Semi-Recurrent Neural Networks In IllutrisTNG

Documentation

H. G. Chittenden

October 27, 2022

Contents

Data Files

Separate hdf5 files exist for data used in the central and satellite neural networks, and observable quantities derived from the test and predicted data.

The following documentation lists all arguments per file, giving their name, shape and units in the following format:

name (shape) [units]

A field labelled with a dagger † is logarithmic with base 10.

Files are also separated by one of the following four suffixes: *_train*, *_test*, *_pred* and *_dark*. These indicate whether they belong to the training, test, predicted or dark simulation data. All fields representing one quantity have the same shape, except for the sample axis, which is axis 0 in all circumstances. In training and test data files, all baryonic quantities are derived from the TNG simulations, whereas prediction and dark files contain output data from the neural networks.

Any field labelled with an asterisk * does not change between files. These exceptions typically represent domains common to certain data, such as time or wavelength.

Each section below documents two separate groups of files, for central and satellite galaxy data. The characters C and S following units indicate fields which are only present in each respective file. If not present, the field exists in both files.

1 Network Data

The data used in the neural networks are contained in the files **central_nndata.h5** (216.3 MB) and **satellite_nndata.h5** (539.9 MB).

Note that different definitions of final stellar mass and metallicity exist in this work: the relevant fields in the Subfind merger trees and the values numerically derived from our galaxy formation histories. Similar yet distinct arguments exist for each of these fields, and are given below.

a_inf (n_samples,) [ ] S Scaled infall time.

a_max (n_samples,) [ ] S Scaled formation time.

beta (n_samples,) [log Gyr^{-2}] C Specific mass accretion gradient.

beta_halo (n_samples,) [log Gyr^{-2}] S Specific mass accretion gradient of the FoF host.

beta_sub (n_samples,) [log Gyr^{-2}] S Specific mass accretion gradient of the satellite subhalo.

centreofmass (n_samples, n_timesteps, n_dimensions) [Mpc] Centre of mass of target subhalo.

d_minima † (n_samples,) [kpc] C Distance to nearest void (minimum) in the cosmic web at $z = 0$.

d_node † (n_samples,) [kpc] C Distance to nearest node (maximum) in the cosmic web at $z = 0$.

d_saddle_1 † (n_samples,) [kpc] C Distance to nearest saddle point in the cosmic web at $z = 0$, with one minimised dimension.

d_saddle_2 † (n_samples,) [kpc] C Distance to nearest saddle point in the cosmic web at $z = 0$, with two minimised dimensions.

d_skel † (n_samples,) [kpc] C Distance to nearest filament in the cosmic web at $z = 0$.

delta (n_samples, n_timesteps) [ ] S DM overdensity within 1Mpc spherical volume.

delta_1 (n_samples, n_timesteps) [ ] C DM overdensity within 1Mpc spherical volume.

delta_3 (n_samples, n_timesteps) [ ] C DM overdensity within 3Mpc spherical volume.

delta_5 (n_samples, n_timesteps) [ ] C DM overdensity within 5Mpc spherical volume.

F_SFH (n_samples, n_frequencies) [Gyr/$M_\odot$] Fourier amplitude (square root of power spectrum) of star formation history.

F_ZH (n_samples, n_frequencies) [$Z_\odot^{-1}$] Fourier amplitude of metallicity history.

Fph_SFH (n_samples, n_frequencies) [rad] Fourier phase of star formation history.

Fph_ZH (n_samples, n_frequencies) [rad] Fourier phase of metallicity history.

formtime (n_samples,) [Gyr] Time of germination of the halo.

frequency * (n_frequencies,) [Gyr^{-1}] Frequency domain of all Fourier data.

ID (n_samples,) [ ] SubfindID in TNG simulation.

logmaxMhdot † (n_samples,) [$M_\odot$/Gyr] Maximum absolute FoF mass accretion rate.

logmaxmhdot † (n_samples,) [$M_\odot$/Gyr] S Maximum absolute subhalo mass accretion rate.

logMh † (n_samples,) [$M_\odot$] FoF halo mass at $z = 0$.

logmh † (n_samples,) [$M_\odot$] S Subfind subhalo mass at $z = 0$.

logMs † (n_samples,) [$M_\odot$] Subfind stellar mass at $z = 0$.

logMS † (n_samples,) [$M_\odot$] Numerical stellar mass at $z = 0$.

logMu † (n_samples,) [ ] S Subhalo to host mass ratio at time of infall.

logVrel † (n_samples,) [km/s] S Subhalo velocity relative to host at time of infall.

logZ † (n_samples,) [$Z_\odot$] Subfind stellar metallicity at $z = 0$.

logZS † (n_samples,) [$Z_\odot$] Numerical stellar metallicity at $z = 0$.

lookback_time * (n_timesteps,) [Gyr] Lookback time domain of all temporal data.

