'virgo'` advanced Virgo in the designed sensitivity;
- `'ligo'` advanced LIGO in the designed sensitivity;
- `'kagra'` KAGRA in the designed sensitivity;
- `'ligo-o3'` advanced LIGO in the O3 run;
- `'et'` Einstein telescope in design;
- `'ligo-like'` customised ligo-type detector;

the argument `event_type` defines the source type your want to simulate your observation on, it takes values:

- `'bhbh'` Black Hole Binary Mergers (BBH);
- `'nsns'` Double Neutron Star Mergers (DNS);
- `'bhns'` Black Hole-Neutron Star Mergers (BHNS);

the argument `population` defines the paramerization of the populatioin mode. It takes values `'I'` or `'II'` for BBH and BHNS, and only takes `'I'` for DNS. `cosmos` is the cosmology model you set with `set_cosmology`.

the argument `det_setup` is the configuration parameters for the customised ligo-type detector. For the default detectors,

```
det_setup=None
```

and for `detector_type='ligo-like'`, it takes value in the format:

```
det_setup=[["LARM_VALUE", "4000"],["PIN_VALUE", "125"],...]
```

which is a list of `[keyword, value_in_str]`. The meaning of the keywords are:

- `LARM_VALUE` Laser arm length in unit of meters;
- `PIN_VALUE` Laser Power in unit of Watts;
- `ITMT_VALUE` Cavity Mirror Transitivity;
- `SRMT_VALUE` Signal Recycling Mirror Transitivity;
- `PRMT_VALUE` Power Recycling Mirror Transitivity;
- `MTM_VALUE` Mirror Mass;

Default values will be used if not specified.

The argument `new_theta` sets the parameters in the population model. If set:

```
new_theta=None
```

default parameters will be used. The format of `new_theta` is a list of numbers,

- BBH-Population I: `[R0, tau, mu, c, gamma, mcut, ql, sig_x]`
- BBH-Population II: `[R0, tau, mu, c, gamma, mcut, m_peak, m_peak_scale, m_peak_sig ql, sig_x]`
- DNS-Population I: `[R0, tau, m_mean, m_scale, m_low, m_high, chi_sigma]`
- BHNS-Population I: `[R0, tau, m_mean, m_scale, m_low, m_high, mu, c, gamma, mcut, chi_sigma]`
- BHNS-Population II: `[R0, tau, m_mean, m_sclae, m_low, m_high, mu, c, gamma, mcut, m_peak, m_peak_scale, m_peak_sig, chi_sigma]`

See here to see the population models and the meaning of the parameters. After you defined the toolsets, you can all the tools to simulate the observation.

To get the expected number of detection:

```
num=Toolset.total_numer(time_obs=time_obs, rho_cri=rho_cri)
```

where the argument `time_obs` is the duration of observation in unit of minutes, `rho_cri` is the SNR threshold of detection.

To get a Pandas Dataframe of synthetic catalogue of detection:

```
df=Toolset.list_params_df(time_obs=time_obs, rho_cri=rho_cri,size=size)
```

where the argument `size` is the maximum number of sources to return, The meaning of the columns are:

- `z`: Cosmological redshift;
- `D(Mpc)`: Luminosity Distance in unit of Mpc;
- `m1`: Mass of the primary black hole in unit of solar mass;
- `m2`: Mass of the secondary black hole in unit of solar mass;
- `X`: Effective Spin;

To get a Pandas Dataframe of synthetic catalogue of detection with uncertainties:

```
df=Toolset.list_with_errors_df(time_obs=time_obs,
                               rho_cri=rho_cri,
                               size=size)
```

3.1.2 Antenna Patterns and Noise of Detector

Import needed package:

```
from detectors_earth import LigoLike, ETLike
```

To simulate a L-shaped LIGO-type detector, define:

```
detector=LigoLike(det)
```

where `det` takes values among `'virgo'`, `'ligo-o3'`, `'ligo'`, `'kagra'`, `'ligo-like'` and to simulate Einstein telescope:

```
detector=ETLike(det='ET')
```

The antenna patterns of the detector are:

```
Fplus,Fcross=detector.ante_pattern(theta,varphi,psi)
```

where `theta` is the polar angle of the GW source in the detector coordinates frame; `varphi` is the azimuth angle of the GW source in the detector coordinates frame; `psi` is the polarization angle of the GW.

The Noise Curve can be obtained with:

```
freq, strain=noise_curve(pars=det_setup)
```

where `det_setup` is in the format as in the above. When simulating default detectors, set `det_setup=None` and the returned `freq` is a list of frequency in unit of Hz, and `strain` is a list of strain in unit of $\sqrt{1/\mathrm{Hz}}$.

