holodeck

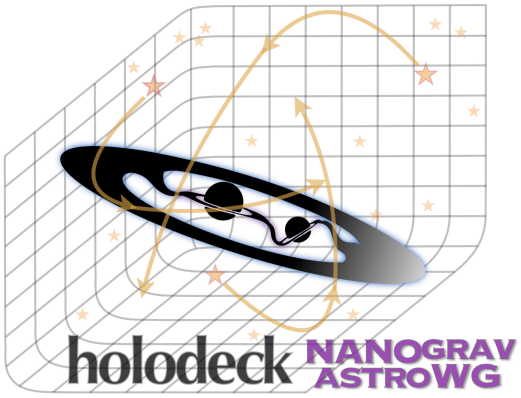
Release 1.2

NANOGrav

Feb 16, 2024

GETTING STARTED GUIDE

Massive Black-Hole Binary Population Synthesis for Gravitational Wave Calculations



holodeck on github

This package is aimed at providing a comprehensive framework for MBH binary population synthesis. The framework includes modules to perform population synthesis using a variety of methodologies from semi-analytic models, to cosmological hydrodynamic simulations, and even observationally-derived galaxy merger catalogs.

This File:

CHAPTER

ONE

GETTING STARTED: INTRODUCTION

This File:

1.1 Overview

The `holodeck` framework simulates populations of MBH binaries, and calculates their GW signals. In general, the calculation proceeds in three stages:

(1) *Populations*: Construct an initial population of MBH 'binaries'. This is typically done for pairs of MBHs when their galaxies merge (i.e. long before the two MBHs are actually a gravitationally-bound binary). The initial populations must specify, for each binary:

   (a) both MBH masses (typically as total-mass M and mass-ratio q),

   (b) the redshift (z) at which the pair of MBHs form,

   (c) the initial separation (a_{init}) of the MBHs at their formation time.

   Additional information can be very useful. In particular, information about the host galaxy of the MBH pair can be used in the binary evolution calculation.

(2) *Binary Evolution*: Evolve the binary population from their initial conditions (i.e. large separations) until they reach the regime of interest (i.e. small separations). In the simplest models, binaries are assumed to coalesce instantaneously, and are assumed to evolve purely due to GW emission. Note that these two assumptions are contradictory. More complex, self-consistent evolution models are recommended. These models typically involve interactions between MBH binaries and their host galaxies ('environmental' interactions). Note that the effects of binary evolution can be broken up into two distinct effects:

   (a) The redshift at which binaries reach the given frequencies (or separations) of interest, and similarly which binaries are able to reach those frequencies before redshift zero, and

   (b) The rate of binary evolution at the given frequencies of interest.

Cases which treat (a) and (b) consistently, we refer to as 'self-consistent' binary evolution models. Often this is not the case, for example assuming that (a) the redshift at which binaries reach all frequencies is identical and equal to the formation redshift (i.e. binaries merge 'instantaneously'); but also assuming that (b) binaries evolve due to GW emission alone.

(3) *Gravitational Waves*: From the population of MBH binaries at the separations (or frequencies) of interest, calculate the resulting GW signals. GW calculations can be done in many different ways, depending on what assumptions are made regarding:

(a) Discretization: whether binaries are treated as discrete objects, i.e. there can only be an integer number of binaries in a given frequency bin (this often relates to whether the number-density, or total-number of binaries is used in the calculation). One can also consider the effects of cosmic variance in this category as well.

(b) Evolution: whether self-consistent models of binary evolution are considered, or if purely GW-driven evolution is assumed (see *Binary Evolution*).

(c) Eccentricity: whether binaries are restricted to circular orbits, or allowed to have eccentric evolution. Eccentricity has multiple effects on binary evolution, mostly (i) by changing the rate of binary hardening, and (ii) by changing the GW frequencies corresponding to each orbital frequency. Circular binaries emit GWs at only the $n = 2$ harmonic of the orbital frequency, while eccentric binaries emit at all integer harmonics.

1.2 Populations

1.2.1 Semi-Analytic Models (SAMs)

`holodeck` SAMs are handled in the `holodeck.sams` module. The core of the module is the `Semi_Analytic_Model` class, in the: `holodeck.sams.sam` file.

The SAMs use simple, analytic components to calculate populations of binaries. The `Semi_Analytic_Model` handles and stores these components which themselves are defined in the `holodeck.sams.comps` file. Holodeck calculates the number-density of MBH binaries, by calculating a number-density of galaxy-galaxy mergers, and then converting from galaxy properties to MBH properties by using an MBH-host relationship.

The SAMs are initialized over a 3-dimensional parameter space of total MBH mass ($M = m_1 + m_2$), MBH mass ratio ($q = m_2/m_1 \leq 1$), and redshift (z). The `holodeck` code typically refers to the number of bins in each of these dimensions as `M`, `Q`, and `Z`; for example, the shape of the number-density of galaxy mergers will be `(M, Q, Z)`. Most calculations retrieve the number of binaries in the Universe at a given set of frequencies (or sometimes binary separations), so the returned values will be 4-dimensional with an additional axis with `F` frequency bins added. For example, the number of binaries at a given set of frequencies will typically be arrays of shape `(M, Q, Z, F)`.

Galaxy Mergers

`holodeck` SAMs always start with a Galaxy Stellar-Mass Function (GSMF) that determines how many galaxies there are as a function of stellar mass, $\psi(m_\star) \equiv \partial n_\star / \partial \log_{10} m_\star$, where $n_\star$ is the comoving number density of galaxies. We then have to add a galaxy merger rate (GMR), $R_\star(M_\star, q_\star) \equiv (1/n_\star)\partial^2 n_{\star\star}/\partial q_\star\, \partial t$, to find the number density of galaxy-pairs:

$$\frac{\partial^3 n_{\star\star}(M_\star, q_\star, z)}{\partial \log_{10} M_\star\, \partial q_\star\, \partial z} = \psi(m_{1,\star})\, R_\star(M_\star, q_\star).$$

Here, $M_\star = m_{1,\star} + m_{2,\star}$ is the total stellar mass of both galaxies, and $q_\star = m_{2,\star}/m_{1,\star} \leq 1$ is the stellar mass ratio. Often in the literature, the GMR is estimated as a galaxy pair fraction (GPF; $P_\star$) divided by a galaxy merger timescale (GMT; $T_\star$), i.e. $R_\star \approx P_\star / T_\star$. The GPF is typically an observationally-derived component, defined roughly as, $P_\star(m_{1,\star}, q_\star) \equiv N_{\star\star}(m_{1,\star}, q_\star)/N_\star(m_{1,\star})$, i.e. the number of galaxy pairs in a given survey divided by the number of all galaxies in the parent sample. Note that there are significant selection effects in determining the number of

galaxy pairs, including cuts on galaxy brightness/mass, and especially on the separations a_0 and a_1 between which pairs can be identified robustly. The GMT is typically derived from numerical simulations, and defined roughly as, $T_\star(M_\star, q_\star) \equiv \int_{a_0}^{a_1} [da/dt]^{-1}_{\star\star}\, da$, i.e. the total time that the galaxy pair spends at separations between a_0 and a_1. So we can also write:

$$\frac{\partial^3 n_{\star\star}(M_\star, q_\star, z)}{\partial \log_{10} M_\star\, \partial q_\star\, \partial z} = \psi(m_{1,\star})\, \frac{P_\star(m_{1,\star}, q_\star)}{T_\star(M_\star, q_\star)}.$$

Implementation: Each component (GSMF, GMR, GPF, GMT) is implemented by constructing a class that inherits from the appropriate base classes:

- GSMF: *_Galaxy_Stellar_Mass_Function*, for example the *GSMF_Schechter* class.
- GMR: *_Galaxy_Merger_Rate*, for example the *GMR_Illustris* class.
- GPF: *_Galaxy_Pair_Fraction*, for example the *GPF_Power_Law* class.
- GMT: *_Galaxy_Merger_Time*, for example the *GMT_Power_Law* class.

The classes need to expose a `__call__` method (i.e. the class instances themselves are callable) which accepts the appropriate arguments and returns the particular distribution.

MBH Populations and MBH-Host Relations

We now have a galaxy-galaxy merger rate, and we need to populate these galaxies with MBHs. To do this, we need an MBH-host relationship, typically in the form of M-MBulge ($m_{\rm BH} = M_\mu(m_{\rm bulge}, z)$; mass of the MBH, relative to the stellar-bulge mass of the host galaxy), and possibly a relationship between bulge mass and overall stellar-mass (i.e. $m_{\rm bulge} = m_{\rm bulge}(m_\star)$). Given this relationship, we can convert to MBH mergers as,

$$\frac{\partial^3 n(M, q, z)}{\partial \log_{10} M\, \partial q\, \partial z} = \frac{\partial^3 n_{\star\star}(M_\star, q_\star, z)}{\partial \log_{10} M_\star\, \partial q_\star\, \partial z} \left[\frac{\partial M_\star}{\partial M}\right] \left[\frac{\partial q_\star}{\partial q}\right],$$

where the masses must be evaluated at the appropriate locations: $m_1 = M_\mu(m_{1,\star})\,\&\, m_2 = M_\mu(m_{2,\star})$.

Implementation: M-MBulge relationships are implemented as subclasses inheriting from the `holodeck._MMBulge_Relation` class (defined in the *holodeck.relations* file), for example the *holodeck.relations.MMBulge_KH2013* class. Subclasses must implement a number of methods to allow for conversion between stellar bulge-mass and MBH mass.

1.2.2 'Discrete' Illustris Populations

1.3 Binary Evolution

1.4 Gravitational Waves

CHAPTER

TWO

GENERATING AND USING HOLODECK LIBRARIES

File Contents

2.1 Overview

holodeck 'libraries' are collections of simulations in which a certain set of parameters are varied, producing different populations and/or GW signatures at each sampled parameter value. Libraries are run from the same parameter-space and using the same hyper parameters (for example, the functional form that is assumed for the galaxy stellar-mass function). Libraries are constructed using the *librarian* module, with a 'parameter space' class that organizes the different simulations. The base-class is called *_Param_Space* (defined in the *holodeck.librarian.params* file), and all parameter space classes inherit from this, and should typically be prefixed by `PS_` to denote that they are parameter spaces. The parameter-space subclasses implement a number of parameters that are varied. Each parameter is implemented as a subclass of *_Param_Dist*, for example the *PD_Uniform* class that implements a uniform distribution.

As an example, the fiducial library and parameter space for *the 15yr astrophysics analysis* was the 'phenomenological uniform' library, implemented as *PS_Classic_Phenom_Uniform* (at the time, it was internally called `PS_Uniform_09B`). This library spanned a 6D parameter space using a 'phenomenological' binary evolution model, and assuming a uniform distribution in sampling from the parameter priors. Two parameters from the galaxy stellar-mass function were varied, along with two parameters from the M-MBulge relationship, and two parameters from the hardening model.

2.2 Parameter Spaces

NOTE: currently parameter-spaces are only designed for use with SAMs.

Parameter spaces are implemented as subclasses of the *_Param_Space* class. The class generates a certain number of samples using a latin hypercube to efficiently sample the parameter space. Each parameter being varied in the parameter space corresponds to parameter distribution, implemented as a *_Param_Dist* subclass. These parameter distributions convert from uniform random variables (samples in the latin hypercube) to the desired distributions. For example, the `PD_Normal(mean, stdev)` class draws from a normal (Gaussian) distribution, and the `PD_Normal(min, max)` class draws from a uniform distribution.

Parameter spaces must subclass *_Param_Space*, and provide 4 elements:

(0) OPTIONAL/Recommended: A class attribute called `DEFAULTS` which is a `dict` of default parameter values for all of the parameters needed by the initialization methods. **This is strongly recommended to ensure that parameters are set consistently, by setting them explicitly.**

(2) An `_init_sam()` a function that takes the input parameters, and then constructs and returns a *Semi_Analytic_Model* instance.

(2) An `_init_hard()` a function that takes the input parameters, and then constructs and returns a `_Hardening` instance.

(4) An `__init__()` method that passes all required parameter distributions (*_Param_Dist* subclasses) to the superclass `__init__()` method.

Public parameter spaces should also be 'registered' to the *holodeck.librarian.param_spaces_dict* dictionary. See *holodeck.librarian*.

2.3 Generating Libraries

CHAPTER

THREE

THE NANOGRAV 15YR ASTROPHYSICS ANALYSIS

Holodeck was used to perform the 'Astrophysical Interpretation' of the 15yr NANOGrav dataset, by the NANOGrav Astrophysics Working Group. The publication can be found here: Agazie+2023 (for the NANOGrav Collaboration) - The NANOGrav 15 yr Data Set: Constraints on Supermassive Black Hole Binaries from the Gravitational-wave Background. The astrophysics datasets can be found here: LINK. The analysis notebooks can be found here: LINK.

The initial set of **NANOGrav 15yr papers** are:

- The 15yr Dataset [N15data]
- Detector and Noise Characterization [N15detchar]
- Measuring a Gravitational Wave Background [N15GWB]
- Astrophysical Interpretation [N15astro]
- Constraints on New Physics [N15NP]
- Search for Anisotropy [N15anisotropy]
- Search for Individual Sources (CWs) [N15CWs]

The use of **holodeck in the 15yr astrophysics interpretation**:

CHAPTER

FOUR

DEFINITIONS AND ABBREVIATIONS

4.1 Abbreviations

- **AGN**: Active Galactic Nucleus, an accreting massive black hole that produced observable EM emission (radio, optical, X-ray, etc). Generally used synonymously with "Quasar" which is traditionally a very bright AGN, produced by an MBH accreting at a very high rate.
- **BH**: Black Hole
- **CBD**: Circum-Binary Disk, an accretion disk surrounding both components of a binary system.
- **CW**: Continuous Wave, as in Continuous Wave GW source
- **DF**: Dynamical Friction, the drag force experienced by a massive object moving in a gravitating background.
- **EM**: Electromagnetic, traditional radiation produced by moving charges (from the radio to optical to gamma-ray).
- **GR**: General relativity.
- **GSMF**: Galaxy Stellar-Mass Function, the number-density of galaxies as a function of their stellar mass.
- **GW/GWB**: Gravitational Waves / Gravitational Wave Background
- **ISCO**: Inner-most Stable Circular Orbit, the radius or location within which no object can maintain a stable orbit around a black hole. The radius is dependent on the spin of the black hole, but equal to 3 times the Schwarzschild radius for a non-spinning ($a = 0$) black hole.
- **LISA**: Laser Interferometer Space Antenna, future space-based GW detector
- **MBH**: Massive Black-Hole, usually treated interchangeably with 'SMBH' (super-massive black-hole)
- **MBHB**: Massive Black-Hole Binary, also referred to as BSMBH, SMBHB, ...
- **NFW**: Navarro-Frenk-White dark matter radial-density profile for a dark-matter halo.
- **PTA**: Pulsar Timing Array.
- **QLF**: Quasar Luminosity Function. The number-density of quasars/AGN as a function of their brightness.
- **SAM**: Semi-Analytic Models. (Grey area relative to SEMs).
- **SEM**: Semi-Empirical Models. (Grey area relative to SAMs).

- **TOA**: Time of Arrival, the specific measured time at which a pulsar's radio bursts are measured by the observer.

4.2 Symbols

- a**:**
  - (1) binary orbital semi-major axis (i.e. separation).
  - (2) the scale-factor of the universe at a given redshift, $a = 1/(1+z)$.
- d_c: Comoving Distance, the distance to a location in the Universe that, for zero relative velocity, remains invariant as a function of redshift. Related to luminosity distance as, $d_c = d_L/(1+z)$.
- d_L: Luminosity Distance, the effective distance to a source in the Universe determining its EM brightness or GW amplitude. Related to comoving distance as, $d_L = d_c\,(1+z)$.
- $\mathcal{M}$‘: Chirp-mass
- M: Mass, or total-mass ($M = m_1 + M_2$) in the context of a binary.
- q: Mass-ratio of a binary, defined to be less-than or equal to unity, $q \equiv m_2/m_1$, where the more massive primary is m_1.
- z: (Cosmological) redshift, a proxy for the distance to a source/location in the Universe, and also a proxy for the age of the Universe at that time. Related to the scale-factor as $z = (1/a) - 1$.

4.3 Terminology

- **AGN/Quasar**: There is a long and convoluted history of terms for different types of EM-bright astronomical sources that are now all believed to be produced by accreting massive black holes. ‘Active galactic nuclei’ is perhaps the most generic term for any accreting MBH which is then producing EM emission. Formally ‘quasars’ are relatively massive MBHs that are accreting at very high rates, and are thus very bright. Often ‘quasar’ is used relatively interchangeably with AGN (for instance in ‘Quasar Luminosity Function’).
- **Hardening**: the shrinking of the semi-major axis of a binary system by extracting energy and angular momentum. This term is also often used to simply mean the process of bringing two bodies (i.e. MBHs) closer together, even if they are not formally a gravitational-bound binary. ‘Hardening’ is typically due to either GW emission, or interactions between the binary and the surrounding environment. More generally, the “hardness” of a binary refers to how its binding energy compares to the kinetic energy of the surrounding environment (typically stars). If the binding energy is larger, then the binary is said to be ‘hard’; if the binding energy is less, it is said to be ‘soft’. The “Heggie Law” or the “Heggie-Hills Law” states that hard binaries tend to harden further, and soft binaries tend to soften further, both based on interactions with the surrounding medium. This is based on a pair of papers: [Heggie1975] and [Hills1975].
- **M-Mbulge**: the $M_{bh} - M_{mathrmbulge}$ relation, whereby the mass of MBHs is closely correlated with the mass of the stellar bulge of their host galaxy.
- **Stellar Bulge**

CHAPTER

FIVE

BIBLIOGRAPHY

5.1 Annotated Bibliography

- **Begelman, Blandford & Rees 1980**, [BBR1980] - Massive black hole binaries in active galactic nuclei
  - The definitive early discussion of massive black-hole binary evolution, outlining the different stages of environmental interaction (dynamical friction, stellar scattering, etc) and mentioning the possibility of stalling in the parsec regime.
  - Includes simplistic, but useful prescriptions for calculating timescales for each regime of evolution.
- **Genel et al. 2014**, [Genel2014] - Introducing the Illustris project: the evolution of galaxy populations across cosmic time
  - One of the standard references for the original Illustris simulations written by the Illustris team.
  - Focuses on the redshift evolution of simulated galaxies.
- **Hogg 1999**, [Hogg1999] - Distance measures in cosmology.
  - This is the go-to reference/cheat-sheet for basic cosmological calculations such as distances (comoving, luminosity), volume of the universe, lookback times, etc.
- **Kelley, Blecha, and Hernquist 2017**, [Kelley2017a] - Massive black hole binary mergers in dynamical galactic environments
  - Describes the MBH-MBH mergers from the Illustris cosmological hydrodynamic simulations.
  - Results include comprehensive semi-analytic models for post-processing the binary mergers at sub-grid scales.
- **Kelley et al. 2017**, [Kelley2017b] - The gravitational wave background from massive black hole binaries in Illustris: spectral features and time to detection with pulsar timing arrays
  - Uses the MBH-MBH merger catalogs from Illustris, along with comprehensive semi-analytic models of the unresolved binary evolution process, to calculate the expected properties of the GWB and PTA detection prospects.
- **Kelley et al. 2018**, [Kelley2018] - Single sources in the low-frequency gravitational wave sky: properties and time to detection by pulsar timing arrays
  - Uses the MBH-MBH merger catalogs from Illustris, along with comprehensive semi-analytic models of the unresolved binary evolution process, to calculate the expected properties of individual continuous wave (CW) GW sources and PTA detection prospects.
- **Nelson et al. 2015**, [Nelson2015] - The illustris simulation: Public data release
  - One of the standard references for the original Illustris simulations written by the Illustris team.
  - Summarizes the Illustris public data and API.

- **Phinney 2001**, [Phinney2001] - A Practical Theorem on Gravitational Wave Backgrounds
  - Pioneering analytic calculation of the GWB by integrating the GW emission of binaries over the history of the universe.
- **Rodriguez-Gomez et al. 2015**, [Rodriguez-Gomez2015] - The merger rate of galaxies in the Illustris simulation: a comparison with observations and semi-empirical models
  - Methods and results for galaxy-galaxy merger rates from the Illustris simulations.
  - These rates are used to prescribe merger rates in the observational-populations *holodeck* catalogs.
- **Sesana et al. 2008** [Sesana2008] - The stochastic gravitational-wave background from massive black hole binary systems: implications for observations with Pulsar Timing Arrays.
  - Thorough description of how to calculate the GWB, with a discussion on some of the nuances.
  - Particular attention is given to the difference between the analytic formalism of [Phinney2001] and numerical / semi-analytic approaches, i.e. the effects of discreteness of binary sources which produces a turnover in the GWB spectrum at high frequencies.
- **Sijacki et al. 2015**, [Sijacki2015] - The Illustris simulation: the evolving population of black holes across cosmic time
  - One of the standard references for the original Illustris simulations written by the Illustris team.
  - Describes the MBH/AGN population derived from the simulations.
- **Siwek, Weinberger, and Hernquist 2023**, [Siwek2023] - Orbital evolution of binaries in circumbinary discs
- **Springel 2010**, [Springel2010] - E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh
  - Methods paper for the arepo hydrodynamics code, used in the Illustris simulations.
- **Vogelsberger et al. 2014**, [Vogelsberger2014] - Introducing the Illustris Project: simulating the coevolution of dark and visible matter in the Universe
  - One of the standard references for the original Illustris simulations written by the Illustris team.
  - Gives a summary of the simulation methodology and results.

5.2 Quick References

These are provided here for easy copy-and-paste usage in other files.