Mhdot (n_samples, n_timesteps) [$M_\odot$/Gyr] Halo mass accretion rate.

mhdot (n_samples, n_timesteps) [$M_\odot$/Gyr] S Subhalo mass accretion rate.

minD (n_samples, n_timesteps) [Mpc] Distance to nearest external subhalo.

mwa (n_samples,) [Gyr] Stellar mass weighted age (numerical).

mwsa (n_samples,) [Gyr] Stellar mass weighted age (NN target).

redshift * (n_timesteps,) [ ] Redshift domain of all temporal data.

rhalf (n_samples, n_timesteps) [Mpc] Dark matter half-mass radius.

SFH (n_samples, n_timesteps) [$M_\odot$/Gyr] Star formation history.

simu (n_samples,) [ ] TNG simulation number. Can be 100 or 300.

skew (n_samples, n_timesteps) [ ] Radial skew of dark matter distribution, up to 3Mpc for central galaxy data, 1Mpc for satellite galaxy data.

time * (n_timesteps,) [Gyr] Cosmic time domain of all temporal data.

vcirc (n_samples, n_timesteps) [$\sqrt{M_\odot/\mathrm{Gyr}}$] Proxy for circular orbital velocity.

ZH (n_samples, n_timesteps) [$Z_\odot$] Stellar metallicity history.

2 Observational Data

The observational data derived from test data and predictions are contained in the files **central_obsdata.h5** (771.6 MB) and **satellite_obsdata.h5** (2.0 GB).

Halpha (n_samples,) [$L_\odot$] Hα line luminosity.
ID * (n_samples,) [ ] SubfindID in TNG simulation.
mag_u (n_samples,) [ ] SDSS u band AB magnitude.
mag_g (n_samples,) [ ] SDSS g band AB magnitude.
mag_r (n_samples,) [ ] SDSS r band AB magnitude.
mag_i (n_samples,) [ ] SDSS i band AB magnitude.
mag_z (n_samples,) [ ] SDSS z band AB magnitude.
simu * (n_samples,) [ ] TNG simulation number. Can be 100 or 300.
spec (n_samples, n_wavelengths) [$L_\odot$/Å] Spectral energy distribution.
wave * (n_wavelengths,) [Å] Wavelength domain of spectra.

Python Modules

This code consists of a single base module which can be run in multiple environments and contains simple or general purpose functions and data. Additional modules, such as the neural network module, must be run in separate environments, and import all functions and data from the base module.

The description of each function lists variables under three categories: Input, Prints and Returns, describing their role in the function. Each argument has its valid data types specified (e.g. *float*, *bool*, *str*). If applicable, a subscript integer indicates the number of dimensions of the variable, where 0 implies this is a single number, 1 implies this is a 1D list/array, etc., and subscript x implies that any dimensionality is valid. An asterisk * indicates that the given input argument is optional.

e.g.

***float*$_{0,1}$ a *** is an optional argument named *a*, of dtype *float*, which can either be a 1D array or a single number.

***str* b** is a required string argument named *b*.

1 Base Module

The base module contains general purpose functions which do not require packages specific to certain python environments, or would be applied in multiple environments.

1.1 ABmag(x, y, Band, D=10, ModelB=False, flambda=True, spline='linear', Nint=100000, xlog=False, ylog=False)

Computes AB band magnitudes by integrating a converted† spectrum over a specified bandpass filter. Optionally applies simple dust attenuation and luminosity distance. Supports the five SDSS 'ugriz' bandpasses.

† Input spectrum flux density converted to maggies, for compatibility with the AB magnitude system.

Input

***float*$_1$ x** Wavelength domain of input spectrum/spectra, in Ångströms. If not in ascending order, x and y are automatically reordered.

***float*$_{1,2}$ y** Luminosity density of input spectrum. Can be a single 1D spectrum or 2D array of shape (n_samples, n_wavelengths). Units specified by *flambda* field.

***str/list* Band** One of five strings deciding the SDSS bandpass over which the spectrum is integrated. Can be 'u', 'g', 'r', 'i' or 'z'. If this is a list of many such strings, each band magnitude is computed in the order given.

***float*$_{0,1}$ D *** Luminosity distance, in parsecs. AB flux is scaled according to $\left(\frac{D}{10\text{pc}}\right)^{-2}$. Default is 10 parsecs, i.e. magnitudes are by default absolute. If multiple values given, multiple magnitudes returned.

***bool* ModelB *** If True, the flux is attenuated according to dust model B from Nelson et al. Default is False.

***bool* flambda *** If True, assumes the units of the spectrum to be in solar luminosities per Ångström. If False, this is instead solar luminosities per Hertz. Default is True.

***str/int*$_0$ spline *** Identical to the *kind* argument in scipy.interpolate.interp1d: specifies the type of spline with which the spectra and bandpass filters are interpolated, or the order of the spline if an integer is given. Default is 'linear'.

***int*$_0$ Nint *** Number of equally spaced points to use for trapezoidal integration of flux density. Default is 10^5.