3.1.3 Detectability of Events and Source Population

Import the needed modules with:

```
from sources_kH import BHB, DNS, BHNS
```

when simulating observation of BBHs, define the population with:

```
population=BHB(cosmos)
```

where `cosmos` is the cosmology model set with `set_cosmology`. Populations of DNS and BHNS can be defined similarly. The default population model is parameterization I with default parameters. To adjust parameters, use:

```
population.set_model_theta(pop=population, new_theta=new_theta)
```

The argument `pop` and `new_theta` take values as in `Tools`.

Given a source with z, m_1 and m_2, the probability of detection of such a source can be calculated with:

```
prob=population.tel_fun(z, m1, m2, rho_cri=rho_cri,
                        ant_fun=detector.ante_pattern,
                        noise_tab=detector.noise_curve)
```

where `detector` is the detector defined above. The user can also use their own noise curve, by setting:

```
noise_tab=[list_of_frequencies, list_of_noise_strain]
```

The user can define their general own cosmic merger rate density as:

```
def cos_mer_rate(z,m1,m2):
    ....# arbitrary user defined population model
```

The number density distribution of expected detection is then:

```
def density_det(T, z, m1, m2):
    return 4*np.pi*T*cos_mer_rate(z,m1,m2)*
    population.tel_fun(z, m1, m2, rho_cri=rho_cri,
                        ant_fun=detector.ante_pattern,
                        noise_tab=detector.noise_curve)*
    cosmos.differential_comoving_volume(z).to_value(u.Gpc**3/u.sr)/(1+z)
```

The user can numerically integrate the above function to obtain the total expected detection, or using a MCMC algorithm to obtain the synthetic catalogue, with arbitrary population model.

3.2 Space-Borne detector

3.2.1 Simulate Observations

Import the needed packages with:

```
from gwtoolbox.tools_earth import set_cosmology, Tools
```

and `set_cosmology` is identical with that from `tools_earth`. The toolset can be defined with:

```
Toolset=Tools(popID=popID, cosmos=cosmos, det_setup=det_setup)
```

the argument `popID` take values of some float numbers, which correspond to different sources and population models:

- 0.1: Supermassive BHB-pop3;
- 0.2: Supermassive BHB-Q3_delays;
- 0.3: Supermassive BHB-Q3_nodelays;
- 1.1: Galactic WD binaries in Nelemans et al. (2001);
- 1.2: Verification WD binaries;
- 2.01-2.11: EMRIs M1-M11;

`cosmos` is the cosmology model defined with `set_cosmology`. The argument `det_setup` takes value of list of numbers:

`det_setup=[lisaLT, lisaP, lisaD]`

or

`det_setup=[lisaLT, lisaP, lisaD,Sacc0,Sopo0,Sops0]`

where:

- lisaLT: light travel time along the laser arm, in second;
- lisaP: laser power, in Watts;
- lisaD: the diameter of the telescope on the spacecraft, in unit of m;
- Sacc0: Accelaration noise in unit of Hz^{-1};
- Sopo0: Laser Shot Noise factor, in unit of Hz^{-1};
- Sops0: Other Optical Metrology system noises, in unit of Hz^{-1}.

To obtain the expected number of detection, and the synthetic catalogue, run:

```
dataframe, totnum=Toolset.dataframe(Tobs, rho_cri, size)
```

The input arguments:

- `Tobs` observation duration, in unit of year;
- `rho_cri` detection threshold of SNR;

- `size` maximum size of output catalogue;

the outputs `dataframe` is the catalogue in format of Pandas dataframe, and `totnum` is the expected number of detection (integer).

To obtain the catalogue with uncertanties:

```
dataframe_error, totnum=Toolset.errordataframe(Tobs, rho_cri, size)
```

3.2.2 Detectors

The LISA noise in TDI-X channel can be output with:

```
freq, Stdix=Toolset.noisecurve_X()
```

where `Toolset` is defined with `Tools` in above, `freq` is frequency in Hz, and `Stdix` is the corresponding noise in TDI-X response. If you only want to define the detector without defining the population, you can:

```
from gwtoolbox.detectors_space import LisaLike
detector=LisaLike(lisaLT=lisaLT,
                  lisaD=lisaD,
                  lisaP=lisaP,
                  Sacc0=Sacc0,
                  Sopo0=Sopo0,
                  Sops0=Sops0)
```

the detector parameters are define in above section. The Michaelson noise power, TDI-X response noises and the corresponding frequency are:

```
Sn=detector.Sn
Sx=detector.Sx
freq=detector.fq
```

The TDI-X noise response at any frequency can be calcualted with:

```
Sx=detector.Stdix(freq)
```