5.3 NASA/ADS Quick-Reference

Generating references on NASA ADS:

- For short references (e.g. in code files):

```
* [%1.1h%Y]_ %3.1M (%Y).
```

- For the *Annotated Bibliography* section:

```
* **%3.1M %Y**, [%1.1h%Y]_ - `%T <%u>`_\n
```

- For full references (e.g. in the *Quick References* section above):

```
.. [%1.1h%Y] %3.1M (%Y), %q, %V, %S.\n   %T\n   %u\n
```

CHAPTER

SIX

DEVELOPMENT & CONTRIBUTIONS

This File:

This project is being led by the NANOGrav Astrophysics Working Group. Details on contributions and the mandatory code of conduct can be found in CONTRIBUTING.md.

Contributions are welcome and encouraged, anywhere from new modules/customizations, to bug-fixes, to improved documentation and usage examples. The git workflow is based around a `main` branch which is intended to be (relatively) stable and operational, and an actively developed `dev` branch. New development should be performed in "feature" branches (made off of the `dev` branch), and then incorporated via pull-request (back into the `dev` branch).

For active developers, please install the additional development package requirements:

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

6.1 Formatting

New code should generally abide by PEP8 formatting, with numpy-style docstrings. Exceptions are:

- lines may be broken at either 100 or 120 columns

6.2 Notebooks

Please strip all notebook outputs before commiting notebook changes. The nbstripout package is an excellent option to automatically strip all notebook output only in git commits (i.e. it doesn't change your notebooks in-place). You can also use `nbconvert` to strip output in place:

```
jupyter nbconvert --clear-output --inplace <NOTEBOOK-NAME>.ipynb
```

To install `nbstripout` for the `holodeck` git repository, make sure you're in the `holodeck` root directory and run:

```
pip install --upgrade nbstripout    # install nbstripout
nbstripout --install                # install git hook in current repo only
```

6.3 Test Suite

Before submitting a pull request, please run the test suite on your local machine.

Tests can be run by using `$ pytest` in the root holodeck directory. Tests can also be run against all supported python versions and system configurations by using `$ tox`. `tox` creates anaconda environments for each supported python version, sets up the package and test suite, and then runs `pytest` to execute tests.

Two types of unit-tests are generally used in `holodeck`.

(1) Simple functions and behaviors are included as normal unit-tests, e.g. in "holodeck/tests" and similar directories. These are automatically run by `pytest` and `tox`.

(2) More complex functionality should be tested in notebooks (in "notebooks/") where they can also be used as demonstrations/tutorials for that behavior. Certain notebooks are also converted into unit-test modules to be automatically run by `pytest` and `tox`. The python script scripts/convert_notebook_tests.py converts target notebooks into python scripts in the `holodeck/tests/converted_notebooks` directory, which are then run by `pytest`. The script scripts/holotest.sh will run the conversion script and then run `pytest`. For help and usage information, run `$ scripts/holotest.sh -h`.

CHAPTER

SEVEN

HOLODECK API REFERENCE

holodeck: Massive Black-Hole Binary Population Synthesis & Gravitational Wave Calculations

This package is aimed at providing a comprehensive framework for MBH binary population synthesis. The framework includes modules to perform pop synth using a variety of methodologies to get a handle on both statistical and systematic uncertainties. Currently, binary populations can be synthesis based on: cosmological hydrodynamic simulations (Illustris), semi-analytic/semi-empirical models, and observational catalogs of local galaxies and/or quasars.

See the *README.md* file for more information. The github repository is: https://github.com/nanograv/holodeck. Additional documentation can be found at: holodeck-gw.readthedocs.io/en/docs/index.html. Note that the readthedocs documentation can also be built locally from the *holodeck/docs* folder. A methods paper for *holodeck* is currently in preparation.

In general, *holodeck* calculations proceed in three stages:

(1) **Population**: Construct an initial population of MBH 'binaries'. This is typically done for pairs of MBHs when their galaxies merge (i.e. long before the two MBHs are actually a gravitationally-bound binary). Constructing the initial binary population may occur in a single step: e.g. gathering MBH-MBH encounters from cosmological hydrodynamic simulations; or it may occur over two steps: (i) gathering galaxy-galaxy encounters, and (ii) prescribing MBH properties for each galaxy.

(2) **Evolution**: Evolve the binary population from their initial conditions (i.e. large separations) until coalescence (i.e. small separations). The complexity of this evolutionary stage can range tremendously in complexity. In the simplest models, binaries are assumed to coalesce instantaneously (in that the age of the universe is the same at formation and coalescence), and are assumed to evolve purely due to GW emission (in that the time spent in any range of orbital frequencies can be calculated from the GW hardening timescale). Note that these two assumptions are contradictory.

(3) **Gravitational Waves**: Calculate the resulting GW signals based on the binaries and their evolution. Note that GWs can only be calculated based on some sort of model for binary evolution. The model may be extremely simple, in which case it is sometimes glanced over.

`holodeck.log = <Logger holodeck (DEBUG)>`

global root logger from *holodeck.logger*

CHAPTER

EIGHT

HOLODECK.SAMS (SEMI-ANALYTIC MODELS) MODULE

holodeck - Semi-Analytic Models (SAMs)

The *holodeck* SAMs construct MBH binary populations using simple semi-analytic and/or semi-empirical relationships. For more information, see the *SAMs getting-started guide*.

The core element of holodeck SAMs is the `Semi_Analytic_Model` class. Instances of this class piece together the different components of the model (defined in `holodeck.sams.comps`) to construct a population. The population is evolved with a binary evolution model, implemented as a subclass of the `_Hardening`, typically in the `holodeck.hardening` module.

8.1 holodeck.sams.comps

Semi-Analytic Model - Components

This module provides the key building blocks for the holodeck SAMs. In particular, the:

- Galaxy Stellar-Mass Function (GSMF) : number-density of galaxies as a function of stellar mass;
- Galaxy Merger Rate (GMR) : rate of galaxy mergers per galaxy;
- Galaxy Pair Fraction (GPF) : fraction of galaxy pairs, relative to all galaxies;
- Galaxy Merger Time (GMT) : duration over which galaxy pairs are observable as pairs.

For more information see the `holodeck.sams` module, or the *SAMs getting-started guide*.

References

- [Sesana2008] Sesana, Vecchio, Colacino 2008.
- **[Rodriguez-Gomez2015] Rodriguez-Gomez, Genel, Vogelsberger, et al. 2015**
  The merger rate of galaxies in the Illustris simulation: a comparison with observations and semi-empirical models https://ui.adsabs.harvard.edu/abs/2015MNRAS.449...49R/abstract
- [Chen2019] Chen, Sesana, Conselice 2019.
- **[Leja2020] Leja, Speagle, Johnson, et al. 2020.**
  A New Census of the 0.2 < z < 3.0 Universe. I. The Stellar Mass Function https://ui.adsabs.harvard.edu/abs/2020ApJ...893..111L/abstract

class holodeck.sams.comps.**_Galaxy_Stellar_Mass_Function**(**args*, ***kwargs*)

Galaxy Stellar-Mass Function base-class. Used to calculate number-density of galaxies.

mbh_mass_func(*mbh*, *redz*, *mmbulge*, *scatter=None*)

Convert from the GSMF to a MBH mass function (number density), using a given Mbh-Mbulge relation.

Parameters

- **mbh** (*array_like*) – Blackhole masses at which to evaluate the mass function.
- **redz** (*array_like*) – Redshift(s) at which to evaluate the mass function.
- **mmbulge** (*relations._MMBulge_Relation* subclass instance) – Scaling relation between galaxy and MBH masses.
- **scatter** (*None, bool, or float*) – Introduce scatter in masses. * *None* or *True* : use the value from *mmbulge._scatter_dex* * *False* : do not introduce scatter * float : introduce scatter with this amplitude (in dex)

Returns

ndens – Number density of MBHs, in units of [Mpc^-3]

Return type

array_like

class holodeck.sams.comps.**GSMF_Schechter**(*phi0=-2.77*, *phiz=-0.27*, *mchar0_log10=11.24*, *mchar0=None*, *mcharz=0.0*, *alpha0=-1.24*, *alphaz=-0.03*)

Single Schechter Function - Galaxy Stellar Mass Function.

This is density per unit log10-interval of stellar mass, i.e. $\Phi = dn/d\log_{10}(M)$

See: [Chen2019] Eq.9 and enclosing section.

_phi_func(*redz*)

See: [Chen2019] Eq.9

_mchar_func(*redz*)

See: [Chen2019] Eq.10 - NOTE: added *redz* term

_alpha_func(*redz*)

See: [Chen2019] Eq.11

class holodeck.sams.comps.**_GSMF_Single_Schechter**(*log10_phi_terms*, *log10_mstar_terms*, *alpha*)

Schechter function, with parameters as quadratics with respect to redshift.

Parameterization follows [Leja2020] and is primarily for use in the *GSMF_Double_Schechter* class. From [Leja2020] Eq.14:

$$\frac{\partial n}{\partial \log_{10} M} = \ln(10)\phi \left(\frac{M}{M_\star}\right)^{\alpha+1} \exp[-M/M_\star].$$

The power-law index α is a scalar value, while the reference mass $M_\star$, and the normalization ϕ are defined as quadratics with respect to redshift:

$$\log_{10}(\phi) = a_0 + a_1 z + a_2 z^2,$$
$$\log_{10}(M_\star) = b_0 + b_1 z + b_2 z^2.$$

Class instances are callable, see **__call__()**, and return galaxy number densities in units of $[\mathrm{Mpc}^{-3}\,\mathrm{dex}^{-1}]$.

_log10_phi_terms

these are the a_i terms in the defintion.

_log10_mstar_terms

these are the b_i terms.

_alpha

power-law index

_phi_func(*redz*)

Evaluate the GSMF normalization (phi) at the given redshift(s).

_mstar_func(*redz*)

Evaluate the GSMF characteristic mass ($M_\star$) at the given redshift(s).

class holodeck.sams.comps.**GSMF_Double_Schechter**(*phi1=[-2.383, -0.264, -0.107]*, *phi2=[-2.818, -0.368, 0.046]*, *mstar=[10.767, 0.124, -0.033]*, *alpha1=-0.28*, *alpha2=-1.48*)

Sum of two Schechter functions, each parameterized as quadratics in redshift.

For each Schechter Function (*_GSMF_Single_Schechter*), the normalizations (ϕ) and characteristic masses ($M_\star$) are parameterized as quadratics with respect to redshift.

Each Schechter function is parameterized as,

$$\frac{\partial n}{\partial \log_{10} M} = \ln(10)\,\phi \cdot \left(\frac{M}{M_\star}\right)^{\alpha+1} \exp[-M/M_\star],$$
$$\log_{10}(\phi) = a_0 + a_1 z + a_2 z^2,$$
$$\log_{10}(M_\star) = b_0 + b_1 z + b_2 z^2.$$

The parameters for ϕ and α are different for the two functions, while the parameters for $M_\star$ are shared (i.e. the same characteristic mass is used for both). Default parameters are the best fits from [Leja2020]. With uncertainties, these are:

```
phi_1:  [ -2.383 ± 0.027,  -0.264 ± 0.071,  -0.107 ± 0.030],
phi_2:  [ -2.818 ± 0.050,  -0.368 ± 0.070,  +0.046 ± 0.020],
M_star: [+10.767 ± 0.026,  +0.124 ± 0.045,  -0.033 ± 0.015].
```

class holodeck.sams.comps.**_Galaxy_Merger_Rate**(**args*, ***kwargs*)

Galaxy Merger Rate base class, used to model merger rates of galaxy pairs.

class holodeck.sams.comps.**GMR_Power_Law**(*norm0_log10=None*, *normz=None*, *malpha0=None*, *malphaz=None*, *mdelta0=None*, *mdeltaz=None*, *qgamma0=None*, *qgammaz=None*)

Galaxy Merger Rate - based on multiple power-laws.

See [Rodriguez-Gomez2015], Table 1. "merger rate as a function of descendant stellar mass M_star, progenitor stellar mass ratio mu_star"

_get_qgamma(*redz*, *mtot*)

NOTE: *mtot* is needed as an argument as it is used by subclass *GMR_Illustris(GMR_Power_Law)*

class holodeck.sams.comps.**GMR_Illustris**(*norm0_log10=None*, *normz=None*, *malpha0=None*, *malphaz=None*, *mdelta0=None*, *mdeltaz=None*, *qgamma0=None*, *qgammaz=None*, *qgammam=None*)

Galaxy Merger Rate - based on fits to Illustris cosmological simulations.

See [Rodriguez-Gomez2015], Table 1. "merger rate as a function of descendant stellar mass M_star, progenitor stellar mass ratio mu_star"

_get_qgamma(*redz*, *mtot*)

NOTE: *mtot* is needed as an argument as it is used by subclass *GMR_Illustris(GMR_Power_Law)*

class holodeck.sams.comps.**_Galaxy_Pair_Fraction**(**args*, ***kwargs*)

Galaxy Pair Fraction base class, used to describe the fraction of galaxies in mergers/pairs.

class holodeck.sams.comps.**GPF_Power_Law**(*frac_norm_allq=0.025*, *frac_norm=None*, *mref=None*, *mref_log10=11.0*, *malpha=0.0*, *zbeta=0.8*, *qgamma=0.0*, *obs_conv_qlo=0.25*, *max_frac=1.0*)

Galaxy Pair Fraction - Single Power-Law

class holodeck.sams.comps.**_Galaxy_Merger_Time**(**args*, ***kwargs*)

Galaxy Merger Time base class, used to model merger timescale of galaxy pairs.

zprime(*mass*, *mrat*, *redz*, ***kwargs*)

Return the redshift after merger (i.e. input *redz* delayed by merger time).

class holodeck.sams.comps.**GMT_Power_Law**(*time_norm=1.7356680000000002e+16*, *mref0=1.9884098706980508e+44*, *malpha=0.0*, *zbeta=-0.5*, *qgamma=0.0*)

Galaxy Merger Time - simple power law prescription

8.2 holodeck.sams.sam

Semi Analytic Modeling (SAM) submodule.

The core element of the SAM module is the *Semi_Analytic_Model* class. This class requires four components as arguments:

(1) Galaxy Stellar Mass Function (GSMF): gives the comoving number-density of galaxies as a function of stellar mass. This is implemented as subclasses of the `_Galaxy_Stellar_Mass_Function` base class.

(2) Galaxy Pair Fraction (GPF): gives the fraction of galaxies that are in a 'pair' with a given mass ratio (and typically a function of redshift and primary-galaxy mass). Implemented as subclasses of the `_Galaxy_Pair_Fraction` subclass.

(3) Galaxy Merger Time (GMT): gives the characteristic time duration for galaxy 'mergers' to occur. Implemented as subclasses of the `_Galaxy_Merger_Time` subclass.

(4) M_bh - M_bulge Relation (mmbulge): gives MBH properties for a given galaxy stellar-bulge mass. Implemented as subcalsses of the `holodeck.relations._MMBulge_Relation` subclass.

The *Semi_Analytic_Model* class defines a grid in parameter space of total MBH mass ($M = M_1 + M_2$), MBH mass ratio (q
$equiv M_1/M_2$), redshift (z), and at times binary separation (semi-major axis a) or binary rest-frame orbital-frequency (f_r). Over this grid, the distribution of comoving number-density of MBH binaries in the Universe is calculated. Methods are also provided that interface with the *kalepy* package to draw 'samples' (discretized binaries) from the distribution, and to calculate GW signatures.

The step of going from a number-density of binaries in (M, q, z) space, to also the distribution in a or f is subtle, as it requires modeling the binary evolution (i.e. hardening rate).

8.2.1 To-Do

- Allow SAM class to take M-sigma in addition to M-Mbulge.

References

- [Sesana2008] Sesana, Vecchio, Colacino 2008.
- [Chen2019] Chen, Sesana, Conselice 2019.

holodeck.sams.sam.**REDZ_SAMPLE_VOLUME = True**

get redshifts by sampling uniformly in 3D spatial volume, and converting

holodeck.sams.sam.**GSMF_USES_MTOT = False**

the mass used in the GSMF is interpretted as M=m1+m2, otherwise use primary m1

holodeck.sams.sam.**GPF_USES_MTOT = False**

the mass used in the GPF is interpretted as M=m1+m2, otherwise use primary m1

holodeck.sams.sam.**GMT_USES_MTOT = False**

the mass used in the GMT is interpretted as M=m1+m2, otherwise use primary m1

class holodeck.sams.sam.**Semi_Analytic_Model**(*mtot=(1.988409870698051e+37, 1.988409870698051e+45, 91)*, *mrat=(0.001, 1.0, 81)*, *redz=(0.001, 10.0, 101)*, *shape=None*, *log=None*, *gsmf=<class 'holodeck.sams.comps.GSMF_Schechter'>*, *mmbulge=<class 'holodeck.relations.MMBulge_KH2013'>*, *gpf=None*, *gmt=None*, *gmr=None*, ***kwargs*)

Bases: `object`

Semi-Analytic Model of MBH Binary populations.

Based on four components: * Galaxy Stellar-Mass Function (GSMF): the distribution of galaxy masses * Galaxy Pair Fraction (GPF): the probability of galaxies having a companion * Galaxy Merger Time (GMT): the expected galaxy-merger timescale for a pair of galaxies * M-MBulge relation: relation between host-galaxy (bulge-mass) and MBH (mass) properties

__init__(*mtot=(1.988409870698051e+37, 1.988409870698051e+45, 91)*, *mrat=(0.001, 1.0, 81)*, *redz=(0.001, 10.0, 101)*, *shape=None*, *log=None*, *gsmf=<class 'holodeck.sams.comps.GSMF_Schechter'>*, *mmbulge=<class 'holodeck.relations.MMBulge_KH2013'>*, *gpf=None*, *gmt=None*, *gmr=None*, ***kwargs*)

Construct a new Semi_Analytic_Model instance.

Parameters

- **mtot** (*list, optional*) –
- **mrat** (*list, optional*) –
- **redz** (*list, optional*) –
- **shape** (*_type_, optional*) –
- **gsmf** (*_type_, optional*) –
- **gpf** (*_type_, optional*) –
- **gmt** (*_type_, optional*) –
- **mmbulge** (*_type_, optional*) –

_gsmf

Galaxy Stellar-Mass Function (*_Galaxy_Stellar_Mass_Function* instance)

_mmbulge

Mbh-Mbulge relation (*relations._MMBulge_Relation* instance)

_gpf

Galaxy Pair Fraction (*_Galaxy_Pair_Fraction* instance)

_gmt

Galaxy Merger Time (*_Galaxy_Merger_Time* instance)

_gmr

Galaxy Merger Rate (*_Galaxy_Merger_Rate* instance)

_density

Binary comoving number-density

_shape

Shape of the parameter-space domain (mtot, mrat, redz)

_gmt_time

GMT timescale of galaxy mergers [sec]

_redz_prime

redshift following galaxy merger process

property edges

[*mtot*, *mrat*, *redz*])

Type
The grid edges defining the domain (list of

property shape

Shape of the parameter space domain (number of edges in each dimension), (3,) tuple

mass_stellar()

Calculate stellar masses for each MBH based on the M-MBulge relation.

Returns
masses – Galaxy total stellar masses for all MBH. [0, :] is primary, [1, :] is secondary [grams].

Return type
(2, N) ndarray of scalar,

property static_binary_density

The number-density of binaries at each bin edge, 'd^3 n / [dlog10M dq dz]' in units of [Mpc^-3].

This is calculated once and cached.

Returns
density – Number density of binaries, per unit redshift, mass-ratio, and log10 of mass. Units of [Mpc^-3].

Return type
(M, Q, Z) ndarray

Notes

- This function effectively calculates Eq.21 & 5 of [Chen2019]; or equivalently, Eq. 6 of [Sesana2008].
- Bins which 'merge' after redshift zero are set to zero density (using the *self._gmt* instance).

dynamic_binary_number_at_fobs(*hard*, *fobs_orb*, ***kwargs*)

_dynamic_binary_number_at_fobs_consistent(*hard*, *fobs_orb*, *steps=200*, *details=False*)

Get correct redshifts for full binary-number calculation.

Slower but more correct than old *dynamic_binary_number*. Same as new cython implementation *sam_cyutils.dynamic_binary_number_at_fobs*, which is more than 10x faster. LZK 2023-05-11

BUG doesn't work for Fixed_Time_2PL

_dynamic_binary_number_at_fobs_inconsistent(*hard*, *fobs_orb*)

_dynamic_binary_number_at_sepa_consistent(*hard*, *target_sepa*, *steps=200*, *details=False*)

Get correct redshifts for full binary-number calculation.

Slower but more correct than old *dynamic_binary_number*. Same as new cython implementation *sam_cyutils.dynamic_binary_number_at_fobs*, which is more than 10x faster. LZK 2023-05-11

gwb_new(*fobs_gw_edges*, *hard=<holodeck.hardening.Hard_GW object>*, *realize=100*)

Calculate GWB using new cython implementation, 10x faster!

gwb_old(*fobs_gw_edges*, *hard=<class 'holodeck.hardening.Hard_GW'>*, *realize=100*)

Calculate GWB using new *dynamic_binary_number_at_fobs* method, better, but slower.

gwb_ideal(*fobs_gw*, *sum=True*, *redz_prime=None*)

Calculate the idealized, continuous GWB amplitude.

Calculation follows [Phinney2001] (Eq.5) or equivalently [Enoki+Nagashima-2007] (Eq.3.6). This calculation assumes a smooth, continuous population of binaries that are purely GW driven. * There are no finite-number effects. * There are no environmental or non-GW driven evolution effects. * There is no coalescence of binaries cutting them off at high-frequencies.

gwb(*fobs_gw_edges*, *hard=<holodeck.hardening.Hard_GW object>*, *realize=100*, *loudest=1*, *params=False*)

Calculate the (smooth/semi-analytic) GWB and CWs at the given observed GW-frequencies.

Parameters

- **fobs_gw_edges** (*(F,) array_like of scalar,*) – Observer-frame GW-frequencies. [1/sec] These are the frequency bin edges, which are integrated across to get the number of binaries in each frequency bin.
- **hard** (*holodeck.evolution._Hardening class or instance*) – Hardening mechanism to apply over the range of *fobs_gw*.
- **realize** (*int*) – Specification of how many discrete realizations to construct. Realizations approximate the finite-source effects of a realistic population.
- **loudest** (*int*) – Number of loudest single sources to distinguish from the background.
- **params** (*Boolean*) – Whether or not to return astrophysical parameters of the binaries.