***bool* xlog *** If True, assumes wavelength domain input is logarithmic. Default is False.

***bool* ylog *** If True, assumes input SEDs are logarithmic. Default is False.

Returns

***float*$_{0,1,2,3}$ magnitude** The magnitude of the source(s) in the chosen photometric band(s), at the given luminosity distance(s).

If 3D, the shape of this array is (n_samples, n_bands, n_distances). Each dimension is removed if any of these fields are a single element or sample.

1.2 binmerger(data, bins, threshold=40)

For a 1D dataset and a set of bin boundaries containing it, this function concatenates any neighbouring bins whose occupancy is smaller than a given threshold, and returns the boundaries of the new set of bins. The result will be a subset of the input bin arrays, which satisfies the condition of minumum occupancy.

Input

***float*$_1$ data** The data which lies within the boundaries of the given bins.

***float*$_1$ bins** The boundaries of the bins to be recalculated.

***int*$_0$ threshold *** Minimum occupancy of the new bins. Default is 40.

Returns

***float*$_1$ newbins** Set of new bin boundaries.

1.3 count(array, order=True, flat=False)

For any array, this returns the elements with unique values, the frequency of each value in the array, and the indices at which they are found.

Input

***float*$_x$ array** The array to be searched for unique values.

***bool* order *** If True, results are returned in ascending order of the elements' values. If False, in descending order of their frequency. Default is True.

***bool* flat *** If True, flattens the input array and returns a 2D tuple of the indices of the flattened array, rather than N+1 dimensions for an ND array. Default is False.

Returns

***float*$_1$ values** An array of the unique elements of the input array.

***int*$_1$ frequencies** An array of the number of counts of each unique element.

***tuple*$_{x+1}$ indices** A tuple of arrays of the indices of the original array at which each unique value is located. Shape is (n_values, frequency) if the input array is 1D or has been flattened; shape is (n_values, n_dimensions, frequency) otherwise.

1.4 distance(A, B, bounds=None, shift=None)

Computes the Pythagorean distance between coordinates, optionally with specified periodic boundary conditions.

Input

***float*$_{1,2}$ A, B** Lists of coordinates. Can be identical in shape: (n_samples, n_dimensions), in which case each distance between A[i] and B[i] is returned; or either can be 1D with length n_dimensions, in which case all distances are from this one point.

***float*$_{0,1,2}$ bounds *** Boundary conditions. If None (default), distances are computed with no boundary conditions. If a single number, this is the periodicity of all axes. If a list/array with length n_dimensions, this applies different periodicities to each axis. If shape is (n_samples, n_dimensions), different boundaries are applied to each sample.

***float*$_{0,1,2}$ shift *** Translation vector. If given, A and B are shifted such that this point is translated to the origin. This argument is redundant if no boundary conditions given. It translates A and B, and has no effect on *bounds*. Same effects of the shape of *bounds* apply to *shift*.

Returns

***float*$_{0,1}$ distances** The distances between all specified points, assuming the given periodic boundaries. Single number if and only if A and B are both 1D.

1.5 find_nearest(array, value)

For a given 1D array, and a number or set of numbers, this finds the element which is closest to each number, and the corresponding index of the array.

Input

***float*$_{1}$ array** The array to be searched.

***float*$_{0,1}$ value** The number or set of numbers to find.

Returns

***int*$_{0,1}$ index** The index/indices at which the closest number(s) are found.

***float*$_{0,1}$ arrvalue** The element(s) of the array which are closest to the input value(s).

1.6 find_perc(array, value)

For a given 1D array, and a percentile or set of percentiles, this finds the element which is closest to each percentile, and the corresponding index of the array.

Input

***float*$_{1}$ array** The array to be searched.

***float*$_{0,1}$ value** The percentile or set of percentiles to find.

Returns

***int*$_{0,1}$ index** The index/indices at which the closest percentile(s) are found.

***float*$_{0,1}$ arrvalue** The element(s) of the array which are closest to the input percentile(s).

1.7 FourierAmp(x, t=None, nfreq=None, log=None)

Computes the Fourier amplitude (square root of power spectrum) of a real-valued 1D signal, optionally with a specified time domain.

Input

***float*$_1$ x** The input signal.

***float*$_1$ t *** The time domain corresponding to the input signal. If not in ascending order, t and x are automatically reordered. If not equally spaced, t and x are resampled from min(t) to max(t), while keeping the same array size. Default is None. If None, this is a set of equally spaced numbers from 0 to 1, and time and frequency units are thus arbitrary.

***int*$_0$ nfreq *** The number of frequencies to return, i.e. the length of each of the two output arrays. Default is None. If None, this equates to the length of t and x.

***float*$_0$ log *** If included, the logarithm of the Fourier amplitude is returned, and this is the base of the logarithm. If None, the output is not logarithmic. Default is None.