3.2.3 SNR of given single source

For supermassive BBH, SNR of single source can be calcualted with:

```
from sources_mHz import SMBHB
source=SMBHB(pop_model=None,cosmos=None)
source.SNR(p, det=detector)
```

where `detector` is the detector class set in above section, and `p` are the parameters of the source, which needs to be defined as follows:

```
from functions_space import *
from parameters import MBHBunits
list=[....]
# a list of parameters,
# the orders and units are defined in parameters.MBHBunits
dict={}
for i in len(MBHBunits.keys()):
    dict[ky[i]]=list[i]
p=ParsUnits(pars_i=dict, units_i=MBHBunits)
```

For Galactic Binaries:

```
from sources_mHz import GB
source=GB(pop_model=None,cosmos=None)
source.SNR_speed(pars, Tobs, dt, det=detector)
```

where `Tobs` is the observation duration in unit of years, `dt` is the time resolution in unit of seconds (doesn't enter the calculation in `SNR_speed`); `pars` is a list of parameters: `pars=[f0,f1,ra,dec,iota,psi]`, whose meaning are:

- `f0` the frequency of GW in unit of Hertz;
- `f1` first time derivative of frequency (not used in `SNR_speed`);
- `ra` Right Ascension in rad;
- `dec` Declination in rad;
- `iota` inclination angle in rad;
- `psi` polarization angle in rad;

For EMRI:

```
from sources_mHz import EMRI
source=EMRI(pop_model=None,cosmos=None)
source.SNR(pars_cat, det, Tobs)
```

the argument `pars_cat` is a list of parameters:

`pars_cat=[tPlunge,lgmu,lgM,e,nu,gam,phi,costhetaSky, phiSky, cosLambda, alpha, SMBHspin, costhetaSpin, phiSpin, Zeta]`

the meaning of the parameters are:

- `tPlunge` time to final plunge in seconds;
- `lgmu` logarithm (in base e) of stellar mass BH mass (in unit of seconds);
- `lgM` logarithm (in base e) of Supermassive BH mass (in unit of seconds);

Check Table I of Babrack & Cutler 2004 for the meaning of rest parameters.

3.3 Pulsar Timing Arrays

3.3.1 SNR of individual Supermassive BHB

Import needed module:

```
from tools_pta import PTA_individual
```

Define the toolset with:

```
toolset=PTA_individual(obs_plan=obs_plan,
                       delta_t=delta_t,
                       T=T,
                       Ndot=Ndot,
                       which_PTA=which_PTA)
```

the meaning arguments:

- `obs_plan="A"`: Observe with current PTAs and archival data;
- `obs_plan="B"`: Observe with current PTA and new observations;
- `obs_plan="C"`: Observe with current PTA and new observations plus possible new pulsars discovered in the future;
- `which_PTA` takes values of 'EPTA', 'NanoGrav', 'PPTA' or 'IPTA'.
- `delta_t`: the average days between observations, only meaningful for `obs_plan="B"` or `obs_plan="C"`;
- `T`: total observation duration (years) for future observations, only meaningful for `obs_plan="B"` or `obs_plan="C"`;
- `Ndot`: the number of new pulsars being included in the array every year;

Check the PTA with:

```
print(toolset.PTA)
```

Calculate the SNR of an individual supermassive BBH with:

```
toolset.SNR(ra, dec, hs, frequency, phi, iota)
```

where

- `ra`: RA in hh:mm:ss (string)
- `dec`: Declination in dd:mm:ss (string)
- `hs`: strain of GW;
- `frequency`: frequency of GW in Hz;
- `phi`: polarization anlge in rad
- `iota`: Inclination angle in rad;

The sky-average sensitivity curve for individual SMBHB:

```
hs, freq=toolset.sens_curve_skyave(rhostar=rhostar)
```

where `rhostar` is the SNR criterion, `freq` is list of frequencies in units of Hz.

3.3.2 Upper limit on Isotropic Stochastic GW background

Import needed module by:

```
from tools_pta import PTA_SGWB
```

and define the PTA toolset with:

```
toolset=PTA_SGWB(cosmos=cosmos,
                 obs_plan=obs_plan,
                 delta_t=delta_t,
                 T=T,
                 Ndot=Ndot,
                 which_PTA=which_PTA,
                 which_SGWB=which_SGWB,
                 index=index)
```

the new arguments are:

- `which_SGWB`: takes value among 'SBHBH', 'CS', 'Primordial' and 'selfdefine', stands for different origins of the stochastic GW background;
- `index`: when `which_SGWB='selfdefine'`, user can set customised spectrum index for the stochastic GW background;

To calculate the Upper limit for the characteristic strain:

```
uplimt_A=toolset.Upper_A(rho_cri)
```

where `rho_cri` is the SNR threshold, to calculate the upper limit in $h^2\Omega_{\rm GW}$:

```
uplimt_Edensity=toolset.UpperLimit(rho_cri)
```

To calculate the SNR for a given characteristic strain A in the PTA:

```
rho=toolset.SNR(A)
```