Returns

- **hc_ss** (*(F, R, L) NDarray of scalars*) – The characteristic strain of the L loudest single sources at each frequency.
- **hc_bg** (*(F, R) NDarray of scalars*) – Characteristic strain of the GWB.

- **sspar** (*(4, F, R, L) NDarray of scalars*) – Astrophysical parametes (total mass, mass ratio, initial redshift, final redshift) of each loud single sources, for each frequency and realization. Returned only if params = True.
- **bgpar** (*(7, F, R) NDarray of scalars*) – Average effective binary astrophysical parameters (total mass, mass ratio, initial redshift, final redshift, final comoving distance, final separation, final angular separation) for background sources at each frequency and realization, Returned only if params = True.

rate_chirps(*hard=None*, *integrate=True*)

Find the event rate of binary coalescences ('chirps').

Get the number of coalescence events per unit time, in units of [1/sec].

Parameters

- **hard** (None or *_Hardening* subclass instance) –
- **integrate** (*bool*) –

Returns

- **redz_final** (*(M, Q, Z)*) – Redshift of binary coalescence. Binaries stalling before *z=0*, have values set to *-1.0*.
- **rate** (*ndarray*) – Rate of coalescence events in each bin, in units of [1/sec]. The shape and meaning depends on the value of the *integrate* flag:
  - if *integrate* == *True*, then the returned values is `dN/dt`, with shape (M-1, Q-1, Z-1)
  - if *integrate* == *False*, then the returned values is `dN/[dlog10M dq dz dt]`, with shape (M, Q, Z)

_integrate_event_rate(*rate*)

_ndens_gal(*mass_gal*, *mrat_gal*, *redz*)

_ndens_mbh(*mass_gal*, *mrat_gal*, *redz*)

_integrated_binary_density(*ndens=None*, *sum=True*)

dynamic_binary_number(**args*, ***kwargs*)

new_gwb(**args*, ***kwargs*)

holodeck.sams.sam.**sample_sam_with_hardening**(*sam*, *hard*, *fobs_orb=None*, *sepa=None*, *sample_threshold=10.0*, *cut_below_mass=None*, *limit_merger_time=None*, ***sample_kwargs*)

Discretize Semi-Analytic Model into sampled binaries assuming the given binary hardening rate.

Parameters

- **sam** (*Semi_Analytic_Model*) – Instance of an initialized semi-analytic model.
- **hard** (*holodeck.evolution._Hardening*) – Binary hardening model for calculating binary hardening rates (dadt or dfdt).
- **fobs_orb** (*ArrayLike*) – Observer-frame orbital-frequencies. Units of [1/sec]. NOTE: Either *fobs_orb* or *sepa* must be provided, and not both.
- **sepa** (*ArrayLike*) – Binary orbital separation. Units of [cm]. NOTE: Either *fobs_orb* or *sepa* must be provided, and not both.

Returns

- **vals** (*(4, S) ndarray of scalar*) – Parameters of sampled binaries. Four parameters are: * mtot : total mass of binary (m1+m2) in [grams] * mrat : mass ratio of binary (m2/m1 <= 1) * redz : redshift of binary * fobs_orb / sepa : observer-frame orbital-frequency [1/s] or binary separation [cm]
- **weights** (*(S,) ndarray of scalar*) – Weights of each sample point.
- **edges** (*(4,) of list of scalars*) – Edges of parameter-space grid for each of above parameters (mtot, mrat, redz, fobs_orb) The lengths of each list will be [(M,), (Q,), (Z,), (F,)]
- **dnum** (*(M, Q, Z, F) ndarray of scalar*) – Number-density of binaries over grid specified by *edges*.

holodeck.sams.sam.**evolve_eccen_uniform_single**(*sam*, *eccen_init*, *sepa_init*, *nsteps*)

Evolve binary eccentricity from an initial value along a range of separations.

Parameters

- **sam** (*holodeck.sam.Semi_Analytic_Model* instance) – The input semi-analytic model. All this does is provide the range of total-masses to determine the minimum ISCO radius, which then determines the smallest separations to evolve until.
- **eccen_init** (*float,*) – Initial eccentricity of binaries at the given initial separation *sepa_init*. Must be between [0.0, 1.0).
- **sepa_init** (*float,*) – Initial binary separation at which evolution begins. Units of [cm].
- **nsteps** (*int,*) – Number of (log-spaced) steps in separation between the initial separation *sepa_init*, and the final separation which is determined as the minimum ISCO radius based on the smallest total-mass of binaries in the *sam* instance.

Returns

- **sepa** (*(E,) ndarray of float*) – The separations at which the eccentricity evolution is defined over. This is the independent variable of the evolution. The shape *E* is the value of the *nsteps* parameter.
- **eccen** (*(E,)*) – The eccentricity of the binaries at each location in separation given by *sepa*. The shape *E* is the value of the *nsteps* parameter.

holodeck.sams.sam.**add_scatter_to_masses**(*mtot*, *mrat*, *dens*, *scatter*, *refine=4*, *log=None*)

Add the given scatter to masses m1 and m2, for the given distribution of binaries.

The procedure is as follows (see *dev-notebooks/sam-ndens-scatter.ipynb*):

- (1) The density is first interpolated to a uniform, regular grid in (m1, m2) space. A 2nd order interpolant is used first. A 0th-order interpolant is used to fill-in bad values. In-between, a 1st-order interpolant is used if *linear_interp_backup* is True.
- (2) The density distribution is convolved with a smoothing function along each axis (m1, m2) to account for scatter.
- (3) The new density distribution is interpolated back to the original (mtot, mrat) grid.

Parameters

- **mtot** (*(M,) ndarray*) – Total masses in grams.
- **mrat** (*(Q,) ndarray*) – Mass ratios.
- **dens** (*(M, Q) ndarray*) – Density of binaries over the given mtot and mrat domain.
- **scatter** (*float*) – Amount of scatter in the M-MBulge relationship, in dex (i.e. over log10 of masses).

- **refine** (*int,*) – The increased density of grid-points used in the intermediate (m1, m2) domain, in step (1).
- **linear_interp_backup** (*bool,*) – Whether a linear interpolant is used to fill-in bad values after the 2nd order interpolant. This generally doesn't seem to fix any values.
- **logspace_interp** (*bool,*) – Whether interpolation should be performed in the log-space of masses. NOTE: strongly recommended.

Returns
m1m2_dens – Binary density with scatter introduced.

Return type
(M, Q) ndarray,

8.3 holodeck.sams.sam_cyutils

$ python setup.py build_ext -i

$ python setup.py develop

holodeck.sams.sam_cyutils.**hard_func_2pwl_gw**()

NOTE: this function will be somewhat slow, because of the explicit broadcasting!

holodeck.sams.sam_cyutils.**integrate_differential_number_3dx1d**()

Integrate the differential number of binaries over each grid bin into total numbers of binaries.

Trapezoid used over first 3 dims (mtot, mrat, redz), and Riemann over 4th (freq). (Riemann seemed empirically to be more accurate for freq, but this should be revisited.) mtot is integrated over *log10(mtot)* and frequency is integrated over *ln(f)*.

Note on array shapes:

- input *dnum* is shaped (M, Q, Z, F)
- input *edges* must be (4,) of array_like of lengths: M, Q, Z, F+1
- output *numb* is shaped (M-1, Q-1, Z-1, F)

Parameters

- **edges** (*(4,) array_like w/ lengths M, Q, Z, F+1*) – Grid edges of *mtot*, *mrat*, *redz*, and *freq*. NOTE:
  - *mtot* should be passed as regular *mtot*, NOT log10(mtot)
  - *freq* should be passed as regular *freq*, NOT ln(freq)
- **dnum** (*(M, Q, Z, F)*) – Differential number of binaries, dN/[dlog10M dq qz dlnf] where 'N' is in units of dimensionless number.

Returns
numb

Return type
(M-1, Q-1, Z-1, F)

CHAPTER

NINE

HOLODECK.LIBRARIAN MODULE

Module to generate and manage holodeck libraries.

Holodeck 'libraries' are collections of simulations in which a certain set of parameters are varied, producing different populations and/or GW signatures at each sampled parameter value. Libraries are run from the same parameter-space and using the same hyper parameters. Libraries are constructed using a 'parameter space' class that organizes the different simulations. The base-class is *_Param_Space* (defined in the *holodeck.librarian.params* file). The parameter-space subclasses are given a number of different parameters to be varied. Each parameter is implemented as a subclass of *_Param_Dist*, for example the *PD_Uniform* class that implements a uniform distribution.

For more information, see the *'libraries' page in the getting-started guide*.

holodeck.librarian.**DEF_NUM_REALS = 100**

Default number of realizations to construct in libraries.

holodeck.librarian.**DEF_NUM_FBINS = 40**

Default number of frequency bins at which to calculate GW signals.

holodeck.librarian.**DEF_NUM_LOUDEST = 5**

Default number of loudest binaries to calculate in each frequency bin.

holodeck.librarian.**DEF_PTA_DUR = 16.03**

Default PTA duration which determines Nyquist frequency bins [yrs].

holodeck.librarian.**param_spaces_dict = {'PS_Classic_GWOnly_Astro_Extended': <class 'holodeck.librarian.param_spaces_classic.PS_Classic_GWOnly_Astro_Extended'>, 'PS_Classic_GWOnly_Uniform': <class 'holodeck.librarian.param_spaces_classic.PS_Classic_GWOnly_Uniform'>, 'PS_Classic_Phenom_Astro_Extended': <class 'holodeck.librarian.param_spaces_classic.PS_Classic_Phenom_Astro_Extended'>, 'PS_Classic_Phenom_Uniform': <class 'holodeck.librarian.param_spaces_classic.PS_Classic_Phenom_Uniform'>, 'PS_New_Test': <class 'holodeck.librarian.param_spaces.PS_Double_Schechter_Rate'>, 'PS_Test': <class 'holodeck.librarian.param_spaces_classic.PS_Test'>}**

Registry of standard parameter-spaces

9.1 holodeck.librarian.combine

holodeck.librarian.combine.**sam_lib_combine**(*path_output*, *log*, *path_pspace=None*, *recreate=False*, *gwb_only=False*)

Parameters

- **path_output** (*str or Path,*) – Path to output directory where combined library will be saved.
- **log** (*logging.Logger*) – Logging instance.
- **path_sims** (*str or None,*) – Path to output directory containing simulation files. If *None* this is set to be the same as *path_output*.
- **path_pspace** (*str or None,*) – Path to file containing _Param_Space subclass instance. If *None* then *path_output* is searched for a *_Param_Space* save file.

Returns

lib_path – Path to library output filename (typically ending with 'sam_lib.hdf5').

Return type

Path,

9.2 holodeck.librarian.fit_spectra

holodeck.librarian.fit_spectra.**fit_all_libraries_in_path**(*path*, *log*, *pattern=None*, *recreate=False*)

Recursively find all *sam_lib.hdf5* files in the given path, and construct spectra fits for them.

holodeck.librarian.fit_spectra.**fit_library_spectra**(*library_path*, *log*, *recreate=False*)

Calculate line fits to library spectra using MPI.

9.3 holodeck.librarian.gen_lib

Library generation interface.

This file can be run from the command-line to generate holodeck libraries, and also provides some API methods for quick/easy generation of simulations. In general, these methods are designed to run simulations for populations constructed from parameter-spaces (i.e. *_Param_Space* subclasses).

This script can be run by executing:

```
python -m holodeck.librarian.gen_lib <ARGS>
```

Run `python -m holodeck.librarian.gen_lib -h` for usage information.

holodeck.librarian.gen_lib.**main**()

Parent method for generating libraries from the command-line.

This function requires `mpi4py` for parallelization.

This method does the following:

(1) Loads arguments from the command-line (`args`, via *_setup_argparse()*).

(a) The `output` directory is created as needed, along with the `sims/` and `logs/` subdirectories in which the simulation datafiles and log output files are saved.

(2) Sets up a `logging.Logger` instance for each processor.

(3) Constructs the parameter space specified by `args.param_space`. If this run is being resumed, then the param-space is loaded from an existing save file in the output directory.

(4) Samples from the parameter space are allocated to all processors.

(5) Each processor iterates over it's allocated parameters, calculates populations, saves them to files in the output directories. This is handled in *run_sam_at_pspace_num()*.

(6) All of the individual simulation files are combined using *holodeck.librarian.combine.sam_lib_combine()*.

Parameters
None –

Return type
None

holodeck.librarian.gen_lib.**run_sam_at_pspace_num**(*args*, *space*, *pnum*)

Run a given simulation (index number `pnum`) in the `space` parameter-space.

This function performs the following:

(1) Constructs the appropriate filename for this simulation, and checks if it already exist. If the file exists and the `args.recreate` option is not specified, the function returns `True`, otherwise the function runs this simulation.

(2) Calls `space.model_for_sample_number` to generate the semi-analytic model and hardening instances; see the `holodeck.librarian.params._Param_Space.model_for_sample_number()` function.

(3) Calculates populations and GW signatures from the SAM and hardening model using *run_model()*, and saves the results to an output file.

(4) Optionally: some diagnostic plots are created in the *make_plots()* function.

Parameters

- **args** (`argparse.ArgumentParser` instance) – Arguments from the *gen_lib_sams.py* script. NOTE: this should be improved.
- **space** (`holodeck.librarian.params._Param_space` instance) – Parameter space from which to construct populations.
- **pnum** (*int*) – Which parameter-sample from `space` should be run.

Returns

- **rv** (*bool*) – `True` if this simulation was successfully run, `False` otherwise.
- **sim_fname** (`pathlib.Path` instance) – Path of the simulation save file.

holodeck.librarian.gen_lib.**run_model**(*sam*, *hard*, *pta_dur=16.03*, *nfreqs=40*, *nreals=100*, *nloudest=5*, *gwb_flag=True*, *details_flag=False*, *singles_flag=False*, *params_flag=False*, *log=None*)

Run the given SAM and hardening model to construct a binary population and GW signatures.

Parameters

- **sam** (*holodeck.sams.sam.Semi_Analytic_Model* instance,) –

- **hard** (holodeck.hardening._Hardening subclass instance,) –
- **pta_dur** (*float, [seconds]*) – Duration of PTA observations in seconds, used to determine Nyquist frequency basis at which GW signatures are calculated.
- **nfreqs** (*int*) – Number of Nyquist frequency bins at which to calculate GW signatures.
- **nreals** (*int*) – Number of 'realizations' (populations drawn from Poisson distributions) to construct.
- **nloudest** (*int*) – Number of loudest binaries to consider in each frequency bin. These are the highest GW strain binaries in each frequency bin, for which the individual source strains are calculated.
- **gwb_flag** –
- **details_flag** –
- **singles_flag** –
- **params_flag** –
- **log** (logging.Logger instance) –

Returns

data – The population and GW data calculated from the simulation. The dictionary elements are:

- fobs_cents : Nyquist frequency bin centers, in units of [seconds].
- fobs_edges : Nyquist frequency bin edgeds, in units of [seconds].
- If details_flag == True:
  - static_binary_density :
  - number :
  - redz_final :
  - gwb_params :
  - num_params :
  - gwb_mtot_redz_final :
  - num_mtot_redz_final :
- If params_flag == True:
  - sspar :
  - bgpar :
- If singles_flag == True:
  - hc_ss :
  - hc_bg :
- If gwb_flag == True:
  - gwb :

Return type
dict

holodeck.librarian.gen_lib.**_calc_model_details**(*edges*, *redz_final*, *number*)

Calculate derived properties from the given populations.

Parameters

- **edges** (*(4,) list of 1darrays*) – [mtot, mrat, redz, fobs_orb_edges] with shapes (M, Q, Z, F+1)
- **redz_final** (*(M,Q,Z,F)*) – Redshift final (redshift at the given frequencies).
- **number** (*(M-1, Q-1, Z-1, F)*) – Absolute number of binaries in the given bin (dimensionless).

Returns

- *gwb_pars*
- *num_pars*
- *gwb_mtot_redz_final*
- *num_mtot_redz_final*

holodeck.librarian.gen_lib.**_setup_argparse**(**args*, ***kwargs*)

Setup the argument-parser for command-line usage.

Parameters

- ***args** (*arguments*) –
- ****kwargs** (*keyword arguments*) –

holodeck.librarian.gen_lib.**_setup_log**(*comm*, *args*)

Setup up the logging module logger for output messaging.

9.3.1 Arguemnts

comm args

returns
log

rtype
logging.Logger instance

holodeck.librarian.gen_lib.**make_plots**(*args*, *data*, *sim_fname*)

Generate diagnostic plots from the given simulation data and save to file.

holodeck.librarian.gen_lib.**make_gwb_plot**(*fobs*, *gwb*, *fit_data*)

Generate a GWB plot from the given data.

holodeck.librarian.gen_lib.**make_ss_plot**(*fobs*, *hc_ss*, *hc_bg*, *fit_data*)

holodeck.librarian.gen_lib.**make_pars_plot**(*fobs*, *hc_ss*, *hc_bg*, *sspar*, *bgpar*)

Plot total mass, mass ratio, initial d_c, final d_c

9.4 holodeck.librarian.lib_utils

holodeck.librarian.lib_utils.**get_fits_path**(*library_path*)

Get the name of the spectral fits file, given a library file path.

holodeck.librarian.lib_utils.**load_pspace_from_path**(*log*, *path*, *space_class=None*)

Load a *_Param_Space* instance from the saved file in the given directory.

Parameters

- **path** (*str or pathlib.Path*) – Path to directory containing save file. A single file matching **.pspace.npz* is required in that directory. NOTE: the specific glob pattern is specified by *holodeck.librarian.PSPACE_FILE_SUFFIX* e.g. '.pspace.npz'
- **space_class** (*_Param_Space* subclass) – Class with which to call the *from_save()* method to load a new *_Param_Space* instance.

Returns

- **log** (*logging.Logger*)
- **space** (*_Param_Space* subclass instance) – An instance of the *space_class* class.
- **space_fname** (*pathlib.Path*) – File that *space* was loaded from.

9.5 holodeck.librarian.param_spaces

Parameter-Space definitions for holodeck libraries.

class holodeck.librarian.param_spaces.**PS_Double_Schechter_Rate**(*log*, *nsamples=None*, *sam_shape=None*, *seed=None*)

9.6 holodeck.librarian.param_spaces_classic

'Classic' parameter spaces used in the NANOGrav 15yr analysis.

class holodeck.librarian.param_spaces_classic.**PS_Classic_GWOnly_Astro_Extended**(*log*, *nsamples=None*, *sam_shape=None*, *seed=None*)

Classic 10D GW-Only, uniform parameter space used in 15yr analysis.

Previously called the *PS_New_Astro_02_GW* parameter space, or 'gw-only+extended'.

class holodeck.librarian.param_spaces_classic.**PS_Classic_GWOnly_Uniform**(*log*, *nsamples=None*, *sam_shape=None*, *seed=None*)

Classic 4D GW-Only, uniform parameter space used in 15yr analysis.

Previously called the *PS_Uniform_07_GW* parameter space, or 'gw-only'.

class holodeck.librarian.param_spaces_classic.**PS_Classic_Phenom_Astro_Extended**(*log*, *nsamples=None*, *sam_shape=None*, *seed=None*)

Classic 12D phenomenological, uniform parameter space used in 15yr analysis.

Previously called the *PS_New_Astro_02B* parameter space, or 'phenom-astro+extended'.

class holodeck.librarian.param_spaces_classic.**PS_Classic_Phenom_Uniform**(*log*, *nsamples=None*, *sam_shape=None*, *seed=None*)

Classic 5D phenomenological, uniform parameter space used in 15yr analysis.

Previously called the *PS_Uniform_09B* parameter space, or 'phenom-uniform'.

class holodeck.librarian.param_spaces_classic.**PS_Test**(*log*, *nsamples=None*, *sam_shape=None*, *seed=None*)

Simple test parameter space in 2D.

9.7 holodeck.librarian.params

Parameters and parameter spaces for holodeck libraries.

class holodeck.librarian.params.**PD_Lin_Log**(*lo*, *hi*, *crit*, *lofrac*, ***kwargs*)

class holodeck.librarian.params.**PD_Log_Lin**(*lo*, *hi*, *crit*, *lofrac*, ***kwargs*)

class holodeck.librarian.params.**PD_Normal**(*mean*, *stdev*, *clip=None*, ***kwargs*)

NOTE: use *clip* parameter to avoid extreme values.

class holodeck.librarian.params.**PD_Piecewise_Uniform_Density**(*edges*, *densities*, ***kwargs*)

class holodeck.librarian.params.**PD_Piecewise_Uniform_Mass**(*edges*, *weights*, ***kwargs*)

class holodeck.librarian.params.**PD_Uniform**(*lo*, *hi*, ***kwargs*)

class holodeck.librarian.params.**PD_Uniform_Log**(*lo*, *hi*, ***kwargs*)

class holodeck.librarian.params.**_Param_Dist**(*clip=None*)

Parameter Distribution base-class for use in Latin HyperCube sampling.

These classes are passed uniform random variables between [0.0, 1.0], and return parameters from the desired distribution.

Subclasses are required to implement the `_dist_func()` function which accepts a float value from [0.0, 1.0] and returns the appropriate corresponding parameter, drawn from the desired distribution. In practice, `_dist_func()` is usually the inverse cumulative-distribution for the desired distribution function.

class holodeck.librarian.params.**_Param_Space**(*log*, *nsamples=None*, *sam_shape=None*, *seed=None*, *random_state=None*, ***param_kwargs*)

Base class for generating holodeck libraries. Defines the parameter space and settings.

Libraries are generated over some parameter space defined by which parameters are being varied.

_normalized_params(*vals*)

Convert input values (uniform/linear) into parameters from the stored distributions.