Returns

***float*$_1$ Freq** Frequency domain of the Fourier amplitude. Units are the inverse of the units of the time domain.

***float*$_1$ Y** Fourier amplitude of the input signal. Units are the inverse of the units of the input signal.

***float*$_0$ nyfreq** Nyquist frequency of the sampler, in the same units as Freq.

1.8 FourierAmps(x, t=None, nfreq=None, log=None)

Computes the Fourier amplitude (square root of power spectrum) of a set of real-valued 1D signals, optionally with a specified time domain.

Input

Identical to FourierAmp, except x is a 2D array of shape (n_samples, n_timesteps).

Returns

Identical to FourierAmp, except Y is a 2D array of shape (n_samples, n_frequencies).

1.9 Gaussian(x, mu=0, sig=1, norm=1)

Standard 1D Gaussian function:

$$f(x;\mu,\sigma,N) = \frac{N}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{1}$$

Input

***float*$_{0,1}$ x** Input data. Can be a single number or 1D array.

***float*$_0$ mu *** Mean of the Gaussian distribution. Default is 0.

***float*$_0$ sig *** Standard deviation of the Gaussian distribution. Default is 1.

***float*$_0$ norm *** Normalisation constant. Decides the height of the distribution and equates to its integral from negative to positive infinity. Default is 1.

Returns

***float*$_{0,1}$ f** Value of the Gaussian function.

1.10 Gaussian2D(x, y, mu_x=0, mu_y=0, sig_x=1, sig_y=1, theta=0, norm=1)

2D Gaussian function.

Input

***float*$_{0,1}$ x, y** Input data. Can be a single number or 1D array for either axis.

***float*$_0$ mu_x, mu_y *** Components of the mean vector. Default is 0 on both axes.

***float*$_0$ sig_x, sig_y *** Components of the standard deviation vector. Default is 1 on both axes.

***float*$_0$ theta *** Clockwise angle of rotation, in radians. Default is 0.

***float*$_0$ norm *** Normalisation constant. Decides the height of the distribution and equates to its integral from negative to positive infinity on both axes. Default is 1.

Returns

***float*$_{0,1,2}$ f** Value of the Gaussian function. Number of dimensions equals the sum of the number of dimensions of x and y. If 2D, shape is (len(y), len(x)).

1.11 h5load(pathtoh5, savetxt=None)

For an hdf5 archive file with specified path, this reads the hdf5 file while printing details of all files in the archive. To load the file without this output, simply use the h5py.File function.

Input

***str* pathtoh5** Filesystem path to the chosen hdf5 file.

***str/None* savetxt *** If None (default), the details of the archive are printed to the python terminal. If a string is given, the output is instead saved to a text file, in the format 'string.txt'.

Prints

***str* Header** Header for the archive description, labelling columns.

***str* Fields** Name of each valid field in the archive.

***tuple*$_1$ Shape** The shape of each archive file.

***str* Bytes** Each file's system size, in bytes. Printed with thousand-separating commas.

***str* Type** Data type of each file.

***str* Total Size** File size of the complete archive, in bytes. Printed with thousand-separating commas.

***str* Number Of Fields** Total number of fields of the archive. Printed with thousand-separating commas.

Returns

***object* h5** The hdf5 archive object as seen in h5py.

Notes

- In our data, some files of similar shape may have different filesystem sizes, due to differences in data precision.
- The function hdf5load is identical to h5load.
- A similar function exists for npz archive files, see npzload.

1.12 h5contents(h5, savetxt=None)

For an already loaded hdf5 archive file, this prints details of all files in the archive, as in h5load.

Input

***object* h5** Readily loaded hdf5 file.

***str/None* savetxt *** If None (default), the details of the archive are printed to the python terminal. If a string is given, the output is instead saved to a text file, in the format 'string.txt'.

Prints

Same details as in h5load.

Notes

- In our data, some files of similar shape may have different filesystem sizes, due to differences in data precision.
- The function hdf5contents is identical to h5contents.
- A similar function exists for npz archive files, see npzcontents.

1.13 IsNDArray(array, N)

Determines whether an array has a given number of dimensions.

Input

***float*$_x$ array** Input array.

***int*$_0$ N** Assumed number of dimensions of the array.

Returns

***bool* IsND** Returns True if array has N dimensions; returns False otherwise.

1.14 lcm(int1, int2)

Computes the lowest common multiple of the numbers int1 and int2.

Input

***int*$_0$ int1, int2** Input integers.

Returns

***int*$_0$ lcm** Lowest common multiple of the two integers.

1.15 logGaussian(x, mu=0, sig=1, norm=1)

Standard 1D log-Gaussian function:

$$f(x;\mu,\sigma,N) = \frac{N}{\sigma x\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{\ln x - \mu}{\sigma}\right)^2\right] \tag{2}$$

Input

***float*$_{0,1}$ x** Input data. Can be a single number or 1D array. Must be positive.

***float*$_0$ mu *** Mean of the log-Gaussian distribution. Default is 0.

***float*$_0$ sig *** Standard deviation of the log-Gaussian distribution. Default is 1.

***float*$_0$ norm *** Normalisation constant. Decides the height of the distribution and equates to its integral from negative to positive infinity. Default is 1.

Returns

***float*$_{0,1}$ f** Value of the log-Gaussian function.

1.16 midpoints(data, log=False)

Returns the midpoints between every two consecutive points of the input array.

Input

***float*$_1$ data** Input array.

***bool* log *** If True, the midpoints on the logarithmic axis are computed, rather than linear midpoints. Default is False.

Returns

***float*$_1$ midpoints** The midpoints of the input array. Length is one less than that of the input array.

1.17 modelB(x, t=0)

Computes the factor of dust attenuation from Nelson et al.:

$$f(\lambda,t) = \exp\left[-\left(1-\frac{7}{10}\Theta\left(t-10\mathrm{Myr}\right)\right)\times\left(\frac{\lambda}{5500\text{Å}}\right)^{-7/10}\right] \tag{3}$$

where $\Theta(x)$ is the Heaviside step function.

Input

***float*$_{0,1}$ x** Wavelength input, in Ångströms.

***float*$_{0}$ t *** Lookback time to source, in gigayears.

Returns

***float*$_{0,1}$ exp(-tau)** Dust attenuation factor.

1.18 mwa(lookback, sfh)

Computes the mass weighted age(s) of input SFH(s).

Input

***float*$_{1}$ lookback** Lookback time domain of the star formation history, in gigayears.

***float*$_{1,2}$ sfh** Star formation history, in solar masses per gigayear. If 2D, must have shape (n_samples, n_timesteps).

Returns

***float*$_{0,1}$ mwa** Mass weighted age(s) of input sample(s).

1.19 mwz(zh, sfh, time, log=False)

Computes the mass weighted metallicity of an input galaxy or set of galaxies.

Input

***float*$_{1,2}$ zh** Metallicity history, in solar units. If 2D, must have shape (n_samples, n_timesteps).

***float*$_{1,2}$ sfh** Star formation history, in solar masses per gigayear. Must have same shape as ZH.

***float*$_{1}$ time** Cosmic time domain of the star formation history, in gigayears.

***bool* log *** If True, returns the decimal logarithm of the mass weighted metallicity. Default is False.

Returns

***float*$_{0,1}$ mwz** Mass weighted metallicities, in solar units.

1.20 npzload(pathtonpz, savetxt=None)

For an npz archive file with specified path, this reads the npz file while printing details of all files in the archive. To load the file without this output, simply use the numpy.load function.

Input

***str* pathtonpz** Filesystem path to the chosen npz file.

***str/None* savetxt *** If None (default), the details of the archive are printed to the python terminal. If a string is given, the output is instead saved to a text file, in the format 'string.txt'.

Prints

***str* Header** Header for the archive description, labelling columns.

***str* Fields** Name of each valid field in the archive.

***tuple*$_1$ Shape** The shape of each archive file.

***str* Bytes** Each file's system size, in bytes. Printed with thousand-separating commas.

***str* Type** Data type of each file.

***str* Total Size** File size of the complete archive, in bytes. Printed with thousand-separating commas.

***str* Number Of Fields** Total number of fields of the archive. Printed with thousand-separating commas.

Returns

***object* npz** The npz archive object as seen in numpy.

Notes

- In our data, some files of similar shape may have different filesystem sizes, due to differences in data precision.
- A similar function exists for hdf5 archive files, see h5load.

1.21 npzcontents(npz, savetxt=None)

For an already loaded npz archive file, this prints details of all files in the archive, as in npzload.

Input

***object* npz** Readily loaded npz file.

***str/None* savetxt *** If None (default), the details of the archive are printed to the python terminal. If a string is given, the output is instead saved to a text file, in the format 'string.txt'.

Prints

Same details as in npzload.

Notes

- In our data, some files of similar shape may have different filesystem sizes, due to differences in data precision.
- A similar function exists for hdf5 archive files, see h5contents.

1.22 npz_equal(npz1, npz2)

Shows whether any two npz files are identical. Arguments can be preloaded archives or paths to files.

Input

***object/str* npz1, npz2** npz files to compare, or filesystem paths to either of the files.

Prints

***str* Uncommon Fields** List of fields of one archive not present in the other, indicating which field is in which archive.

***str* Unequal Data** List of fields whose data is not the same in both archives.

***str* Equality** Statement of equality if the two archives are identical.

Returns

***bool* Equality** Returns True if the npz files are identical, False otherwise.

1.23 n_weighted_moment(values, weights=None, n=1)

Calculates the n^{th} weighted moment (mean, variance, etc.) from a set of datapoints and optional weights.

Input

***float*$_1$ values** Set of datapoints whose weighted moment will be calculated.

***float*$_1$ weights *** Set of weights for each datapoint. Default is None. If None, all weights are unity, i.e. the result is unweighted.

***int*$_0$ n *** Order of the weighted moment to calculate. Strictly positive number. Default is 1, thus returning the mean.