For example, if this parameter space has 2 dimensions, where the distributions are:

0. 'value_a' is a uniform parameter from [-1.0, 1.0], and
1. 'value_b' normal with mean 10.0 and stdev 1.0

Then input values of `[0.75, 0.5]` are mapped to parameters `[0.5, 10.0]`, which will be returned as `{value_a: 0.5, value_b: 10.0}`.

Parameters
vals (*(P,) iterable of float,*) – A list/iterable of *P* float values, matching the number of parameters (i.e. dimensions) in this parameter space. Each value is passed to the corresponding distribution for that parameter.

Returns
params – The resulting parameters in the form of key-value pairs where the keys are the parameter names, and the values are drawn from the correspinding distributions.

Return type
dict,

classmethod from_save(*fname*, *log*)

Create a new _Param_Space instance loaded from the given file.

Parameters
fname (*str*) – Filename containing parameter-space save information, generated form *_Param_Space.save*.

Returns
space

Return type
_Param_Space instance

classmethod model_for_params(*params*, *sam_shape=None*)

Construct a model (SAM and hardening instances) from the given parameters.

Parameters

- **params** (*dict*) – Key-value pairs for sam/hardening parameters. Each item much match expected parameters that are set in the *defaults* dictionary.
- **sam_shape** (*None or int or (3,) int*) –

Returns

- **sam** (`holodeck.sam.Semi_Analytic_Model` instance)
- **hard** (`holodeck.hardening._Hardening` instance)

save(*path_output*)

Save the generated samples and parameter-space info from this instance to an output file.

This data can then be loaded using the *_Param_Space.from_save* method.

Parameters
path_output (*str*) – Path in which to save file. This must be an existing directory.

Returns
fname – Output path including filename in which this parameter-space was saved.

Return type
str

9.8 holodeck.librarian.posterior_populations

Script to generate Holodeck populations, drawing from the 15yr analysis constraints.

See *holodeck-pops.ipynb* for example usage of this data.

9.8.1 Usage

See *python gen_holodeck_pop.py -h* for usage information.

Example:

```
python gen_holodeck_pops.py -t 20 -f 30 -r 100 -l 10 -m
                                |     |     |      |   |--> use maxmimum-likelihood values
                                |     |     |      |------> 10 loudest binaries in each␣
↪frequency bin
                                |     |     |--------------> 100 realizations of␣
↪populations
                                |     |--------------------> 30 frequency bins
                                |--------------------------> 20 years observing baseline =␣
↪1/(20yr) lowest frequency
```

9.8.2 To-Do

- Improve handling of data path.
- **Improve handling/specification of parameter space.**
  - Allow changes to be passed in through API and or CL
  - Make each particular 15yr dataset specify its own parameter space (these need to match up anyway!)

holodeck.librarian.posterior_populations.**get_maxlike_pars_from_chains**(*chains=None*)

Load the maximum-likelihood (ML) parameters from the given chains (i.e. parameter posteriors).

KDEs from *kalepy* are used to construct the ML parameters.

Parameters

chains (*dict*) – The MCMC parameter values for each of the parameters in this holodeck parameter-space. These chains should typically be loaded using the *load_chains* function.

Returns

pars – Maximum likelihood parameters drawn from the *chains*. This will be a single float value for each of the parameters in the holodeck parameter-space, for example:

```
['hard_time', 'gsmf_phi0', 'gsmf_mchar0_log10',
'mmb_mamp_log10', 'mmb_scatter_dex', 'hard_gamma_inner']
```

Return type

dict

holodeck.librarian.posterior_populations.**load_chains**(*path_data*)

Load the MCMC chains from the given path.

The path must contain the expected files resulting from fitting with *ceffyl*.

Parameters

path_data (*str* or *pathlib.Path*) – Path to directory containing the *pars.txt* and *chain_1.0.txt* files.

Returns

data – The values at each step of the MCMC chains for each parameters. For example, the parameters may be:

```
['hard_time', 'gsmf_phi0', 'gsmf_mchar0_log10',
'mmb_mamp_log10', 'mmb_scatter_dex', 'hard_gamma_inner']
```

in which case each of these will be an entry in the dictionary, where the values are an array of the steps in each of these parameters.

Return type

dict

holodeck.librarian.posterior_populations.**load_population_for_pars**(*pars*, *pta_dur=16.0*, *nfreqs=60*, *nreals=103*, *nloudest=10*)

Construct a holodeck population.

Parameters

- **pars** (*dict*) – Binary population parameters for the appropriate parameter space *PSPACE*. Typically the *pars* should be loaded using either the *sample_pars_from_chains* or the *get_maxlike_pars_from_chains* function.
- **pta_dur** (*scalar [seconds]*) – Duration of PTA observations, used to determine Fourier frequency bins. Bin centers are at frequencies `f_i = (i+1) / pta_dur`
- **nfreqs** (*int*) – Number of frequency bins.
- **nreals** (*int*) – Number of realizations to construct.
- **nloudest** (*int*) – Number of loudest binaries to calculate, per frequency bin.

Returns

data – Binary population and derived properties. Entries:

- *number* : ndarray (M, Q, Z, F) Number of binaries in the Universe in each bin. The bins are total mass (M), mass ratio (Q), redshift (Z), and frequency (F).
- *hc_ss* : ndarray (F, R, L) GW characteristic strain of the loudest L binaries in each frequency bin (F) and realization (R). The GW frequencies are assumed to be 2x the orbital frequencies (i.e. circular orbits).
- *hc_bg* : ndarray (F, R) GW characteristic strain of all binaries besides the L loudest in each frequency bin, for frequency bins *F* and realizations *R*. The GW frequencies are assumed to be 2x the orbital frequencies (i.e. circular orbits).
- *sspar* : ndarray (P, F, R, L) Binary parameters of the loudest *L* binaries in each frequency bin *F* for realizations *R*. The P=4 parameters included are {total mass [grams], mass ratio, initial redshift, final redshift}, where initial redshift is at the time of galaxy merger, and final redshift is when reaching the frequency bin.
- *mtot_edges* : ndarray (M+1,) The edges of the total-mass dimension of the SAM grid, in units of [grams]. Note that there are *M+1* bin edges for *M* bins.
- *mrat_edges* : ndarray (Q+1,) The edges of the mass-ratio dimension of the SAM grid. Note that there are *Q+1* bin edges for *Q* bins.

- *redz_edges* : ndarray (Z+1,) The edges of the redshfit dimension of the SAM grid. Note that there are *Z+1* bin edges for *Z* bins.
- *fobs_orb_edges* : ndarray (F+1,) The edges of the orbital-frequency dimension of the SAM grid. Note that there are *F+1* bin edges for *F* bins.

Return type
dict

holodeck.librarian.posterior_populations.**main**(*args=None*)

Top level function that does all the work.

holodeck.librarian.posterior_populations.**sample_pars_from_chains**(*chains=None*)

Sample randomly from the given chains (i.e. parameter posteriors).

Parameters
chains (*dict*) – The MCMC parameter values for each of the parameters in this holodeck parameter-space. These chains should typically be loaded using the *load_chains* function.

Returns

pars – Randomly selected parameters drawn from the *chains*. This will be a single float value for each of the parameters in the holodeck parameter-space, for example:

```
['hard_time', 'gsmf_phi0', 'gsmf_mchar0_log10',
'mmb_mamp_log10', 'mmb_scatter_dex', 'hard_gamma_inner'],
```

Return type
dict

holodeck.librarian.posterior_populations.**setup_argparse**(**args*, ***kwargs*)

Setup parameters/arguments.

Note that this can be used to set parameters NOT from command-line usage, but in this case the *args* argument must be set to empty. For example:

This will load of all the default arguments (NOTE the empty string argument is typically needed):
`args = gen_holodeck_pops.setup_argparse("")`

This will set the desired parameters, and otherwise load the defaults:
`args = gen_holodeck_pops.setup_argparse("", nloudest=12, nreals=6, maxlike=True)`

CHAPTER

TEN

HOLODECK.ACCRETION

Massive Black Hole Binary (MBHB) accretion models to evolve individual Massive Black Hole (MBH) masses using the illustris accretion rates.

10.1 Authors

Magdalena Siwek

class holodeck.accretion.**Accretion**(*accmod='Basic'*, *f_edd=0.01*, *mdot_ext=None*, *eccen=0.0*, *subpc=True*, ***kwargs*)

Preferential Accretion prescription

accmod

Type
{'Basic', 'Proportional', 'Primary', 'Secondary', 'Siwek22', 'Duffell'}, optional

f_edd

Type
double, optional

mdot_ext

Type
Any, optional

eccen

Type
float, optional

subpc

Type
boolean, optional

:meth:`mdot_eddington(mass)`

Calculate the total accretion rate based on masses and a fraction of the Eddington limit.

:meth:`pref_acc(mdot, evol, step)`

Contains a variety of accretion models to choose from to calculate primary vs secondary accretion rates.

mdot_eddington(*mass*, *eps=0.1*)

Calculate the total accretion rate based on masses and a fraction of the Eddington limit.

mass : float eps : float, optional

Radiative efficiency epsilon. Defaults to 0.1.

medd : float

holodeck.constants : constants used for calculation of the accretion rate.

The limiting Eddington accretion rate is defined as: .. math:: `dot{M}_{mathsf{Edd}} = rac{4 pi GM m_p}{epsilon c sigma_{mathsf{T}}}`

```
>>> acc = Accretion()
>>> mass =
>>> print(acc.mdot_eddington(mass))
```

pref_acc(*mdot*, *evol*, *step*)

Contains a variety of accretion models to choose from to calculate primary vs secondary accretion rates.

The accretion models are as follows: * Basic * Primary * Secondary * Proportional * Siwek22 * Duffell

Parameters

- **mdot** – Gas inflow rate in solar masses. Units of [M/year]
- **evol** – evolution class instance which contains the eccentricities of the current evolution
- **step** (*int*) – current timestep

Returns

mdot_arr – Array of accretion rates for each binary in the timestep.

Return type

ndarray

See also:

`Evolution._take_next_step()` : Relationship

Notes

The instance of the evolution class must also be suppled in case eccentricities need to be accessed.

CHAPTER

ELEVEN

HOLODECK.CONSTANTS

Numerical Constants

All constants are used in CGS units, as raw floats. Most of the holodeck package works in CGS units whenever possible. Constants and units should only be added when they are frequently used (i.e. in multiple files/submodules).

Notes

- [cm] = centimeter
- [g] = gram
- [s] = second
- [erg] = cm^2 * g / s^2
- [Jy] = jansky = [erg/s/cm^2/Hz]
- [fr] franklin = statcoulomb = electro-static unit [esu]
- [K] Kelvin

holodeck.constants.**ARCSEC = 4.84813681109536e-06**
: arcsecond in radians []

holodeck.constants.**AU = 14959787070000.0**
: Astronomical Unit [cm]

holodeck.constants.**DAY = 86400.0**
: Day [s]

holodeck.constants.**EDDT = 63219.620781981204**
: Eddington Luminosity prefactor factor [erg/s/g]

holodeck.constants.**EVOLT = 1.6021766339999997e-12**
: Electronvolt in ergs

holodeck.constants.**GPC = 3.085677581491367e+27**
: Giga-parsec [cm]

holodeck.constants.**GYR = 3.15576e+16**
: Giga-year [s]

holodeck.constants.**HPLANCK = 6.62607015e-27**
: Planck constant [erg/s]

holodeck.constants.**JY = 1e-23**
Jansky [erg/s/cm^2/Hz]

holodeck.constants.**KBOLTZ = 1.380649e-16**
Boltzmann constant [erg/K]

holodeck.constants.**KMPERSEC = 100000.0**
km/s [cm/s]

holodeck.constants.**KPC = 3.0856775814913673e+21**
Kilo-parsec [cm]

holodeck.constants.**LSOL = 3.828e+33**
Solar Luminosity [erg/s]

holodeck.constants.**MELC = 9.1093837015e-28**
Electron Mass [g]

holodeck.constants.**MPC = 3.085677581491367e+24**
Mega-parsec [cm]

holodeck.constants.**MPRT = 1.67262192369e-24**
Proton Mass [g]

holodeck.constants.**MSOL = 1.988409870698051e+33**
Solar Mass [g]

holodeck.constants.**MYR = 31557600000000.0**
Mega-year [s]

holodeck.constants.**NWTG = 6.674299999999999e-08**
Newton's Gravitational Constant [cm^3/g/s^2]

holodeck.constants.**PC = 3.085677581491367e+18**
Parsec [cm]

holodeck.constants.**QELC = 4.803204712570263e-10**
Fundamental unit of charge (electron charge) [fr]

holodeck.constants.**RSOL = 69570000000.0**
Solar Radius [cm]

holodeck.constants.**SCHW = 1.4852320538237328e-28**
Schwarzschild Constant (2*G/c^2) [cm]

holodeck.constants.**SIGMA_SB = 5.6703744191844314e-05**
Stefan-Boltzmann constant [erg/cm^2/s/K^4]

holodeck.constants.**SIGMA_T = 6.6524587321000005e-25**
Thomson/Electron -Scattering cross-section [cm^2]

holodeck.constants.**SPLC = 29979245800.0**
Speed of light [cm/s]

holodeck.constants.**YR = 31557600.0**
year [s]

CHAPTER

TWELVE

HOLODECK.CYUTILS

Module for methods implemented in cython.

To use and load this module, you will need to build the cython extension using:

$ python setup.py build_ext -i

from the holodeck root directory (containing the *setup.py* file).

And you still need to install holodeck in develop mode, using:

$ python setup.py develop

holodeck.cyutils.**Sh_rest**()

Calculate the noise from all the single sources except the source in question and the next N_excl loudest sources.

Parameters

- **hc_ss** (*(F,R,L) NDarray*) – Characteristic strain from all loud single sources.
- **hc_bg** (*(F,R) NDarray*) – Characteristic strain from all but loudest source at each frequency.
- **freqs** (*(F,) 1Darray*) – Frequency bin centers.
- **nexcl** (*int*) – Number of loudest single sources to exclude from hc_rest noise, in addition to the source in question.

Returns

Sh_rest – The noise in a single pulsar from other GW sources for detecting each single source.

Return type

(F,R,L) NDarray of scalars

holodeck.cyutils.**gamma_of_rho_interp**()

rho

[1Darray of scalars] SNR of single sources, in flat array

rsort

[1Darray] order of flat rho values smallest to largest

rho_interp_grid

[1Darray] rho values corresponding to each gamma

gamma_interp_grid

[1Darray] gamma values corresponding to each rho

holodeck.cyutils.**loudest_hc_and_par_from_sorted**()

Calculates the characteristic strain from loud single sources and a background of all other sources.

Parameters

- **number** (*[M, Q, Z, F] NDarray*) – number in each bin
- **h2fdf** (*[M, Q, Z, F] NDarray*) – Strain amplitude squared x frequency / frequency bin width for each bin.
- **nreals** – Number of realizations.
- **nloudest** – Number of loudest sources to separate in each frequency bin.
- **mt** (*(M,) 1Darray of scalars*) – Total masses, M, of each bin center.
- **mr** (*(Q,) 1Darray of scalars*) – Mass ratios, q, of each bin center.
- **rz** (*(Z,) 1Darray of scalars*) – Redshifts, z, of each bin center.
- **msort** (*(M*Q*Z,) 1Darray*) – M indices of each bin, sorted from largest to smallest h2fdf.
- **qsort** (*(M*Q*Z,) 1Darray*) – q indices of each bin, sorted from largest to smallest h2fdf.
- **zsort** (*(M*Q*Z,) 1Darray*) – z indices of each bin, sorted from largest to smallest h2fdf.
- **normal_threshold** (*float*) – Threshold for approximating poisson sampling as normal.

Returns

- **hc2ss** (*(F, R, L) Ndarray of scalars*) – Char strain squared of the loudest single sources.
- **hc2bg** (*(F, R) Ndarray of scalars*) – Char strain squared of the background.
- **lspar** (*(3, F, R) NDarray of scalars*) – Average effective M, q, z parameters of the loudest L sources.
- **bgpar** (*(3, F, R) NDarray of scalars*) – Average effective M, q, z parameters of the background.
- **ssidx** (*(3, F, R, L) NDarray of ints*) – Indices of the loudest single sources.

holodeck.cyutils.**loudest_hc_and_par_from_sorted_redz**()

Calculates the characteristic strain and binary parameters from loud single sources and a background of all other sources.

Parameters

- **number** (*[M, Q, Z, F] NDarray*) – number in each bin
- **h2fdf** (*[M, Q, Z, F] NDarray*) – Strain amplitude squared x frequency / frequency bin width for each bin.
- **nreals** – Number of realizations.
- **nloudest** – Number of loudest sources to separate in each frequency bin.
- **mt** (*(M,) 1Darray of scalars*) – Total masses, M, of each bin center.
- **mr** (*(Q,) 1Darray of scalars*) – Mass ratios, q, of each bin center.
- **rz** (*(Z,) 1Darray of scalars*) – Redshifts, z, of each bin center.
- **redz_final** (*(M,Q,Z,F) NDarray of scalars*) – Final redshifts of each bin.
- **dcom_final** (*(M,Q,Z,F) NDarray of scalars*) – Final comoving distances of each bin.

- **sepa** (*(M,Q,Z,F) NDarray of scalars*) – Final separations of each mass and frequency combination.
- **angs** (*(M,Q,Z,F)*) – Final angular separations of each bin.
- **msort** (*(M*Q*Z,) 1Darray*) – M indices of each bin, sorted from largest to smallest h2fdf.
- **qsort** (*(M*Q*Z,) 1Darray*) – q indices of each bin, sorted from largest to smallest h2fdf.
- **zsort** (*(M*Q*Z,) 1Darray*) – z indices of each bin, sorted from largest to smallest h2fdf.
- **normal_threshold** (*float*) – Threshold for approximating poisson sampling as normal.

Returns

- **hc2ss** (*(F, R, L) Ndarray of scalars*) – Char strain squared of the loudest single sources.
- **hc2bg** (*(F, R) Ndarray of scalars*) – Char strain squared of the background.
- **sspar** (*(4, F, R) NDarray of scalars*) – Effective M, q, z parameters of the loudest L sources. mass, ratio, redshift, redshift_final
- **bgpar** (*(4, F, R) NDarray of scalars*) – Average effective M, q, z parameters of the background. mass, ratio, redshift, redshift_final

holodeck.cyutils.**loudest_hc_from_sorted**()

Calculates the characteristic strain from loud single sources and a background of all other sources.

Parameters

- **number** (*[M, Q, Z, F] NDarray*) – number in each bin
- **h2fdf** (*[M, Q, Z, F] NDarray*) – Strain amplitude squared x frequency / frequency bin width for each bin.
- **nreals** – Number of realizations.
- **nloudest** – Number of loudest sources to separate in each frequency bin.
- **msort** (*(M*Q*Z,) 1Darray*) – M indices of each bin, sorted from largest to smallest h2fdf.
- **qsort** (*(M*Q*Z,) 1Darray*) – q indices of each bin, sorted from largest to smallest h2fdf.
- **zsort** (*(M*Q*Z,) 1Darray*) – z indices of each bin, sorted from largest to smallest h2fdf.
- **normal_threshold** (*float*) – Threshold for approximating poisson sampling as normal.

Returns

- **hc2ss** (*(F, R, L) Ndarray of scalars*) – Char strain squared of the loudest single sources.
- **hc2bg** (*(F, R) Ndarray of scalars*) – Char strain squared of the background.

holodeck.cyutils.**sam_calc_gwb_single_eccen**()

Pure-python wrapper for the SAM eccentric GWB calculation method. See: *_sam_calc_gwb_single_eccen()*.

holodeck.cyutils.**sam_calc_gwb_single_eccen_discrete**()

Pure-python wrapper for the SAM eccentric GWB calculation method. See: *_sam_calc_gwb_single_eccen()*.

holodeck.cyutils.**snr_ss**()

Calculate single source SNR.

Parameters

- **amp** (*(F,R,L) NDarray*) – Dimensionless strain amplitude of loudest single sources
- **F_iplus** (*(P,F,S,L) NDarray*) – Antenna pattern function for each pulsar.

- **F_icross** (*(P,F,S,L) NDarray*) – Antenna pattern function for each pulsar.
- **iotas** (*(F,S,L) NDarray*) – Inclination, used to calculate: $a_pol = 1 + np.cos(iotas)^2$ $b_pol = -2np.cos(iotas)$
- **dur** (*scalar*) – Duration of observations.
- **Phi_0** (*(F,S,L) NDarray*) – Initial GW phase
- **S_i** (*(P,F,R,L) NDarray*) – Total noise of each pulsar wrt detection of each single source, in s^3.
- **freqs** (*(F,) 1Darray*) – Observed frequency bin centers.

Returns

snr_ss – SNR from the whole PTA for each single source with each realized sky position (S) and realized strain (R)

Return type

(F,R,S,L) NDarray

holodeck.cyutils.**sort_h2fdf**()

Get indices of sorted h2fdf. :param h2fdf: h_s^2 * f / df of a source in each bin. :type h2fdf: (M,Q,Z) NDarray

Returns

indices

Return type

?

holodeck.cyutils.**ss_bg_hc**()

Calculates the characteristic strain from loud single sources and a background of all other sources.

Parameters

- **number** (*[M, Q, Z, F] ndarray*) – number in each bin
- **h2fdf** (*[M, Q, Z, F] ndarray*) – strain squared x frequency / frequency bin width for each bin
- **nreals** – number of realizations

Returns

- **hc2ss** (*(F, R) Ndarray of scalars*)
- **hc2bg** (*(F, R) Ndarray of scalars*)
- **ssidx** (*(3, F, R) Ndarray of ints*) – Index of the loudest single source, -1 if there are none at the frequency/realization.

holodeck.cyutils.**ss_bg_hc_and_par**()

Calculates the characteristic strain from loud single sources and a background of all other sources.