Returns

***float*$_0$ mu-n** n^{th} weighted moment of the input data.

1.24 nzmean(dat, axis=1, threshold=1e-3)

Computes the mean of datapoints while ignoring all values below and including a specified threshold.

Input

***float*$_1$ dat** Data whose average is to be computed.

***int*$_0$ axis *** Axis over which to compute the average. Default is 1.

***float*$_0$ threshold *** Largest value at which datapoints are ignored when computing the mean. Default is 0.001.

Returns

***float*$_0$ mu** Mean of data along the given axis, and above the specified threshold.

1.25 polyexp(X, params, base=10)

Computes the exponential of a polynomial function of any order N:

$$f(x;\{\alpha\},b) = \exp\left[\ln b \sum_{j=0}^{N} \alpha_j x^j\right] \tag{4}$$

Input

***float*$_1$ X** Datapoints of the polynomial.

***float*$_1$ params** Coefficients of the polynomial, in descending order of their power. The length of this array is therefore one plus the order of the polynomial.

***float*$_0$ base *** The base of the exponentiation of the polynomial. Default is 10.

Returns

***float*$_1$ f** Function values of the exponentiated polynomial at the given input points.

1.26 readfile(filepath)

Code for importing a numerical text file to a 1D or 2D numpy array. Useful for very large text files.

Input

***str* filepath** Path to the text file.

Returns

***float*$_{1,2}$ data** The contents of the text file in numpy format.

1.27 rebindigits(oldbins, newbins)

For two sets of bin boundaries, this returns the indices of the second set of bins which fall into each bin in the first.

Note that a zero is appended to the first array, such that the first set of indices shows data below the lower bound. The output will have an additional element if any of the second bins are above the upper bound.

Input

***float*$_1$ oldbins** First set of bins. Must be monotonically increasing, and the smallest bin has a minimum allowed value of zero.

***float*$_1$ newbins** Second set of bins. Must be monotonically increasing, and the smallest bin has a minimum allowed value of zero.

Returns

***tuple*$_2$ Indices** Tuple containing an array for each of the first bins, which lists all indices of the second array which belong to each bin. Tuple element is empty if second bins not in the given first bin.

1.28 rebindigits_linear(oldbins, number)

Special case of rebindigits, in which the second set of bins are linearly spaced between the bounds of the first set. The number of bins in the second set is specified and their boundaies and size are returned along with the bin indices.

Input

***float*$_1$ oldbins** First set of bins. Must be monotonically increasing, and the smallest bin has a minimum allowed value of zero.

***int*$_1$ number** Number of linearly spaced bins for the second set.

Returns

***float*$_1$ newbins** Second set of bin boundaries. Length is one more than the specified number of bins, *number*.

***float*$_0$ interval** Spacing between second set of bins.

***tuple*$_2$ Indices** Tuple containing an array for each of the first bins, which lists all indices of the second array which belong to each bin. Tuple element is empty if second bins not in the given first bin.

1.29 rmNanInf(arr)

Replaces all NaN or infinite array elements with zeros.

Input

***float*$_x$ arr** Input array.

Returns

***float*$_x$ array** Input array, identical except for NaN and infinity replaced with zeros.

1.30 running_stdev(dat, window)

Computes a standard deviation filter of a 1D dataset. Useful for computing scatter.

Input

***float*$_1$ dat** Input array.

***int*$_0$ window** Size of filter window, in number of datapoints. This window is centered on the target element, getting smaller as it approaches the edges of the input array.

Returns

***float*$_0$ stdev** Standard deviation filtered data. Same shape as input array.

1.31 s2ms(t, String=True, Round=True)

Converts a time given in seconds to minutes and seconds, or to hours, minutes and seconds if longer than or equal to 1 hour. Returns either a numpy array or a string.

Input

***float*$_0$ t** The input time, in seconds.

***bool* String *** If True, the output is given as a single string. Default is True.

***bool* Round *** If True, the result is rounded to the nearest second, in which case the output will be in integer format. Default is True.

Returns

***float/int/str*$_1$ data** The time in h:m:s format, in descending order of unit value.

1.32 sfhint(sfh, time, log=False

Computes an integral of star formation histories over cosmic time to obtain a final (unrecycled) stellar mass value.

Input

***float*$_{1,2}$ sfh** Star formation history, in solar masses per gigayear. If 2D, must have shape (n_samples, n_timesteps).

***float*$_1$ time** Cosmic time domain of the star formation history, in gigayears.

***bool* log *** If True, returns the decimal logarithm of the stellar mass. Default is False.

Returns

***float*$_{0,1}$ ms** Final stellar mass(es), in solar units.

1.33 sort_arrays(arrays)

For a tuple of at least two 1D arrays of equal length, this sorts all arrays with respect to the first array of the tuple.

Input

***float/tuple*$_2$ arrays** Arrays to be sorted. Ensure that the reference array is the first element of this input.

Returns

***float*$_2$ Sorted Arrays** The first array will now be in ascending order; the rest will be ordered according to the first, conserving the correspondence between their elements.

1.34 verb(index, length, numperc=100, string=None)

Verbose output for use in loops. Prints percentage of completion of said loop.

Input

***int*$_0$ index** Index of the loop, i.e. number of calls to the loop thus far, starting from zero.

***int*$_0$ length** Total number of calls to the loop.

***int*$_0$ numperc *** Number of percentage progress updates to print. Default is 100, i.e. every 1% is printed. If 20, every 5% will be printed.

***str* string *** Prefix to each printed update. If None, only the percentage completed is printed. Otherwise, this string is printed as well. Useful for identifying the ongoing task. Default is None.

Prints

***str* Verbose Output** String indicating the percentage of the task which has been completed.

1.35 virv(mass, halfrad)

Proxy for the virial velocity of a dark matter subhalo, in terms of its total mass and half-mass radius.

$$\tilde{v}_{\rm vir}(t) = \sqrt{\frac{m_h(t)}{R_{1/2}(t)}} \tag{5}$$

Units are unimportant for general use of this function, however this project typically uses solar masses and megapersecs.

Input

***float*$_{\rm x}$ mass** Total subhalo mass.

***float*$_{\rm x}$ halfrad** Dark matter half-mass radius.

Returns

***float*$_{\rm x}$ velocity** Virial velocity proxy, in arbitrary units.

1.36 External Functions & Classes

- **chisquare** chisquare from scipy.stats
- **Counter** Counter from collections
- **FlatLambdaCDM** FlatLambdaCDM from astropy.cosmology
- **gf** gaussian_filter from scipy.ndimage
- **interp1d** interp1d from scipy.interpolate
- **interp2d** interp2d from scipy.interpolate
- **iqr** iqr from scipy.stats
- **mf** median_filter from scipy.ndimage
- **monotonic** monotonic from time
- **MultipleLocator** MultipleLocator from matplotlib.ticker
- **spearmanr** spearmanr from scipy.stats
- **u** units from astropy

2 Network Module

This module contains the codes used to build and train the neural networks, and the quantile transformation tools used in preprocessing. This module requires installation of SciKit-Learn and TensorFlow for Python.

2.1 CentralNetwork() & SatelliteNetwork()

The function CentralNetwork compiles the network designed for use with central galaxies and returns the keras Model object. The function SatelliteNetwork does the same for the satellite galaxy model.

Returns

***object* model** Model object containing the complete network.

2.2 GQTscalnorm(data, add_dim=False)

Gaussian Quantile Transformation of 2D (non-temporal) or 3D (temporal) dataset, with scalar normalisation.

***float*$_{2,3}$ data** Data to be transformed. If this is 3D data, axes must be in order (samples, timesteps, variables). If 2D, timesteps do not apply.

***bool* add_dim *** If True, an additional dimension will be added to a 2D transformed output, such that it has shape (n_samples, n_variables, 1). This is useful when concatenating 3D arrays. Default is False.

Returns

***float*$_{2,3}$ qData** Quantile Transformed output.

***object* GQT** SKLearn transformation object, for quick access to the transformation.

The function **GQTscalinv** takes the outputs of this function in the given order and the *add_dim* boolean to perform an inverse transformation. This returns only the un-transformed data in the same shape as qData, unless *add_dim* is True.

2.3 GQTvecnorm(data)

Gaussian Quantile Transformation of 3D (temporal) dataset, with vector normalisation.

***float*$_3$ data** Data to be transformed. Axes must be in order (samples, timesteps, variables).

Returns

***float*$_3$ qData** Quantile Transformed output.

***object* GQT** SKLearn transformation object, for quick access to the transformation.

The function **GQTvecinv** takes the outputs of this function in the given order to perform an inverse transformation. This returns only the un-transformed data in the same shape as qData.

2.4 lr_exp(E, lr0=0.001, k=0.1)

Variable exponential learning rate, for recursive use in network training. For epoch E, the learning rate $\Gamma(E)$ is defined:

$$\Gamma(E) = \Gamma_0 \exp\left[-\gamma E\right] \tag{6}$$

where Γ_0 is the initial learning rate, and γ is the exponential decay rate.

***int*$_0$ E** Epoch number.

***float*$_0$ lr0 *** Initial learning rate. Default is 0.001.

***float*$_0$ k *** Exponential decay rate. Default is 0.1.

Returns

***float*$_0$ lr** Learning rate at specified epoch.

2.5 lr_expcap(E, lr0=0.001, k=0.1, cap=6e-5)

Capped exponential learning rate. Similar to lr_exp, but with a minimum allowed value of the learning rate. If this minimum value is κ, the learning rate $\Gamma(E)$ is akin to eq. (6):

$$\Gamma(E) = \max\left[\Gamma_0 \exp\left[-\gamma E\right], \kappa\right] \tag{7}$$