Parameters

- **number** (*[M, Q, Z, F] NDarray*) – number in each bin
- **h2fdf** (*[M, Q, Z, F] NDarray*) – Strain amplitude squared x frequency / frequency bin width for each bin.
- **nreals** – Number of realizations.
- **mt** (*(M,) 1Darray of scalars*) – Total masses, M, of each bin center.
- **mr** (*(Q,) 1Darray of scalars*) – Mass ratios, q, of each bin center.

- **rz** (*(Z,) 1Darray of scalars*) – Redshifts, z, of each bin center.

Returns

- **hc2ss** (*(F, R) Ndarray of scalars*) – Char strain squared of the loudest single sources.
- **hc2bg** (*(F, R) Ndarray of scalars*) – Char strain squared of the background.
- **ssidx** (*(3, F, R) NDarray of ints*) – Indices of the loudest single sources. -1 if there are no single sources at that frequency/realization.
- **bgpar** (*(3, F, R) NDarray of scalars*) – Average effective M, q, z parameters of the background.
- **sspar** (*(3, F, R) NDarray of scalars*) – M, q, z parameters of the loudest single sources.

CHAPTER

THIRTEEN

HOLODECK.EVOLUTION

CHAPTER

FOURTEEN

HOLODECK.GRAVWAVES

Gravitational Wave (GW) calculations module.

This module provides tools for calculating GW signals from MBH binaries. Currently the components here are used with the 'discrete' / 'illustris' population of binaries, and not the semi-analytic or observational population models.

holodeck.gravwaves.**gws_from_sampled_strains**(*fobs_gw_edges*, *fo*, *hs*, *weights*)

Calculate GW background/foreground from sampled GW strains.

Parameters

- **fobs_gw_edges** (*(F,) array_like of scalar*) – Observer-frame GW-frequency bin edges.
- **fo** (*(S,) array_like of scalar*) – Observer-frame GW-frequency of each binary sample. Units of [1/sec]
- **hs** (*(S,) array_like of scalar*) – GW source strain (*not characteristic strain*) of each binary sample.
- **weights** (*(S,) array_like of int*) – Weighting factor for each binary. NOTE: the GW calculation is ill-defined if weights have fractional values (i.e. float values, instead of integral values; but the type itself doesn't matter)

Returns

- **gwf_freqs** (*(F,) ndarray of scalar*) – Observer-frame GW frequency of foreground sources in each frequency bin. Units of [1/sec].
- **gwfore** (*(F,) ndarray of scalar*) – Strain amplitude of foreground sources in each frequency bin.
- **gwback** (*(F,) ndarray of scalar*) – Strain amplitude of the background in each frequency bin.

holodeck.gravwaves.**poisson_as_needed**(*values*, *thresh=10000000000.0*)

Calculate Poisson distribution when values are below threshold, otherwise approximate with normal distribution.

Parameters

- **values** (*ndarray*) – Expectation values for poisson distribution.
- **thresh** (*float*) – Expectation value above which to use Normal distribution approximation.

Returns

output – (Approximately) Poisson distributed values. Same shape as input *values*.

Return type

ndarray

holodeck.gravwaves.**sam_calc_gwb_single_eccen_discrete**(*gwfobs*, *sam*, *sepa_evo*, *eccen_evo*, *nharms=100*, *nreals=None*)

Parameters

- **gwfobs** (*(F,) array_like*) – Observer-frame frequencies at which to calculate GWB.
- **sam** (*Semi_Analytic_Model* instance) – Binary population to sample. See *holodeck.simple_sam* or 'holodeck.sam`
- **sepa_evo** – Separation at each evolution step.
- **eccen_evo** (*(E,) array_like*) – Eccentricities at each evolution step. The same for all binaries, corresponding to fixed binary separations for all binaries.
- **nharms** (*int, optional*) – Number of harmonics to use in calculating GWB.
- **nreals** (*int or None, optional*) – Number of realizations to calculate in Poisson sampling.

Returns

gwb – GW Background: the characteristic strain of the GWB in each frequency bin. Does not include the strain from the loudest binary in each bin (*gwf*).

Return type

(F,) ndarray,

holodeck.gravwaves.**sampled_gws_from_sam**(*sam*, *fobs_gw*, *hard=<class 'holodeck.hardening.Hard_GW'>*, ***kwargs*)

Sample the given binary population between the target frequencies, and calculate GW signals.

NOTE: the input *fobs* are interpretted as bin edges, and GW signals are calculate within the corresponding bins.

Parameters

- **sam** (*Semi_Analytic_Model* instance,) – Binary population to sample.
- **fobs_gw** (*(F+1,) array_like,*) – Target observer-frame GW-frequencies of interest in units of [1/sec]
- **hard** (*holodeck.evolution._Hardening* instance,) – Binary hardening model used to calculate binary residence time at each frequency.
- **kwargs** (*dict,*) – Additional keyword-arguments passed to *sample_sam_with_hardening()*

Returns

- **gff** (*(F,) ndarry,*) – Observer-frame GW-frequencies of the loudest binary in each bin [1/sec].
- **gwf** (*(F,) ndarry,*) – GW Foreground: the characteristic strain of the loudest binary in each frequency bin.
- **gwb** (*(F,) ndarry,*) – GW Background: the characteristic strain of the GWB in each frequency bin. Does not include the strain from the loudest binary in each bin (*gwf*).

CHAPTER

FIFTEEN

HOLODECK.HARDENING

Binary evolution hardening submodules.

15.1 To-Do (Hardening)

- **Dynamical_Friction_NFW**
  - Allow stellar-density profiles to also be specified (instead of using a hard-coded Dehnen profile)
  - Generalize calculation of stellar characteristic radius. Make self-consistent with stellar-profile, and user-specifiable.
- **Evolution**
  - *_sample_universe()* : sample in comoving-volume instead of redshift
- **Sesana_Scattering**
  - Allow stellar-density profile (or otherwise the binding-radius) to be user-specified and flexible. Currently hard-coded to Dehnen profile estimate.
- **_SHM06**
  - Interpolants of hardening parameters return 1D arrays.
- **Fixed_Time_2PL**
  - Handle *rchar* better with respect to interpolation. Currently not an interpolation variable, which restricts it's usage.
  - This class should be separated into a generic *_Fixed_Time* class that can use any functional form, and then a 2-power-law functional form that requires a specified normalization. When they're combined, it will produce the same effect. Another good functional form to implement would be GW + log-uniform hardening time, the same as the current phenomenological model but with both power-laws set to 0.

References

- [BBR1980] Begelman, Blandford & Rees 1980.
- [Chen2017] Chen, Sesana, & Del Pozzo 2017.
- [Kelley2017a] Kelley, Blecha & Hernquist 2017.
- [Quinlan1996] Quinlan 1996.
- [Sesana2006] Sesana, Haardt & Madau et al. 2006.

- [Sesana2010] Sesana 2010.
- [Siwek2023] Siwek+2023

class holodeck.hardening.**CBD_Torques**(*f_edd=0.1*, *subpc=True*)

Binary Orbital Evolution based on Hydrodynamic Simulations by Siwek+23.

This module uses data from Siwek+23, which supplies rates of change of binary semi-major axis a_b and binary eccentricity e_b. The calculation of a_b and e_b versus time requires accretion rates (for scale).

dadt_dedt(*evo*, *step*)

Circumbinary Disk Torque hardening rate.

Parameters

- **evo** (*Evolution*) – Evolution instance providing binary parameters at the given intergration step.
- **step** (*int*) – Integration step at which to calculate hardening rates.

Returns

- **dadt** (*array_like*) – Binary hardening rates in units of [cm/s], defined to be negative.
- **dedt** (*array_like*) – Binary rate-of-change of eccentricity in units of [1/sec].

class holodeck.hardening.**Dynamical_Friction_NFW**(*mmbulge=None*, *msigma=None*, *smhm=None*, *coulomb=10.0*, *attenuate=True*, *rbound_from_density=True*)

Dynamical Friction (DF) hardening module assuming an NFW dark-matter density profile.

This class calculates densities and orbital velocities based on a NFW profile with parameters based on those of each MBH binary. The *holodeck.observations.NFW* class is used for profile calculations, and the halo parameters are calculated from Stellar-mass–Halo-mass relations (see 'arguments' below). The 'effective-mass' of the inspiralling secondary is modeled as a power-law decreasing from the sum of secondary MBH and its stellar-bulge (calculated using the *mmbulge* - Mbh-Mbulge relation), down to just the bare secondary MBH after 10 dynamical times. This is to model tidal-stripping of the secondary host galaxy.

Attenuation of the DF hardening rate is typically also included, to account for the inefficiency of DF once the binary enters the hardened regime. This is calculated using the prescription from [BBR1980]. The different characteristic radii, needed for the attenuation calculation, currently use a fixed Dehnen stellar-density profile as in [Chen2017], and a fixed scaling relationship to find the characteristic stellar-radius.

Notes

- This module does not evolve eccentricity.
- The hardening rate (da/dt) is not allowed to be larger than the orbital/virial velocity of the halo (as a function of radius).

dadt_dedt(*evo*, *step*, *attenuate=None*)

Calculate DF hardening rate given *Evolution* instance, and an integration *step*.

Parameters

- **evo** (*Evolution* instance) – The evolutionary tracks of the binary population, providing binary parameters.
- **step** (*int,*) – Integration step at which to calculate hardening rates.

Returns

- **dadt** (*(N,) np.ndarray of scalar,*) – Binary hardening rates in units of [cm/s].
- **dedt** (*(N,) np.ndarray or None*) – Rate-of-change of eccentricity, which is not included in this calculation, it is zero. *None* is returned if the input *eccen* is None.

class holodeck.hardening.**Fixed_Time_2PL**(*time*, *mtot*, *mrat*, *redz*, *sepa_init*, *rchar=3.0856775814913674e+20*, *gamma_inner=-1.0*, *gamma_outer=1.5*, *progress=False*, *interpolate_norm=False*)

Provide a binary hardening rate such that the total lifetime matches a given value.

This class uses a phenomenological functional form (defined in `Fixed_Time.function()`) to model the hardening rate (da/dt) of binaries. The functional form is,

$$\dot{a} = -A * (1.0 + x)^{-g_2 - 1} / x^{g_1 - 1},$$

where $x \equiv a/r_{\rm char}$ is the binary separation scaled to a characteristic transition radius ($r_{\rm char}$) between two power-law indices g_1 and g_2. There is also an overall normalization A that is calculated to yield the desired binary lifetimes.

The normalization for each binary, to produce the desired lifetime, is calculated as follows:

(1) A set of random test binary parameters are chosen.

(2) The normalization constants are determined, using least-squares optimization, to yield the desired lifetime.

(3) Interpolants are constructed to interpolate between the test binary parameters.

(4) The interpolants are called on the provided binary parameters, to calculate the interpolated normalization constants to reach the desired lifetimes.

Construction/Initialization: note that in addition to the standard `Fixed_Time.__init__()` constructor, there are two additional constructors are provided:

- `Fixed_Time.from_pop()` - accept a `holodeck.population._Discrete_Population`,
- `Fixed_Time.from_sam()` - accept a `holodeck.sam.Semi_Analytic_Model`.

#! Using a callable for *rchar* probably doesnt work - *_calculate_norm_interpolant* looks like #! it only accepts a scalar value.

dadt_dedt(*evo*, *step*)

Calculate hardening rate at the given integration *step*, for the given population.

Parameters

- **evo** (*Evolution* instance) – The evolutionary tracks of the binary population, providing binary parameters.
- **step** (*int,*) – Integration step at which to calculate hardening rates.

Returns

- **dadt** (*(N,) np.ndarray*) – Binary hardening rates in units of [cm/s].
- **dedt** (*(N,) np.ndarray or None*) – Rate-of-change of eccentricity, which is not included in this calculation, it is zero. *None* is returned if the input *eccen* is None.

classmethod from_pop(*pop*, *time*, ***kwargs*)

Initialize a *Fixed_Time* instance using a provided *_Discrete_Population* instance.

Parameters

- **pop** (*_Discrete_Population*) – Input population, from which to use masses, redshifts and separations.

- **time** (*float, callable or array_like*) – Total merger time of binaries, units of [sec], specifiable in the following ways:
  - float : uniform merger time for all binaries
  - callable : function *time(mtot, mrat, redz)* which returns the total merger time
  - array_like : (N,) matching the shape of *mtot* (etc) giving the merger time for each binary
- ****kwargs** (*dict*) – Additional keyword-argument pairs passed to the *Fixed_Time* initialization method.

Returns
Instance configured for the given binary population.

Return type
Fixed_Time

classmethod from_sam(*sam*, *time*, *sepa_init=3.0856775814913676e+22*, ***kwargs*)

Initialize a *Fixed_Time* instance using a provided *Semi_Analytic_Model* instance.

Parameters

- **sam** (*holodeck.sam.Semi_Analytic_Model*) – Input population, from which to use masses, redshifts and separations.
- **time** (*float, callable or array_like*) – Total merger time of binaries, units of [sec], specifiable in the following ways:
  - float : uniform merger time for all binaries
  - callable : function *time(mtot, mrat, redz)* which returns the total merger time
  - array_like : (N,) matching the shape of *mtot* (etc) giving the merger time for each binary
- **sepa_init** (*float or array_like*) – Initial binary separation. Units of [cm].
  - float : initial separation applied to all binaries,
  - array_like : initial separations for all binaries, shaped (N,) matching the number binaries.
- ****kwargs** (*dict*) – Additional keyword-argument pairs passed to the *Fixed_Time* initialization method.

Returns
Instance configured for the given binary population.

Return type
Fixed_Time

classmethod function(*norm*, *xx*, *gamma_inner*, *gamma_outer*)

Hardening rate given the parameters for this hardening model.

The functional form is,

$$\dot{a} = -A * (1.0 + x)^{-g_o ut + g_i n} / x^{g_i n - 1},$$

Where A is an overall normalization, and $x = a/r_{\rm char}$ is the binary separation scaled to a characteristic transition radius ($r_{\rm char}$) between two power-law indices $g_i nner$ and $g_o uter$.

Parameters

- **norm** (*array_like*) – Hardening rate normalization, units of [cm/s].
- **xx** (*array_like*) – Dimensionless binary separation, the semi-major axis in units of the characteristic (i.e. transition) radius of the model *rchar*.

- **gamma_inner** (*scalar*) – Power-law of hardening timescale in the inner regime (small separations: r < rchar).
- **gamma_outer** (*scalar*) – Power-law of hardening timescale in the outer regime (large separations: r > rchar).

class holodeck.hardening.**Fixed_Time_2PL_SAM**(*sam*, *time*, *sepa_init=3.0856775814913673e+21*, *rchar=3.0856775814913675e+19*, *gamma_inner=-1.0*, *gamma_outer=1.5*, *num_steps=300*)

Provide a binary hardening rate such that the total lifetime matches a given value.

class holodeck.hardening.**Hard_GW**

Gravitational-wave driven binary hardening.

static **dadt**(*mtot*, *mrat*, *sepa*, *eccen=None*)

Calculate GW Hardening rate of semi-major-axis vs. time.

See [Peters1964], Eq. 5.6

Parameters

- **mtot** (*array_like*) – Total mass of each binary system. Units of [gram].
- **mrat** (*array_like*) – Mass ratio of each binary, defined as $q \equiv m_1/m_2 \leq 1.0$.
- **sepa** (*array_like*) – Binary semi-major axis (separation), in units of [cm].
- **eccen** (*array_like or None*) – Binary eccentricity, *None* is the same as zero eccentricity (circular orbit).

Returns

dadt – Hardening rate in semi-major-axis, result is negative, units [cm/s].

Return type

np.ndarray

static **dadt_dedt**(*evo*, *step*)

Calculate GW binary evolution (hardening rate) in semi-major-axis and eccentricity.

Parameters

- **evo** (*Evolution*) – Evolution instance providing the binary parameters for calculating hardening rates.
- **step** (*int*) – Evolution integration step index from which to load binary parameters. e.g. separations are loaded as `evo.sepa[:, step]`.

Returns

- **dadt** (*np.ndarray*) – Hardening rate in semi-major-axis, returns negative value, units [cm/s].
- **dedt** (*np.ndarray*) – Hardening rate in eccentricity, returns negative value, units [1/s].

static **deda**(*sepa*, *eccen*)

Rate of eccentricity change versus separation change.

See [Peters1964], Eq. 5.8

Parameters

- **sepa** (*array_like,*) – Binary semi-major axis (i.e. separation) [cm].
- **eccen** (*array_like,*) – Binary orbital eccentricity.

Returns
rv – Binary deda rate [1/cm] due to GW emission. Values are always positive.

Return type
array_like,

`static dedt(mtot, mrat, sepa, eccen=None)`

Calculate GW Hardening rate of eccentricity vs. time.

See [Peters1964], Eq. 5.7

If *eccen* is *None*, zeros are returned.

Parameters

- **mtot** (*array_like*) – Total mass of each binary system. Units of [gram].
- **mrat** (*array_like*) – Mass ratio of each binary, defined as $q \equiv m_1/m_2 \leq 1.0$.
- **sepa** (*array_like*) – Binary semi-major axis (separation), in units of [cm].
- **eccen** (*array_like or None*) – Binary eccentricity, *None* is the same as zero eccentricity (circular orbit).

Returns
dedt – Hardening rate in eccentricity, result is <= 0.0, units [1/s]. Zero values if *eccen* is *None*.

Return type
np.ndarray

`class holodeck.hardening.Sesana_Scattering(gamma_dehnen=1.0, mmbulge=None, msigma=None)`

Binary-Hardening Rates calculated based on the Sesana stellar-scattering model.

This module uses the stellar-scattering rate constants from the fits in [Sesana2006] using the *_SHM06* class. Scattering is assumed to only be effective once the binary is bound. An exponential cutoff is imposed at larger radii.

`dadt_dedt(evo, step)`

Stellar scattering hardening rate.

Parameters

- **evo** (*Evolution*) – Evolution instance providing binary parameters at the given intergration step.
- **step** (*int*) – Integration step at which to calculate hardening rates.

Returns

- **dadt** (*array_like*) – Binary hardening rates in units of [cm/s], defined to be negative.
- **dedt** (*array_like*) – Binary rate-of-change of eccentricity in units of [1/sec].

CHAPTER

SIXTEEN

HOLODECK.LOGGER

Logging module.

This module produces a *logging.Logger* instance that can log both to stdout (i.e. using print) and also to an output file. This is especially useful for long or parallelized calculations where more significant diagnostic outputs are required for debugging and/or record-keeping.

holodeck.logger.**get_logger**(*name='holodeck'*, *level_stream=30*, *tostr=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>*, *tofile=None*, *level_file=10*)

Create a standard logger object which logs to file and or stdout stream.

Parameters

- **name** (*str,*) – Handle for this logger, must be distinct for a distinct logger.
- **level_stream** (*int,*) – Logging level for stream.
- **tostr** (*bool,*) – Log to stdout stream.
- **tofile** (str or *None*,) – Filename to log to (turned off if *None*).
- **level_file** (*int,*) – Logging level for file.

Returns

logger – Logger object to use for logging.

Return type

logging.Logger object,

CHAPTER

SEVENTEEN

HOLODECK.PLOT

Plotting module.

Provides convenience methods for generating standard plots and components using *matplotlib*.

class holodeck.plot.**MidpointLogNormalize**(*vmin=None*, *vmax=None*, *midpoint=0.0*, *clip=False*)

class holodeck.plot.**MidpointNormalize**(*vmin=None*, *vmax=None*, *midpoint=0.0*, *clip=False*)

Normalise the colorbar so that diverging bars work there way either side from a prescribed midpoint value)

e.g. im=ax1.imshow(array, norm=MidpointNormalize(midpoint=0.,vmin=-100, vmax=100))

holodeck.plot.**figax**(*figsize=[7, 5]*, *ncols=1*, *nrows=1*, *sharex=False*, *sharey=False*, *squeeze=True*, *scale=None*, *xscale='log'*, *xlabel=''*, *xlim=None*, *yscale='log'*, *ylabel=''*, *ylim=None*, *left=None*, *bottom=None*, *right=None*, *top=None*, *hspace=None*, *wspace=None*, *widths=None*, *heights=None*, *grid=True*, ***kwargs*)

Create matplotlib figure and axes instances.

Convenience function to create fig/axes using *plt.subplots*, and quickly modify standard parameters.

Parameters

- **figsize** (*(2,) list, optional*) – Figure size in inches.
- **ncols** (*int, optional*) – Number of columns of axes.
- **nrows** (*int, optional*) – Number of rows of axes.
- **sharex** (*bool, optional*) – Share xaxes configuration between axes.
- **sharey** (*bool, optional*) – Share yaxes configuration between axes.
- **squeeze** (*bool, optional*) – Remove dimensions of length (1,) in the *axes* object.
- **scale** (*[type], optional*) – Axes scaling to be applied to all x/y axes. One of [‘log’, ‘lin’].
- **xscale** (*str, optional*) – Axes scaling for xaxes [‘log’, ‘lin’].
- **xlabel** (*str, optional*) – Label for xaxes.
- **xlim** (*[type], optional*) – Limits for xaxes.
- **yscale** (*str, optional*) – Axes scaling for yaxes [‘log’, ‘lin’].
- **ylabel** (*str, optional*) – Label for yaxes.
- **ylim** (*[type], optional*) – Limits for yaxes.
- **left** (*[type], optional*) – Left edge of axes space, set using *plt.subplots_adjust()*, as a fraction of figure.

- **bottom** (*[type], optional*) – Bottom edge of axes space, set using *plt.subplots_adjust()*, as a fraction of figure.
- **right** (*[type], optional*) – Right edge of axes space, set using *plt.subplots_adjust()*, as a fraction of figure.
- **top** (*[type], optional*) – Top edge of axes space, set using *plt.subplots_adjust()*, as a fraction of figure.
- **hspace** (*[type], optional*) – Height space between axes if multiple rows are being used.
- **wspace** (*[type], optional*) – Width space between axes if multiple columns are being used.
- **widths** (*[type], optional*) –
- **heights** (*[type], optional*) –
- **grid** (*bool, optional*) – Add grid lines to axes.

Returns

- **fig** (*matplotlib.figure.Figure*) – New matplotlib figure instance containing axes.
- **axes** ([ndarray] *matplotlib.axes.Axes*) – New matplotlib axes, either a single instance or an ndarray of axes.

holodeck.plot.**plot_bg_ss**(*fobs*, *bg*, *ss=None*, *bglabel=None*, *sslabel=None*, *xlabel='GW Frequency $[\\mathrm{yr}^{-1}]$'*, *ylabel='GW Characteristic Strain'*, ***kwargs*)

Can plot strain or power spectral density, just need to set ylabel accordingly

holodeck.plot.**scientific_notation**(*val*, *man=1*, *exp=0*, *dollar=True*)

Convert a scalar into a string with scientific notation (latex formatted).

Parameters

- **val** (*scalar*) – Numerical value to convert.
- **man** (int or *None*) – Precision of the mantissa (decimal points); or *None* for omit mantissa.
- **exp** (int or *None*) – Precision of the exponent (decimal points); or *None* for omit exponent.
- **dollar** (*bool*) – Include dollar-signs ('$') around returned expression.

Returns

rv_str – Scientific notation string using latex formatting.

Return type

str

holodeck.plot.**smap**(*args=[0.0, 1.0]*, *cmap=None*, *log=False*, *norm=None*, *midpoint=None*, *under='0.8'*, *over='0.8'*, *left=None*, *right=None*)

Create a colormap from a scalar range to a set of colors.

Parameters

- **args** (*scalar or array_like of scalar*) – Range of valid scalar values to normalize with
- **cmap** (None, str, or `matplotlib.colors.Colormap` object) – Colormap to use.
- **log** (*bool*) – Logarithmic scaling
- **norm** (None or *matplotlib.colors.Normalize*) – Normalization to use.
- **under** (str or *None*) – Color specification for values below range.

- **over** (str or *None*) – Color specification for values above range.
- **left** (float {0.0, 1.0} or *None*) – Truncate the left edge of the colormap to this value. If *None*, 0.0 used (if *right* is provided).
- **right** (float {0.0, 1.0} or *None*) – Truncate the right edge of the colormap to this value If *None*, 1.0 used (if *left* is provided).

Returns

smap – Scalar mappable object which contains the members: *norm*, *cmap*, and the function *to_rgba*.

Return type

matplotlib.cm.ScalarMappable

holodeck.plot.**truncate_colormap**(*cmap*, *minval=0.0*, *maxval=1.0*, *n=100*)

https://stackoverflow.com/a/18926541

CHAPTER

EIGHTEEN

HOLODECK.POPULATION

CHAPTER

NINETEEN

HOLODECK.RELATIONS

Empirical and phenomenological scaling relationships.

This module defines numerous classes and accessor methods to implement scaling relationships between different empirical quantities, for example BH-Galaxy relations, or Stellar-Mass vs Halo-Mass relations. *abc* base classes are used to implement generic functionality, and define APIs while subclasses are left to perform specific implementations. In general most classes implement both the forward and reverse versions of relationships (e.g. stellar-mass to halo-mass, and also halo-mass to stellar-mass). Reverse relationships are often interpolated over a grid.

Most of the relationships currently implemented are among three groups (and corresponding base classes):

- **BH-Host Relations** (*_Host_Relation*): These produce mappings between host galaxy properties (e.g. bulge mass) and the properties of their black holes (i.e. BH mass).
  - **Mbh-Mbulge relations** ("M-Mbulge"; *_MMBulge_Relation*): mapping from host galaxy stellar bulge mass to black-hole mass.
  - **Mbh-Sigma relations** ("M-Sigma"; *_MSigma_Relation*): mapping from host galaxy velocity dispersion (sigma) to black-hole mass.
- **Density Profiles** (*_Density_Profile*): matter density as a function of spherical radius.
- **Stellar-Mass vs. Halo-Mass Relations** (*_StellarMass_HaloMass*): mapping from halo-mass to stellar-mass.

19.1 To-Do

- Pass concentration-relation (or other method to calculate) to NFW classes on instantiation
- For redshift-dependent extensions to relations, use multiple-inheritance instead of repeating attributes.

References

- [Behroozi2013] : Behroozi, Wechsler & Conroy 2013.
- [Guo2010] Guo, White, Li & Boylan-Kolchin 2010.
- [Klypin2016] Klypin et al. 2016.
- [KH2013] Kormendy & Ho 2013.
- [MM2013] McConnell & Ma 2013.
- [NFW1997] Navarro, Frenk & White 1997.

class holodeck.relations.**Behroozi_2013**(**args*, ***kwargs*)

Redshift-dependent Stellar-Mass - Halo-Mass relation based on Behroozi et al. 2013.

[Behroozi2013] best fit values are at the beginning of Section 5 (pg.9), uncertainties are 1-sigma.

stellar_mass(*mhalo*, *redz*)

This is [Behroozi2013] Eq.3 (upper)

class holodeck.relations.**Guo_2010**

Stellar-Mass - Halo-Mass relation from Guo et al. 2010.

[Guo2010] Eq.3

classmethod stellar_mass(*mhalo*)

Calculate the stellar-mass for the given halo mass.

Parameters
mhalo (*ArrayLike*) – Halo mass. [gram]

Returns
mstar – Stellar mass. [gram]

Return type
ArrayLike

class holodeck.relations.**Klypin_2016**

Class to calculate dark matter halo 'concentration' parameters based on [Klypin2016].

This class does not need to be instantiated, all methods are class methods, simply call `Klypin_2016.concentration()`.

Interpolate between redshifts and masses to find DM halo concentrations. [Klypin2016] Eq. 24 & Table 2.

classmethod concentration(*mhalo: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *redz: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Return the halo concentration for the given halo mass and redshift.

Parameters

- **mhalo** (*ArrayLike*) – Halo mass. [grams]
- **redz** (*ArrayLike*) – Redshift.

Returns
conc – Halo concentration parameters. []

Return type
ArrayLike

class holodeck.relations.**MMBulge_KH2013**(*mamp=None*, *mamp_log10=None*, *mplaw=None*, *mref=None*, *bulge_mfrac=0.615*, *scatter_dex=None*)

Mbh-MBulge Relation, single power-law, from Kormendy & Ho 2013.

Values taken from [KH2013] Eq.10.

MASS_AMP_LOG10 = None

class holodeck.relations.**MMBulge_MM2013**(*mamp=None*, *mamp_log10=None*, *mplaw=None*, *mref=None*, *bulge_mfrac=0.615*, *scatter_dex=None*)

Mbh-MBulge Relation from McConnell & Ma 2013

[MM2013] Eq. 2, with values taken from Table 2 ("Dynamical masses", first row, "MPFITEXY")

class holodeck.relations.**MMBulge_Redshift**(**args*, *zplaw=None*, ***kwargs*)

Mbh-Mbulge relation with an additional redshift power-law dependence.

Provides black hole mass as a function of galaxy bulge mass and redshift with a normalization that depends on redshift. zplaw=0 (default) is identical to MMBulge_Standard. mamp = mamp0 * (1 + z)**zplaw

TODO: make sure all of the inherited methods from *MMBulge_Standard* are appropriate for redshift dependencies!! In particular, check *dmstar_dmbh* check which redshifts need to be passed into this function. does not pass all cases as is

MASS_AMP_LOG10 = None

dmstar_dmbh(*mstar*, *redz*)

Calculate the partial derivative of stellar mass versus BH mass dM_star/dM_bh.

$$dM_star/dM_bh = [dM_star/dM_bulge] * [dM_bulge/dM_bh] \ = [1/f_bulge] * [M_bulge/(plaw * M_bh)]$$

Parameters
mstar (*array_like,*) – Total stellar mass of galaxy. [grams]

Returns
deriv – Jacobian term.

Return type
array_like,

mbh_from_host(*pop*, *scatter*)

Convert from abstract host galaxy properties to blackhole mass.

The *pop* instance must contain the attributes required for this class's scaling relations. The required properties are stored in this class's *_PROPERTIES* attribute.

Parameters
pop (*_Discrete_Population*,) – Population instance having the attributes required by this particular scaling relation.

Returns
mbh – Black hole mass. [grams]

Return type
ArrayLike

mbh_from_mbulge(*mbulge*, *redz*, *scatter*)

Convert from stellar-bulge mass to black-hole mass.

Parameters

- **mbulge** (*array_like,*) – Stellar bulge-mass of host galaxy. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
mbh – Mass of black hole. [grams]

Return type
array_like,

mbh_from_mstar(*mstar*, *redz*, *scatter*)

Convert from total stellar mass to black-hole mass.

Parameters

- **mstar** (*array_like,*) – Total stellar mass of host galaxy. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
mbh – Mass of black hole. [grams]

Return type
array_like,

mbulge_from_mbh(*mbh*, *redz*, *scatter*)

Convert from black-hole mass to stellar-bulge mass.

Parameters

- **mbh** (*array_like,*) – Mass of black hole. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
mbulge – Mass of stellar bulge. [grams]

Return type
array_like,

mstar_from_mbh(*mbh*, *redz*, *scatter*)

Convert from black-hole mass to total stellar mass.

Parameters

- **mbh** (*array_like,*) – Mass of black hole. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
Total stellar mass of host galaxy. [grams]

Return type
array_like,

class holodeck.relations.**MMBulge_Redshift_KH2013**(**args*, *zplaw=None*, ***kwargs*)

Mbh-MBulge Relation from Kormendy & Ho 2013, w/ optional redshift evolution of normalization.

BUG/FIX: use multiple-inheritance for this

Values taken from [KH2013] Eq.10 (pg. 61 of PDF, "571" of ARAA)

MASS_AMP_LOG10 = None

class holodeck.relations.**MMBulge_Redshift_MM2013**(**args*, *zplaw=None*, ***kwargs*)

Mbh-MBulge Relation from McConnell & Ma 2013 for z=0 plus redshift evolution of the normalization

BUG/FIX: use multiple-inheritance for this

[MM2013] Eq. 2, with values taken from Table 2 ("Dynamical masses", first row, "MPFITEXY")

MASS_AMP = None

class holodeck.relations.**MMBulge_Standard**(*mamp=None*, *mamp_log10=None*, *mplaw=None*, *mref=None*, *bulge_mfrac=0.615*, *scatter_dex=None*)

Simple Mbh-Mbulge relation as a single power-law.

Notes

- Single power-law relationship between BH mass and Stellar-bulge mass. $Mbh = M0 * (M_bulge/Mref)^plaw * 10^Normal(0, eps)$
- Constant bulge mass-fraction relative to total stellar mass. $M_bulge = f_bulge * M_star$

bulge_mass_frac(*mstar*)

Return the stellar-bulge mass fraction (M_bulge / M_star).

Parameters
mstar (*array_like,*) – Host stellar-mass.

Returns
Bulge mass fraction.

Return type
array_like,

dmstar_dmbh(*mstar*, *redz=None*)

Calculate the partial derivative of stellar mass versus BH mass dM_star/dM_bh.

$$dM_star/dM_bh = [dM_star/dM_bulge] * [dM_bulge/dM_bh] = [1/f_bulge] * [M_bulge/(plaw * M_bh)]$$

Parameters
mstar (*array_like,*) – Total stellar mass of galaxy. [grams]

Returns
deriv – Jacobian term.

Return type
array_like,

mbh_from_host(*pop*, *scatter=None*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Convert from abstract host galaxy properties to blackhole mass.

The *pop* instance must contain the attributes required for this class's scaling relations. The required properties are stored in this class's *_PROPERTIES* attribute.

Parameters
pop (*_Discrete_Population*,) – Population instance having the attributes required by this particular scaling relation.

Returns
mbh – Black hole mass. [grams]

Return type
ArrayLike

mbh_from_mbulge(*mbulge*, *redz=None*, *scatter=None*)

Convert from stellar-bulge mass to black-hole mass.

Parameters

- **mbulge** (*array_like,*) – Stellar bulge-mass of host galaxy. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
mbh – Mass of black hole. [grams]

Return type
array_like,

mbh_from_mstar(*mstar*, *redz=None*, *scatter=None*)

Convert from total stellar mass to black-hole mass.

Parameters

- **mstar** (*array_like,*) – Total stellar mass of host galaxy. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
mbh – Mass of black hole. [grams]

Return type
array_like,

mbulge_from_mbh(*mbh*, *redz=None*, *scatter=None*)

Convert from black-hole mass to stellar-bulge mass.

Parameters

- **mbh** (*array_like,*) – Mass of black hole. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
mbulge – Mass of stellar bulge. [grams]

Return type
array_like,

mstar_from_mbh(*mbh*, *redz=None*, *scatter=None*)

Convert from black-hole mass to total stellar mass.

Parameters

- **mbh** (*array_like,*) – Mass of black hole. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
Total stellar mass of host galaxy. [grams]

Return type
array_like,

mstar_from_mbulge(*mbulge*, *redz=None*)

Convert from stellar bulge-mass to black-hole mass.

Parameters

- **mbulge** (*array_like,*) – Stellar bulge-mass of host galaxy. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
mbh – Mass of black hole. [grams]

Return type
array_like,

class holodeck.relations.**MSigma_KH2013**(*mamp=None*, *sigma_plaw=None*, *sigma_ref=None*, *scatter_dex=None*)

Mbh-Sigma Relation from Kormendy & Ho 2013.

[KH2013] Eq. 10, (pg. 65 of PDF, "575" of ARAA)

class holodeck.relations.**MSigma_MM2013**(*mamp=None*, *sigma_plaw=None*, *sigma_ref=None*, *scatter_dex=None*)

Mbh-Sigma Relation from McConnell & Ma 2013.

[MM2013] Eq. 2, with values taken from Table 2 ("M-sigma all galaxies", first row, "MPFITEXY")

class holodeck.relations.**MSigma_Standard**(*mamp=None*, *sigma_plaw=None*, *sigma_ref=None*, *scatter_dex=None*)

Simple M-sigma relation (BH mass vs. host velocity dispersion) as a single power-law.

Notes

- Single power-law relationship between BH mass and Stellar-bulge mass. $Mbh = M0 * (sigma/sigma_ref)^{plaw} * 10^{N}ormal(0, eps)$

mbh_from_host(*pop*, *scatter*)

Convert from abstract host galaxy properties to blackhole mass.

The *pop* instance must contain the attributes required for this class's scaling relations. The required properties are stored in this class's *_PROPERTIES* attribute.

Parameters
pop (*_Discrete_Population,*) – Population instance having the attributes required by this particular scaling relation.

Returns
mbh – Black hole mass. [grams]

Return type
ArrayLike

mbh_from_vdisp(*vdisp*, *scatter*)

Convert from host galaxy stellar velocity dispersion to black-hole mass.

Parameters

- **vdisp** (*array_like,*) – Host-galaxy velocity dispersion. [cm/s].

- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
mbh – Mass of black hole. [grams]

Return type
array_like,

vdisp_from_mbh(*mbh*, *scatter*)

Convert from black-hole mass to host galaxy stellar velocity dispersion.

Parameters

- **mbh** (*array_like,*) – Mass of black hole. [grams]
- **scatter** (*bool,*) – Whether or not to include scatter in scaling relationship. Uses *self._scatter_dex* attribute.

Returns
vdisp – Host-galaxy velocity dispersion. [cm/s].

Return type
array_like,

class holodeck.relations.**NFW**

Navarro, Frank & White dark-matter density profile from [NFW1997].

static density(*rads: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *mhalo: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *redz: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

NFW DM Density profile.

Parameters

- **rads** (*ArrayLike*) – Target radial distances. [cm]
- **mhalo** (*ArrayLike*) – Halo mass. [grams]
- **redz** (*ArrayLike*) – Redshift. []

Returns
dens – Densities at the given radii. [g/cm^3]

Return type
ArrayLike

static density_characteristic(*mhalo: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *redz: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Return the DM halo parameters for characteristic density.

Parameters

- **mhalo** (*ArrayLike*) – Halo mass. [grams]
- **redz** (*ArrayLike*) – Redshift.

Returns

rho_s – DM halo characteristic density. [g/cm^3]

Return type

ArrayLike

static mass(*rads: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *mhalo: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *redz: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

DM mass enclosed at the given radii from an NFW profile.

Parameters

- **rads** (*ArrayLike*) – Target radial distances. [cm]
- **mhalo** (*ArrayLike*) – Halo mass. [gram]
- **redz** (*ArrayLike*) – Redshift. []

Returns

mass – Mass enclosed within the given radii. [gram]

Return type

ArrayLike

static radius_scale(*mhalo: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *redz: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Return the DM-halo scale radius.

Parameters

- **mhalo** (*ArrayLike*) – Halo mass. [grams]

- **redz** (*ArrayLike*) – Redshift.

Returns
rs – Scale radius of the DM halo. [cm]

Return type
ArrayLike

holodeck.relations.**get_mmbulge_relation**(*mmbulge: _MMBulge_Relation | Type[_MMBulge_Relation] | None = None*) → _MMBulge_Relation

Return a valid Mbh-Mbulge instance.

Parameters
mmbulge (None or (type or instance of *_MMBulge_Relation*),) – If *None*, then a default M-Mbulge relation is returned. Otherwise, the type is checked to make sure it is a valid instance of an *_MMBulge_Relation*.

Returns
Instance of an Mbh-Mbulge relationship.

Return type
_MMBulge_Relation

holodeck.relations.**get_msigma_relation**(*msigma: _MSigma_Relation | Type[_MSigma_Relation] | None = None*) → _MSigma_Relation

Return a valid M-sigma (BH Mass vs. host galaxy velocity dispersion) instance.

Parameters
msigma (None or (class or instance of *_MSigma_Relation*),) – If *None*, then a default M-sigma relation is returned. Otherwise, the type is checked to make sure it is a valid instance of an *_MSigma_Relation*.

Returns
Instance of an Mbh-sigma relationship.

Return type
_MSigma_Relation

holodeck.relations.**get_stellar_mass_halo_mass_relation**(*smhm: _StellarMass_HaloMass | Type[_StellarMass_HaloMass] | None = None*) → _StellarMass_HaloMass

Return a valid Stellar-Mass vs. Halo-Mass relation instance.

Parameters
smhm (None or (type or instance of *_StellarMass_HaloMass*),) – If *None*, then a default relation is returned. Otherwise, the type is checked to make sure it is a valid instance of an *_StellarMass_HaloMass*.

Returns
Instance of an Mbh-Mbulge relationship.

Return type
_StellarMass_HaloMass

CHAPTER

TWENTY

HOLODECK.UTILS

Utility functions and tools.

References

- [Peters1964] Peters 1964
- [Enoki2004] Enoki, Inoue, Nagashima, & Sugiyama 2004
- [Sesana2004] Sesana, Haardt, Madau, & Volonteri 2004
- [EN2007] Enoki & Nagashima 2007

class holodeck.utils.**_Modifier**

Base class for all types of post-processing modifiers.

Notes

- Must be subclassed for use.
- `__call__(base)` ==> `modify(base)`

abstract modify(*base: object*)

Perform an in-place modification on the passed object instance.

Parameters
base (*object*) – The object instance to be modified.

holodeck.utils.**_gw_ecc_func**(*eccen*)

GW Hardening rate eccentricitiy dependence F(e).

See [Peters1964] Eq. 5.6, or [EN2007] Eq. 2.3

Parameters
eccen (*array_like,*) – Binary orbital eccentricity [].

Returns
fe – Eccentricity-dependence term of GW emission [].

Return type
array_like

holodeck.utils.**_integrate_grid_differential_number**(*edges*, *dnum*, *freq=False*)

Integrate the differential number-density of binaries over the given grid (edges).

NOTE: the *edges* provided MUST all be in linear space, mass is converted to `log10(M)` and frequency is converted to `ln(f)`. NOTE: the density *dnum* MUST correspond to *dn/ [dlog10(M) dq dz dln(f)]*

Parameters

- **edges** (*(4,) iterable of ArrayLike*) –
- **dnum** (*ndarray*) –
- **freq** (*bool*) – Whether or not to also integrate the frequency dimension.

Returns

number – Number of binaries in each bin of mass, mass-ratio, redshift, frequency. NOTE: if *freq=False*, then *number* corresponds to *dN/dln(f)*, the number of binaries per log-interval of frequency.

Return type

ndarray

holodeck.utils.**_parse_log_norm_pars**(*vals*, *size*, *default=None*)

Parse/Sanitize the parameters for a log-normal distribution.

- () ==> (N,)
- (2,) ==> (N,) log_normal(vals)
- (N,) ==> (N,)

BUG: this function should probably be deprecated / removed !

Parameters

vals (*object,*) –

Input can be a single value, (2,) array_like, of array_like of size *size*:

- scalar : this value is broadcast to an ndarray of size *size*
- (2,) array_like : these two arguments are passed to *log_normal_base_10* and *size* samples are drawn
- (N,) array_like : if *N* matches *size*, these values are returned.

Returns

vals – Returned values.

Return type

ndarray

holodeck.utils.**_parse_val_log10_val_pars**(*val*, *val_log10*, *val_units=1.0*, *name='value'*, *only_one=True*)

Given either a parameter value, or the log10 of the value, ensure that both are set.

Parameters

- **val** (*array_like or None,*) – The parameter value itself in the desired units (specified by *val_units*).
- **val_log10** (*array_like or None,*) – The log10 of the parameter value in natural units.
- **val_units** (*array_like,*) – The conversion factor from natural units (used in *val_log10*) to the desired units (used in *val*).
- **name** (*str,*) – The name of the variable for use in error messages.

- **only_one** (*bool,*) – Whether one, and only one, of *val* and *val_log10* should be provided (i.e. not *None*).

Returns

- **val** (*array_like,*) – The parameter value itself in desired units. e.g. mass in grams, s.t. mass = Msol * 10^{mass_log10}
- **val_log10** (*array_like,*) – The log10 of the parameter value in natural units. e.g. log10 of mass in solar-masses, s.t. mass = Msol * 10^{mass_log10}

holodeck.utils.**angs_from_sepa**(*sepa*, *dcom*, *redz*)

Calculate angular separation

Parameters

- **sepa** (*ArrayLike*) – Binary separation, in cm
- **dcom** (*ArrayLike*) – Binary comoving distance, in cm
- **redz** (*ArrayLike*) – Binary redshift

Returns
angs – Angular separation

Return type
ArrayLike

holodeck.utils.**char_strain_to_psd**(*freqs*, *hc*)

Parameters

- **freqs** (*array_like*) – Frequencies of interest in [1/sec]. Note: these should NOT be in units of reference frequency, but in units of [Hz] = [1/sec].
- **hc** (*array_like*) – Characteristic strain.

Returns
psd – Power spectral density of gravitational waves.

Return type
array_like

holodeck.utils.**char_strain_to_strain_amp**(*hc*, *fc*, *df*)

Calculate the strain amplitude of single sources given their characteristic strains.

Parameters

- **hc** (*array_like*) – Characteristic strain of the single sources.
- **fc** (*array_like*) – Observed orbital frequency bin centers.
- **df** (*array_like*) – Observed orbital frequency bin widths.

Returns
hs – Strain amplitude of the single sources.

Return type
(F,R,L)

holodeck.utils.**chirp_mass**(*m1*, *m2=None*)

Calculate the chirp-mass of a binary.

Parameters

- **m1** (*array_like,*) – Mass [grams] This can either be the mass of the primary component, if scalar or 1D array_like, or the mass of both components, if 2D array_like, shaped (N, 2).
- **m2** (*None or array_like,*) – Mass [grams] of the other component of the binary. If given, the shape must be broadcastable against *m1*.

Returns
mc – Chirp mass [grams] of the binary.

Return type
array_like,

holodeck.utils.**chirp_mass_mtmr**(*mt*, *mr*)

Calculate the chirp-mass of a binary.

Parameters

- **mt** (*array_like,*) – Total mass [grams]. This is `M = m1+m2`.
- **mr** (*array_like,*) – Mass ratio. `q = m2/m1 <= 1`. This is defined as the secondary (smaller) divided by the primary (larger) mass.

Returns
mc – Chirp mass [grams] of the binary.

Return type
array_like,

holodeck.utils.**deprecated_fail**(*new_func*, *msg=''*, *exc_info=True*)

Decorator for functions that have been deprecated, warn and raise error.

holodeck.utils.**deprecated_pass**(*new_func*, *msg=''*, *exc_info=True*)

Decorator for functions that have been deprecated, warn and pass arguments to new function.

holodeck.utils.**deprecated_warn**(*msg*, *exc_info=True*)

Decorator for functions that will be deprecated, add warning, but still execute function.

holodeck.utils.**dfdt_from_dadt**(*dadt*, *sepa*, *mtot=None*, *frst_orb=None*)

Convert from hardening rate in separation to hardening rate in frequency.

Parameters

- **dadt** (*array_like*) – Hardening rate in terms of binary separation.
- **sepa** (*array_like*) – Binary separations.
- **mtot** (*None or array_like*) – Binary total-mass in units of [gram]. Either *mtot* or *frst_orb* must be provided.
- **frst_orb** (*None or array_like*) – Binary rest-frame orbital-frequency in units of [1/sec]. Either *mtot* or *frst_orb* must be provided.

Returns

- *dfdt* – Hardening rate in terms of rest-frame frequency. [1/sec^2] NOTE: Has the opposite sign as *dadt*.
- *frst_orb* – Orbital frequency, in the rest-frame. [1/sec]

holodeck.utils.**eccen_func**(*cent: float*, *width: float*, *size: int*) → ndarray

Draw random values between [0.0, 1.0] with a given center and width.

This function is a bit contrived, but the *norm* defines the center-point of the distribution, and the *std* parameter determines the width of the distribution. In all cases the resulting values are only between [0.0, 1.0]. This function is typically used to draw initial random eccentricities.

Parameters

- **cent** (*float,*) – Specification of the center-point of the distribution. Range: positive numbers. Values *norm << 1* correspond to small eccentricities, while *norm >> 1* are large eccentricities, with the distribution symmetric around *norm=1.0* (and eccens of 0.5).
- **width** (*float,*) – Specification of the width of the distribution. Specifically how near or far values tend to be from the given central value (*norm*). Range: positive numbers. Note that the 'width' of the distribution depends on the *norm* value, in addition to *std*. Smaller values (typically *std << 1*) produce narrower distributions.
- **size** (*int,*) – Number of samples to draw.

Returns

eccen – Values between [0.0, 1.0] with shape given by the *size* parameter.

Return type

ndarray,

holodeck.utils.**eddington_accretion**(*mass*, *eps=0.1*)

Eddington Accretion rate, $\dot{M}_{Edd} = L_{Edd}/\epsilon c^2$.

Parameters

- **mass** (*array_like of scalar*) – BH Mass.
- **eps** (*array_like of scalar*) – Efficiency parameter.

Returns

mdot – Eddington accretion rate.

Return type

array_like of scalar

holodeck.utils.**fit_powerlaw**(*xx*, *yy*, *init=[-15.0, -0.6666666666666666]*)

Fit the given data with a power-law.

Returns

- *log10_amp*
- *plaw*

holodeck.utils.**fit_powerlaw_fixed_index**(*xx*, *yy*, *index=-0.6666666666666666*, *init=[-15.0]*)

Returns

- *log10_amp*
- *plaw*

holodeck.utils.**fit_turnover_psd**(*xx*, *yy*, *fref*, *init=[-16, -4.333333333333333, 9.506426344208685e-09, 2.5]*)

Parameters

- **xx** (*(F,)*) – Frequencies in units of reference-frequency (e.g. 1/yr)
- **yy** (*(F,)*) – GWB PSD

holodeck.utils.**fobs_from_frst**(*frst*, *redz*)

Calculate observed frequency from rest-frame frequency and redshift.

Parameters

- **frst** (*array_like*) – Rest-frame frequencies.
- **redz** (*array_like*) – Redshifts.

Returns
fobs – Observer-frame frequencies.

Return type
array_like

holodeck.utils.**frac_str**(*vals*, *prec=2*)

Return a string with the fraction and decimal of non-zero elements of the given array.

e.g. [0, 1, 2, 0, 0] ==> "2/5 = 4.0e-1"

Parameters

- **vals** (*(N,) array_like,*) – Input array to find non-zero elements of.
- **prec** (*int*) – Decimal precision in scientific notation string.

Returns
rv – Fraction string.

Return type
str,

holodeck.utils.**frst_from_fobs**(*fobs*, *redz*)

Calculate rest-frame frequency from observed frequency and redshift.

Parameters

- **fobs** (*array_like*) – Observer-frame frequencies.
- **redz** (*array_like*) – Redshifts.

Returns
fobs – Rest-frame frequencies.

Return type
array_like

holodeck.utils.**frst_isco**(*m1*, *m2=0.0*, ***kwargs*)

Get rest-frame orbital frequency of ISCO orbit.

Parameters

- **m1** (*array_like, units of [gram]*) – Total mass, or mass of the primary. Added together with *m2* to get total mass.
- **m2** (*array_like, units of [gram] or None*) – Mass of secondary, or None if *m1* is already total mass.

Returns
fisco

Return type
array_like, units of [Hz]

holodeck.utils.**get_file_size**(*fnames*, *precision=1*)

Return a human-readable size of a file or set of files.

Parameters

- **fnames** (*str or list*) – Paths to target file(s)
- **precisions** (*int,*) – Sesired decimal precision of output

Returns

byte_str – Human-readable size of file(s)

Return type

str

holodeck.utils.**get_scatter_weights**(*uniform_cents*, *dist*)

Get the weights (fractional mass) that should be transferred to each bin to introduce the given scatter.

Parameters

- **uniform_cents** (*(N,) ndarray*) – Uniformly spaced bin-centers specifying distances in the parameter of interest (e.g. mass).
- **dist** (*scipy.stats._distn_infrastructure.rv_continuous_frozen* instance) – Object providing a CDF function *cdf(x)* determining the weights for each bin. e.g. `dist = sp.stats.norm(loc=0.0, scale=0.1)`

Returns

dm – Array of weights for bins with the given distances. [-N+1, -N+2, …, -2, -1, 0, +1, +2, …, +N-2, +N-1]

Return type

(2*N - 1,) ndarray

holodeck.utils.**get_subclass_instance**(*value*, *default*, *superclass*, *allow_none=False*)

Convert the given *value* into a subclass instance.

None ==> instance from *default* class Class ==> instance from that class instance ==> check that this is an instance of a subclass of *superclass*, error if not

Parameters

- **value** (*object,*) – Object to convert into a class instance.
- **default** (*class,*) – Default class constructor to use if *value* is None.
- **superclass** (*class,*) – Super/parent class to compare against the class instance from *value* or *default*. If the class instance is not a subclass of *superclass*, a ValueError is raised.

Returns

value – Class instance that is a subclass of *superclass*.

Return type

object,

:raises ValueError : if the class instance is not a subclass of *superclass*.:

holodeck.utils.**gw_char_strain_nyquist**(*dur_obs*, *hs*, *frst_orb*, *redz*, *dfdt_rst*)

GW Characteristic Strain assuming frequency bins are Nyquist sampled.

Nyquist assumption: the bin-width is equal to 1/T, for T the total observing duration.

See, e.g., [Sesana2004], Eq.35, and surrounding text. NOTE: make sure this is the correct definition of "characteristic" strain for your application!

! THIS FUNCTION MAY NOT BE CORRECT [LZK:2022-08-25] !

Parameters

- **dur_obs** (*array_like,*) – Duration of observations, in the observer frame, in units of [sec]. Typically this is a single float value.
- **hs** (*array_like,*) – Strain amplitude of the source. Dimensionless.
- **frst_orb** (*array_like,*) – Observer-frame orbital frequency, units of [1/sec].
- **redz** (*array_like,*) – Redshift of the binary. Dimensionless.
- **dfdt_rst** (*array_like,*) – Rate of orbital-frequency evolution of the binary, in the rest-frame. Units of [1/sec^2].

Returns
hc – Characteristic strain of the binary.

Return type
array_like,

holodeck.utils.**gw_dade**(*sepa*, *eccen*)

Rate of semi-major axis evolution versus eccentricity, due to GW emission (da/de).

NOTE: returned value is positive (e and a go in same direction). See [Peters1964], Eq. 5.7

Parameters

- **sepa** (*array_like*) – Binary semi-major axis (separation) [grams].
- **eccen** (*array_like*) – Binary eccentricity [grams].

Returns
dade – Rate of change of semi-major axis versus eccentricity [cm]. NOTE: returned value is positive.

Return type
array_like

holodeck.utils.**gw_dedt**(*m1*, *m2*, *sepa*, *eccen*)

GW Eccentricity Evolution rate (de/dt) due to GW emission.

NOTE: returned value is negative.

See [Peters1964], Eq. 5.8

Parameters

- **m1** (*array_like,*) – Mass of one component of the binary [grams].
- **m2** (*array_like,*) – Mass of other component of the binary [grams].
- **sepa** (*array_like,*) – Binary semi-major axis (i.e. separation) [cm].
- **eccen** (*array_like,*) – Binary orbital eccentricity.

Returns
dedt – Rate of eccentricity change of the binary. NOTE: returned value is negative or zero.

Return type
array_like

holodeck.utils.**gw_freq_dist_func**(*nn*, *ee=0.0*, *recursive=True*)

GW frequency distribution function.

See [EN2007] Eq. 2.4; this function gives g(n,e).

NOTE: recursive relation fails for zero eccentricities! TODO: could choose to use non-recursive when zero eccentricities are found?

TODO: replace *ee* variable with *eccen*

Parameters

- **nn** (*int,*) – Number of frequency harmonic to calculate.
- **ee** (*array_like,*) – Binary eccentricity.

Returns
gg – GW Frequency distribution function g(n,e).

Return type
array_like,

holodeck.utils.**gw_hardening_rate_dadt**(*m1*, *m2*, *sepa*, *eccen=None*)

GW Hardening rate in separation (da/dt).

NOTE: returned value is negative.

See [Peters1964], Eq. 5.6

Parameters

- **m1** (*array_like,*) – Mass of one component of the binary [grams].
- **m2** (*array_like,*) – Mass of other component of the binary [grams].
- **sepa** (*array_like,*) – Binary semi-major axis (i.e. separation) [cm].
- **eccen** (*None or array_like,*) – Binary orbital eccentricity. Treated as zero if *None*.

Returns
dadt – Binary hardening rate [cm/s] due to GW emission.

Return type
array_like,

holodeck.utils.**gw_hardening_rate_dfdt**(*m1*, *m2*, *frst_orb*, *eccen=None*)

GW Hardening rate in frequency (df/dt).

Parameters

- **m1** (*array_like*) – Mass of one component of each binary [grams].
- **m2** (*array_like*) – Mass of other component of each binary [grams].
- **freq_orb** (*array_like*) – Rest frame orbital frequency of each binary [1/s].
- **eccen** (*array_like, optional*) – Eccentricity of each binary.

Returns
dfdt – Hardening rate in terms of frequency for each binary [1/s^2].

Return type
array_like,

holodeck.utils.**gw_hardening_timescale_freq**(*mchirp*, *frst*)

GW Hardening timescale in terms of frequency (not separation).

`tau = f_r / (df_r / dt)`, e.g. [EN2007] Eq.2.9

Parameters

- **mchirp** (*array_like,*) – Chirp mass in [grams]
- **frst** (*array_like,*) – Rest-frame orbital frequency [1/s].

Returns
tau – GW hardening timescale defined w.r.t. orbital frequency [sec].

Return type
array_like,

holodeck.utils.**gw_lum_circ**(*mchirp*, *freq_orb_rest*)

Calculate the GW luminosity of a circular binary.

[EN2007] Eq. 2.2

Parameters

- **mchirp** (*array_like,*) – Binary chirp mass [grams].
- **freq_orb_rest** (*array_like,*) – Rest-frame binary orbital frequency [1/s].

Returns
lgw_circ – GW Luminosity [erg/s].

Return type
array_like,

holodeck.utils.**gw_strain_source**(*mchirp*, *dcom*, *freq_rest_orb*)

GW Strain from a single source in a circular orbit.

For reference, see: * [Sesana2004] Eq.36 : they use *f_r* to denote rest-frame GW-frequency. * [Enoki2004] Eq.5.

Parameters

- **mchirp** (*array_like,*) – Binary chirp mass [grams].
- **dcom** (*array_like,*) – Comoving distance to source [cm].
- **freq_orb_rest** (*array_like,*) – Rest-frame binary orbital frequency [1/s].

Returns
hs – GW Strain (*not* characteristic strain).

Return type
array_like,

holodeck.utils.**interp**(*xnew: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *xold: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *yold: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *left: float = nan*, *right: float = nan*, *xlog: bool = True*, *ylog: bool = True*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Linear interpolation of the given arguments in log/lin-log/lin space.

Parameters

- **xnew** (*npt.ArrayLike*) – New locations (independent variable) to interpolate to.
- **xold** (*npt.ArrayLike*) – Old locations of independent variable.
- **yold** (*npt.ArrayLike*) – Old locations of dependent variable.
- **left** (*float, optional*) – Fill value for locations below the domain *xold*.
- **right** (*float, optional*) – Fill value for locations above the domain *xold*.
- **xlog** (*bool, optional*) – Linear interpolation in the log of x values.
- **ylog** (*bool, optional*) – Linear interpolation in the log of y values.

Returns
y1 – Interpolated output values of the dependent variable.

Return type
npt.ArrayLike

holodeck.utils.**isinteger**(*val: object*) → bool

Test if the input value is an integral (integer) number.

Parameters
val (*object*) – Value to test.

Returns
True if the input value is an integer number.

Return type
bool

holodeck.utils.**isnumeric**(*val: object*) → bool

Test if the input value can successfully be cast to a float.

Parameters
val (*object*) – Value to test.

Returns
True if the input value can be cast to a float.

Return type
bool

holodeck.utils.**kepler_freq_from_sepa**(*mass*, *sepa*)

Calculate binary orbital frequency using Kepler's law.

Parameters

- **mass** (*array_like*) – Binary total mass [grams].
- **sepa** (*array_like*) – Binary semi-major axis or separation [cm].

Returns
freq – Binary orbital frequency [1/s].

Return type
array_like

holodeck.utils.**kepler_sepa_from_freq**(*mass*, *freq*)

Calculate binary separation using Kepler's law.

Parameters

- **mass** (*array_like*) – Binary total mass [grams]
- **freq** (*array_like*) – Binary orbital frequency [1/s].

Returns
sepa – Binary semi-major axis (i.e. separation) [cm].

Return type
array_like

holodeck.utils.**lambda_factor_dlnf**(*frst*, *dfdt*, *redz*, *dcom=None*)

Account for the universe's differential space-time volume for a given hardening rate.

For each binary, calculate the factor:

$$\Lambda \equiv (dVc/dz) * (dz/dt) * [dt/dln(f)]$$

, which has units of [Mpc^3]. When multiplied by a number-density [Mpc^-3], it gives the number of binaries in the Universe *per log-frequency interval*. This value must still be multiplied by $\Delta \ln(f)$ to get a number of binaries across a frequency in.

Parameters

- **frst** (*ArrayLike*) – Binary frequency (typically rest-frame orbital frequency; but it just needs to match what's provided in the *dfdt* term. Units of [1/sec].
- **dfdt** (*ArrayLike*) – Binary hardening rate in terms of frequency (typically rest-frame orbital frequency, but it just needs to match what's provided in *frst*). Units of [1/sec^2].
- **redz** (*ArrayLike*) – Binary redshift. Dimensionless.
- **dcom** (*ArrayLike*) – Comoving distance to binaries (for the corresponding redshift, *redz*). Units of [cm]. If not provided, calculated from given *redz*.

Returns
lambda_fact – The differential comoving volume of the universe per log interval of binary frequency.

Return type
ArrayLike

holodeck.utils.**load_hdf5**(*fname*, *keys=None*)

Load data and header information from HDF5 files into dictionaries.

Parameters

- **fname** (*str*) – Filename to load (must be an *hdf5* file).
- **keys** (*None or (list of str)*) – Specific keys to load from the top-level of the HDF5 file. *None*: load all top-level keys.

Returns

- **header** (*dict,*) – All entries from *hdf5.File.attrs*, typically used for meta-data.
- **data** (*dict,*) – All top level datasets from the hdf5 file, specifically everything returned from *hdf5.File.keys()*.

holodeck.utils.**log_normal_base_10**(*mu: float*, *sigma: float*, *size: int | List[int] | None = None*, *shift: float = 0.0*) → ndarray

Draw from a log-normal distribution using base-10 standard-deviation.

i.e. the *sigma* argument is in "dex", or powers of ten.

Parameters

- **mu** (*float*) – Mean value of the distribution.
- **sigma** (*float*) – Standard deviation in dex (i.e. powers of ten). *sigma=1.0* means a standard deviation of one order of magnitude around mu.
- **size** (*Union[int, list[int]], optional*) – Number of values to draw. Either a single integer, or a tuple of integers describing a shape.
- **shift** (*float, optional*) –

Returns
dist – Resulting distribution values.

Return type
npt.ArrayLike

holodeck.utils.**m1m2_from_mtmr**(*mt: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *mr: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Convert from total-mass and mass-ratio to individual masses.

Parameters

- **mt** (*array_like*) – Total mass of the binary.
- **mr** (*array_like*) – Mass ratio of the binary.

Returns
Primary and secondary masses respectively. 0-primary (more massive component), 1-secondary (less massive component)

Return type
(2,N) ndarray

holodeck.utils.**minmax**(*vals: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *filter: bool = False*) → ndarray

Find the minimum and maximum values in the given array.

Parameters

- **vals** (*npt.ArrayLike*) – Input values in which to find extrema.
- **filter** (*bool, optional*) – Select only finite values from the input array.

Returns
extr – Minimum and maximum values.

Return type
(2,) np.ndarray

holodeck.utils.**mtmr_from_m1m2**(*m1*, *m2=None*)

Convert from primary and secondary masses into total-mass and mass-ratio.

NOTE: it doesn't matter if *m1* or *m2* is the primary or secondary.

Parameters

- **m1** (*array_like,*) – Mass. If this is a single value, or a 1D array, it denotes the mass of one component of a binary. It can also be shaped, (N,2) where the two elements are the two component masses.
- **m2** (*None or array_like,*) – If array_like, it must match the shape of *m1*, and corresponds to the companion mass.

Returns
Total mass and mass-ratio. If the input values are floats, this is just shaped (2,).

Return type
(2,N) ndarray

holodeck.utils.**ndinterp**(*xx*, *xvals*, *yvals*, *xlog=False*, *ylog=False*)

Interpolate 2D data to an array of points.

xvals and *yvals* are (N, M) where the interpolation is done along the 1th (*M*) axis (i.e. interpolation is done independently for each *N* row. Should be generalizeable to higher dim.

Parameters

- **xx** (*(T,) or (N, T) ndarray*) – Target x-values to interpolate to.
- **xvals** (*(N, M) ndarray*) – Evaluation points (x-values) of the functions to be interpolated. Interpolation is performed over the 1th (last) axis. NOTE: values *must* be monotonically increasing along axis=1 !
- **yvals** (*(N, M) ndarray*) – Function values (y-values) of the function to be interpolated. Interpolation is performed over the 1th (last) axis.

Returns
ynew – Interpolated function values, for each of N functions and T evaluation points.

Return type
(N, T) ndarray

holodeck.utils.**nyquist_freqs**(*dur*, *cad*)

DEPRECATED. Use *holodeck.utils.pta_freqs* instead.

holodeck.utils.**print_stats**(*stack=True*, *print_func=<built-in function print>*, ***kwargs*)

Print out basic properties and statistics on the input key-value array_like values.

Parameters

- **stack** (*bool,*) – Whether or not to print a backtrace to stdout.
- **print_func** (*callable,*) – Function to use for returning/printing output.
- **kwargs** (*dict,*) – Key-value pairs where values are array_like for the shape/stats to be printed.

holodeck.utils.**pta_freqs**(*dur=505868328.00000006*, *num=40*, *cad=None*)

Get Fourier frequency bin specifications for the given parameters.

Parameters

- **dur** (*float,*) – Total observing duration, which determines the minimum sensitive frequency, `1/dur`. Typically *dur* should be given in units of [sec], such that the returned frequencies are in units of [1/sec] = [Hz]
- **num** (*int,*) – Number of frequency bins. If *cad* is not None, then the number of frequency bins is determined by *cad* and the *num* value is disregarded.

- **cad** (float or *None*,) – Cadence of observations, which determines the maximum sensitive frequency (i.e. the Nyquist frequency). If *cad* is not given, then *num* frequency bins are constructed.

Returns

- **cents** (*(F,) ndarray*) – Bin-center frequencies for *F* bins. The frequency bin centers are at: `F_i = (i + 1.5) / dur` for i between 0 and *num-1*. The number of frequency bins, *F* is the argument *num*, or determined by *cad* if it is given.
- **edges** (*(F+1,) ndarray*) – Bin-edge frequencies for *F* bins, i.e. *F+1* bin edges. The frequency bin edges are at: `F_i = (i + 1) / dur` for i between 0 and *num*. The number of frequency bins, *F* is the argument *num*, or determined by *cad* if it is given.

holodeck.utils.**python_environment**()

Tries to determine the current python environment, one of: 'jupyter', 'ipython', 'terminal'.

Returns
Description of the current python environment, one of ['jupter', 'ipython', 'terminal'].

Return type
str

holodeck.utils.**quantiles**(*values: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *percs: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | None = None*, *sigmas: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | None = None*, *weights: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | None = None*, *axis: int | None = None*, *values_sorted: bool = False*, *filter: str | None = None*) → ndarray | MaskedArray

Compute weighted percentiles.

NOTE: if *values* is a masked array, then only unmasked values are used!

Parameters

- **values** (*(N,)*) – input data
- **percs** (*(M,) scalar*) – Desired quantiles of the data. Within range of [0.0, 1.0].
- **weights** ((N,) or *None*) – Weights for each input data point in *values*.
- **axis** (int or *None*,) – Axis over which to calculate quantiles.
- **values_sorted** (*bool*) – If True, then input values are assumed to already be sorted. Otherwise they are sorted before calculating quantiles (for efficiency).

Returns
percs – Array of quantiles of the input data.

Return type
(M,) float

holodeck.utils.**rad_isco**(*m1*, *m2=0.0*, *factor=3.0*)

Inner-most Stable Circular Orbit, radius at which binaries 'merge'.

ENH: allow single (total) mass argument. ENH: add function to calculate factor as a function of BH spin.

Parameters

- **m1** (*array_like,*) – Mass of first (either) component of binary [grams].
- **m2** (*array_like,*) – Mass of second (other) component of binary [grams].
- **factor** (*float,*) – Factor by which to multiple the Schwarzschild radius to define the ISCO. 3.0 for a non-spinning black-hole.

Returns
rs – Radius of the inner-most stable circular orbit [cm].

Return type
array_like,

holodeck.utils.**redz_after**(*time*, *redz=None*, *age=None*)

Calculate the redshift after the given amount of time has passed.

Parameters

- **time** (*array_like in units of [sec]*) – Amount of time to pass.
- **redz** (*None or array_like,*) – Redshift of starting point after which *time* is added.
- **age** (*None or array_like, in units of [sec]*) – Age of the Universe at the starting point, after which *time* is added.

Returns
new_redz – Redshift of the Universe after the given amount of time.

Return type
array_like

holodeck.utils.**rk4_step**(*func*, *x0*, *y0*, *dx*, *args=None*, *check_nan=0*, *check_nan_max=5*)

Perform a single 4th-order Runge-Kutta integration step.

holodeck.utils.**roll_rows**(*arr*, *roll_num*)

Roll each row (axis=0) of the given array by an amount specified.

Parameters

- **arr** (*(R, D) ndarray*) – Input data to be rolled.
- **roll_num** (*(R,) ndarray of int*) – Amount to roll each row. Must match the number of rows (axis=0) in *arr*.

Returns
result – Rolled version of the input data.

Return type
(R, D) ndarray

Example

```
>>> a = np.arange(12).reshape(3, 4); b = [1, -1, 2]; utils.roll_rows(a, b)
array([[ 3,  0,  1,  2],
       [ 5,  6,  7,  4],
       [10, 11,  8,  9]])
```

holodeck.utils.**scatter_redistribute_densities**(*cents*, *dens*, *dist=None*, *scatter=None*, *axis=0*)

Redistribute *dens* across the target axis to account for scatter/variance.

Parameters

- **cents** (*(N,) ndarray*) – Locations of bin centers in the parameter of interest.
- **dist** (*scipy.stats._distn_infrastructure.rv_continuous_frozen* instance) – Object providing a CDF function *cdf(x)* determining the weights for each bin. e.g. `dist = sp.stats.norm(loc=0.0, scale=0.1)`
- **dens** (*ndarray*) – Input values to be redistributed. Must match the size of *cents* along axis *axis*.

Returns

dens_new – Array with resitributed values. Same shape as input *dens*.

Return type

ndarray

holodeck.utils.**schwarzschild_radius**(*mass*)

Return the Schwarschild radius [cm] for the given mass [grams].

Parameters

m1 (*array_like*) – Mass [grams]

Returns

rs – Schwzrschild radius for this mass.

Return type

array_like,

holodeck.utils.**sep_to_merge_in_time**(*m1*, *m2*, *time*)

The initial separation required to merge within the given time.

See: [Peters1964]

Parameters

- **m1** (*array_like,*) – Mass of one component of the binary [grams].
- **m2** (*array_like,*) – Mass of other component of the binary [grams].
- **time** (*array_like,*) – The duration of time of interest [sec].

Returns

Initial binary separation [cm].

Return type

array_like

holodeck.utils.**stats**(*vals: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *percs: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | None = None*, *prec: int = 2*, *weights=None*) → str

Return a string giving quantiles of the given input data.

Parameters

- **vals** (*npt.ArrayLike,*) – Input values to get quantiles of.
- **percs** (*npt.ArrayLike, optional*) – Quantiles to calculate.
- **prec** (*int, optional*) – Precision in scientific notation of output.

Returns
rv – Quantiles of input formatted as a string of scientific notation values.

Return type
str

Raises
TypeError – raised if input data is not iterable.:

holodeck.utils.**time_to_merge_at_sep**(*m1*, *m2*, *sepa*)

The time required to merge starting from the given initial separation.

See: [Peters1964].

Parameters

- **m1** (*array_like,*) – Mass of one component of the binary [grams].
- **m2** (*array_like,*) – Mass of other component of the binary [grams].
- **sepa** (*array_like,*) – Binary semi-major axis (i.e. separation) [cm].

Returns
Duration of time for binary to coalesce [sec].

Return type
array_like

holodeck.utils.**tqdm**(**args*, ***kwargs*)

Construct a progress bar appropriately based on the current environment (script vs. notebook)

Parameters

- ***args** (All arguments are passed directory to the *tqdm* constructor.) –
- ****kwargs** (All arguments are passed directory to the *tqdm* constructor.) –

Returns
Decorated iterator that shows a progress bar.

Return type
tqdm.tqdm_gui

holodeck.utils.**trapz**(*yy: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *xx: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *axis: int = -1*, *cumsum: bool = True*)

Perform a cumulative integration along the given axis.

Parameters

- **yy** (*ArrayLike of scalar,*) – Input to be integrated.
- **xx** (*ArrayLike of scalar,*) – The sample points corresponding to the *yy* values. This must either be shaped as * the same number of dimensions as *yy*, with the same length along the *axis* dimension, or * 1D with length matching *yy[axis]*
- **axis** (*int,*) – The axis over which to integrate.

Returns

ct – Cumulative trapezoid rule integration.

Return type

ndarray of scalar,

holodeck.utils.**trapz_loglog**(*yy: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *xx: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *bounds: Tuple[float, float] | None = None*, *axis: int = -1*, *dlogx: float | None = None*, *lntol: float = 0.01*, *cumsum: bool = True*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Calculate integral, given $y = dA/dx$ or $y = dA/d\log x$ w/ trapezoid rule in log-log space.

We are calculating the integral A given sets of values for y and x. To associate *yy* with dA/dx then *dlogx* = *None* [default], otherwise, to associate *yy* with $dA/d\log x$ then *dlogx* = *True* for natural-logarithm, or *dlogx* = *b* for a logarithm of base *b*.

For each interval (x[i+1], x[i]), calculate the integral assuming that y is of the form,

$y = a * x^{gamma}$

Parameters

- **yy** (*ndarray*) –
- **xx** (*(X,) array_like of scalar,*) –
- **bounds** (*(2,) array_like of scalar,*) –
- **axis** (*int,*) –
- **dlogx** (*scalar or None,*) –
- **lntol** (*scalar,*) –

Return type

integ

Notes

- When bounds are given that are not identical to input *xx* values, then interpolation must be performed. This can be done on the resulting cumsum'd values, or on the input integrand values. The cumsum values are *not necessarily a power-law* (for negative indices), and thus the interpolation is better performed on the input *yy* values.
- Interpolating the cumulative-integral works very badly, instead interpolate the x/y values initially to obtain the integral at the appropriate locations.

CHAPTER

TWENTYONE

GETTING STARTED

(1) Read the `holodeck` *getting started* guide.
(2) Install `holodeck` following the *installation* instructions below.
(3) Explore the package demonstration notebooks.
(4) Read the Development & Contributions guide.

CHAPTER

TWENTYTWO

INSTALLATION

The holodeck framework is currently under substantial, active development. Recent versions will not generally be available with pip or conda install. Currently holodeck requires python >= 3.9 (tests are run on versions 3.9, 3.10, 3.11). The recommended installation is:

0) OPTIONAL & recommended: create and activate a new **anaconda** environment to isolate your build:

```
conda create --name holo311 python=3.11; conda activate holo311
```

Note that you will need to activate this environment every time you want to use holodeck. If you're not familiar with **anaconda**, take a look at their official Getting started guide. To use your anaconda environment with jupyter notebooks, make sure to add this environment to your ipython kernels:

```
conda install -c conda-forge ipykernel
python -m ipykernel install --user --name=holo311
```

1) Clone the holodeck repository, and move into the repo directory:

```
git clone https://github.com/nanograv/holodeck.git; cd holodeck
```

2) Install the required external packages specified in the requirements file:

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

OPTIONAL: install development requirements:

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

3) Build the required c libraries from holodeck cython code:

```
python setup.py build_ext -i
```

4) Perform a development/editable local installation:

```
python setup.py develop
```

The 'editable' installation allows the code base to be modified, and have those changes take effect when using the holodeck module without having to rebuild/reinstall it. Note that any changes to the cython library files do still require a rebuild by running steps (3) and (4) above.

22.1 MPI

For some scripts (particularly for generating libraries), an MPI implementation is required (e.g. `openmpi`), along with the mpi4py package. This is not included as a requirement in the `requirements.txt` file as it significantly increases the installation complexity, and is not needed for many `holodeck` use cases. If you already have an MPI implementation installed on your system, you should be able to install `mpi4py` with anaconda: `conda install mpi4py`. To see if you have `mpi4py` installed, run `python -c 'import mpi4py; print(mpi4py.__version__)'` from a terminal.

macos users: if you are using homebrew on macos, you should be able to simply run: `brew install mpi4py` which will include the required openmpi implementation.

CHAPTER

TWENTYTHREE

ATTRIBUTION & REFERENCING

The `holodeck` package uses an MIT license.

A dedicated paper on `holodeck` is currently in preparation, but the package is also described in the recent astrophysics analysis from the NANOGrav 15yr dataset.

```
@ARTICLE{2023ApJ...952L..37A,
      author = {{Agazie}, Gabriella and {et al} and {Nanograv Collaboration}},
      title = "{The NANOGrav 15 yr Data Set: Constraints on Supermassive Black Hole␣
↪Binaries from the Gravitational-wave Background}",
      journal = {\apjl},
         year = 2023,
      month = aug,
      volume = {952},
      number = {2},
         eid = {L37},
      pages = {L37},
         doi = {10.3847/2041-8213/ace18b},
archivePrefix = {arXiv},
      eprint = {2306.16220},
primaryClass = {astro-ph.HE},
      adsurl = {https://ui.adsabs.harvard.edu/abs/2023ApJ...952L..37A},
}
```

CHAPTER

TWENTYFOUR

INDICES AND TABLES

- genindex
- search
- modindex

BIBLIOGRAPHY

[N15GWB] Agazie et al. (2023), ApJL, 951, 1. The NANOGrav 15 yr Data Set: Evidence for a Gravitational-wave Background https://ui.adsabs.harvard.edu/abs/2023ApJ...951L...8A

[N15astro] Agazie et al. (2023), ApJL, 952, 2. The NANOGrav 15 yr Data Set: Constraints on Supermassive Black Hole Binaries from the Gravitational-wave Background https://ui.adsabs.harvard.edu/abs/2023ApJ...952L..37A

[N15NP] Afzal et al. (2023), ApJL, 951, 1. The NANOGrav 15 yr Data Set: Search for Signals from New Physics https://ui.adsabs.harvard.edu/abs/2023ApJ...951L..11A

[N15data] Agazie et al. (2023), ApJL, 951, 1. The NANOGrav 15 yr Data Set: Observations and Timing of 68 Millisecond Pulsars https://ui.adsabs.harvard.edu/abs/2023ApJ...951L...9A

[N15anisotropy] Agazie et al. (2023), ApJL, 956, 1. The NANOGrav 15 yr Data Set: Search for Anisotropy in the Gravitational-wave Background https://ui.adsabs.harvard.edu/abs/2023ApJ...956L...3A

[N15CWs] Agazie et al. (2023), ApJL, 951, 2. The NANOGrav 15 yr Data Set: Bayesian Limits on Gravitational Waves from Individual Supermassive Black Hole Binaries https://ui.adsabs.harvard.edu/abs/2023ApJ...951L..50A

[N15detchar] Agazie et al. (2023), ApJL, 951, 1. The NANOGrav 15 yr Data Set: Detector Characterization and Noise Budget https://ui.adsabs.harvard.edu/abs/2023ApJ...951L..10A

[Behroozi2013] : Behroozi, Wechsler & Conroy 2013. ApJ, 770, 1. The Average Star Formation Histories of Galaxies in Dark Matter Halos from z = 0-8 https://ui.adsabs.harvard.edu/abs/2013ApJ...770...57B/abstract

[BBR1980] Begelman, Blandford & Rees 1980. Nature, 287, 5780. Massive black hole binaries in active galactic nuclei. https://ui.adsabs.harvard.edu/abs/1980Natur.287..307B/abstract

[Chen2017] Chen, Sesana, & Del Pozzo 2017 Efficient computation of the gravitational wave spectrum emitted by eccentric massive black hole binaries in stellar environments https://ui.adsabs.harvard.edu/abs/2017MNRAS.470.1738C/abstract

[Chen2019] Chen, Sesana, Conselice 2019. MNRAS, 488, 1. Constraining astrophysical observables of galaxy and supermassive black hole binary mergers using pulsar timing arrays https://ui.adsabs.harvard.edu/abs/2019MNRAS.488..401C/abstract

[EN2007] Enoki & Nagashima 2007. PTP, 117, 2. astro-ph/0609377. The Effect of Orbital Eccentricity on Gravitational Wave Background Radiation from Supermassive Black Hole Binaries https://ui.adsabs.harvard.edu/abs/2007PThPh.117..241E/abstract

[Enoki2004] Enoki, Inoue, Nagashima, & Sugiyama 2004. ApJ, 615, 1. astro-ph/0404389. Gravitational Waves from Supermassive Black Hole Coalescence in a Hierarchical Galaxy Formation Model https://ui.adsabs.harvard.edu/abs/2004ApJ...615...19E/abstract

[Genel2014] : Genel et al. (2014), MNRAS, 445, 1. Introducing the Illustris project: the evolution of galaxy populations across cosmic time https://ui.adsabs.harvard.edu/abs/2014MNRAS.445..175G

[Guo2010] Guo, White, Li & Boylan-Kolchin 2010. MNRAS, 404, 3. How do galaxies populate dark matter haloes? https://ui.adsabs.harvard.edu/abs/2010MNRAS.404.1111G/abstract

[WMAP9] Hinshaw, Larson, Komatsu et al. 2013. ApJS, 208, 2. (1212.5226). Nine-year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Cosmological Parameter Results. https://ui.adsabs.harvard.edu/abs/2013ApJS..208...19H/abstract

[Hogg1999] Hogg 1999. arXiv. (astro-ph/9905116). Distance measures in cosmology. https://ui.adsabs.harvard.edu/abs/1999astro.ph..5116H

[Kelley2017a] Kelley, Blecha, and Hernquist (2017), MNRAS, 464, 3. Massive black hole binary mergers in dynamical galactic environments https://ui.adsabs.harvard.edu/abs/2017MNRAS.464.3131K

[Kelley2017b] Kelley et al. (2017), MNRAS, 471, 4. The gravitational wave background from massive black hole binaries in Illustris: spectral features and time to detection with pulsar timing arrays https://ui.adsabs.harvard.edu/abs/2017MNRAS.471.4508K

[Kelley2018] Kelley et al. (2018), MNRAS, 477, 1. Single sources in the low-frequency gravitational wave sky: properties and time to detection by pulsar timing arrays https://ui.adsabs.harvard.edu/abs/2018MNRAS.477..964K

[Klypin2016] : Klypin, Yepes, Gottlöber, et al. 2016. MNRAS, 457, 4. MultiDark simulations: the story of dark matter halo concentrations and density profiles https://ui.adsabs.harvard.edu/abs/2016MNRAS.457.4340K/abstract

[KH2013] Kormendy & Ho 2013. ARAA, 51, 1. Coevolution (Or Not) of Supermassive Black Holes and Host Galaxies https://ui.adsabs.harvard.edu/abs/2013ARA%26A..51..511K/abstract

[Leja2020] Leja et al. (2020), ApJ, 893, 2. A New Census of the 0.2 < z < 3.0 Universe. I. The Stellar Mass Function https://ui.adsabs.harvard.edu/abs/2020ApJ...893..111L

[MM2013] McConnell & Ma 2013. ApJ, 764, 2. Revisiting the Scaling Relations of Black Hole Masses and Host Galaxy Properties https://ui.adsabs.harvard.edu/abs/2013ApJ...764..184M/abstract

[NFW1997] Navarro, Frenk & White 1997. ApJ, 490, 2. A Universal Density Profile from Hierarchical Clustering https://ui.adsabs.harvard.edu/abs/1997ApJ...490..493N/abstract

[Nelson2015] Nelson et al. (2015), A&C, 13,. The illustris simulation: Public data release https://ui.adsabs.harvard.edu/abs/2015A&C....13...12N

[Peters1964] Peters 1964. PR, 136, 4B. Gravitational Radiation and the Motion of Two Point Masses https://ui.adsabs.harvard.edu/abs/1964PhRv..136.1224P/abstract

[Phinney2001] Phinney 2001. arXiv. (astro-ph/0108028). A Practical Theorem on Gravitational Wave Backgrounds. https://ui.adsabs.harvard.edu/abs/2001astro.ph..8028P/abstract

[Quinlan1996] Quinlan 1996 The dynamical evolution of massive black hole binaries I. Hardening in a fixed stellar background https://ui.adsabs.harvard.edu/abs/1996NewA....1...35Q/abstract

[Rodriguez-Gomez2015] : Rodriguez-Gomez et al. (2015), MNRAS, 449, 1. The merger rate of galaxies in the Illustris simulation: a comparison with observations and semi-empirical models https://ui.adsabs.harvard.edu/abs/2015MNRAS.449...49R

[Sesana2004] Sesana, Haardt, Madau, & Volonteri 2004. ApJ, 611, 2. astro-ph/0401543. Low-Frequency Gravitational Radiation from Coalescing Massive Black Hole Binaries in Hierarchical Cosmologies http://adsabs.harvard.edu/abs/2004ApJ...611..623S

[Sesana2006] Sesana, Haardt & Madau et al. 2006 Interaction of Massive Black Hole Binaries with Their Stellar Environment. I. Ejection of Hypervelocity Stars https://ui.adsabs.harvard.edu/abs/2006ApJ...651..392S/abstract

[Sesana2008] Sesana, Vecchio, Colacino 2008. MNRAS, 390, 1. (0804.4476). The stochastic gravitational-wave background from massive black hole binary systems: implications for observations with Pulsar Timing Arrays. https://ui.adsabs.harvard.edu/abs/2008MNRAS.390..192S/abstract

[Sesana2010] Sesana 2010 Self Consistent Model for the Evolution of Eccentric Massive Black Hole Binaries in Stellar Environments: Implications for Gravitational Wave Observations https://ui.adsabs.harvard.edu/abs/2010ApJ...719..851S/abstract

[Sijacki2015] Sijacki et al. (2015), MNRAS, 452, 1. The Illustris simulation: the evolving population of black holes across cosmic time https://ui.adsabs.harvard.edu/abs/2015MNRAS.452..575S

[Siwek2023] Siwek, Weinberger, and Hernquist (2023), MNRAS, 522, 2. Orbital evolution of binaries in circumbinary discs https://ui.adsabs.harvard.edu/abs/2023MNRAS.522.2707S

[Springel2010] Springel (2010), MNRAS, 401, 2. E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh https://ui.adsabs.harvard.edu/abs/2010MNRAS.401..791S

[Vogelsberger2014] Vogelsberger et al. (2014), MNRAS, 444, 2. Introducing the Illustris Project: simulating the coevolution of dark and visible matter in the Universe https://ui.adsabs.harvard.edu/abs/2014MNRAS.444.1518V

PYTHON MODULE INDEX

h

INDEX

Symbols

G

H

N

P

Q

R

S

T

V

Y

Z