***int*$_0$ E** Epoch number.

***float*$_0$ lr0 *** Initial learning rate. Default is 0.001.

***float*$_0$ k *** Exponential decay rate. Default is 0.1.

***float*$_0$ cap *** Minimumallowed learning rate. Default is 6e-5.

Returns

***float*$_0$ lr** Learning rate at specified epoch.

2.6 lr_step(E, lr0=0.001, dr=0.5, Ed=10)

Step-based variable learning rate. Learning rate is multiplied by a constant r at equally spaced intervals, E_d epochs apart.

$$\Gamma(E) = \Gamma_0 \times r^{\left\lfloor \frac{E}{E_d} \right\rfloor} \tag{8}$$

***int*$_0$ E** Epoch number.

***float*$_0$ lr0 *** Initial learning rate. Default is 0.001.

***float*$_0$ dr *** Multiplicative step factor. Default is 0.5.

***int*$_0$ Ed *** Number of epochs between steps. Default is 10.

Returns

***float*$_0$ lr** Learning rate at specified epoch.

2.7 lr_step_r(E, lr0=0.001, dr=0.5, rp=0.25, Es=10)

Random step function. For each epoch after a specified number, there is a probability that the learning rate is randomly multiplied by the step factor.

***int*$_0$ E** Epoch number.

***float*$_0$ lr0 *** Initial learning rate. Default is 0.001.

***float*$_0$ dr *** Multiplicative step factor. Default is 0.5.

***float*$_0$ rp *** The probability of the learning rate being changed. Default is 0.25.

***int*$_0$ Es *** Number of epochs after which the learning rate can be changed. Default is 10.

Returns

***float*$_0$ lr** Learning rate at specified epoch.

2.8 External Functions & Classes

- **keras** keras from tensorflow
- **RFR** RandomForestRegressor from sklearn.ensemble
- **QuantileTransformer** QuantileTransformer from sklearn.preprocessing

3 SPS Module

This module contains the code for the emulation of spectra and other observables from SFH and ZH predictions. This requires installation of the PythonFSPS package.

3.1 lineflux(sfh, zh, lookback, line='Halpha')

Computes the luminosity of an emission lime for a given star formation and metallicity history.

Input

***float*$_1$ sfh** 1D star formation history array, in solar masses per gigayear.

***float*$_1$ zh** 1D metallicity history array, in solar metallicity units.

***float*$_1$ lookback** 1D lookback time (age) array, in gigayears.

***str* line *** Name of emission line whose luminosity will be computed. Valid strings are from a list of keys specified by linekeys. Default is 'Halpha'.

Returns

***float*$_0$ Flux** Luminosity of the emission line, in solar luminosities.

3.2 linefluxes(sfh, zh, lookback, line='Halpha', verbose=True, numperc=100, string='Line Fluxes')

Computes emission line luminosity as in lineflux, for 2D arrays of multiple star formation and metallicity histories.

As this can take time with large datasets, this includes support for the verb function to print progress updates.

Input

***float*$_2$ sfh** 2D array of star formation histories, in solar masses per gigayear. Axes must be in order (samples, timesteps).

***float*$_2$ zh** 2D array of metallicity histories, in solar metallicity units. Axes must be in order (samples, timesteps).

***float*$_1$ lookback** 1D lookback time (age) array, in gigayears.

***str* line *** Name of emission line whose luminosity will be computed. Valid strings are from a list of keys specified by linekeys. Default is 'Halpha'.

***bool* verbose *** If True (default), verbose statements of the percentage of the task's completion are printed in terminal.

Two additional arguments *numperc* and *string* can be passed to this function, which have the same role as in verb (see verb for details). Defaults here are 100 and 'Line Fluxes', respectively.

Returns

***float*$_1$ Fluxes** 1D array of line emission luminosity for each star formation and metallicity history, in solar luminosities.

3.3 linewavelength(line='Halpha')

Returns the wavelength of an emission line specified by a key.

Input

***str* line *** Name of emission line. Valid strings are from a list of keys specified by linekeys. Default is 'Halpha'.

Returns

***float*$_0$ wavelength** Wavelength of emission line, in Ångströms.

3.4 linekeys(savetxt=None)

Prints the full, enumerated list of emission line keys for use in related functions, and their wavelengths, in ascending order of wavelength.

Input

***str/None* savetxt *** If None (default), the data are printed to the python terminal. If a string is given, the output is instead saved to a text file, in the format 'string.txt'.

Prints

***int*$_0$ index** Index of the emission line.

***str* key** Emission line key.

***float*$_0$ wavelength** Wavelength of the emission line, in Ångströms.

3.5 spectrum(sfh, zh, lookback, pera=True, lambdamin=2980, lambdamax=11230)

Computes the spectral energy distribution from a given star formation and metallicity history.

Input

***float*$_1$ sfh** 1D star formation history array, in solar masses per gigayear.

***float*$_1$ zh** 1D metallicity history array, in solar metallicity units.

***float*$_1$ lookback** 1D lookback time (age) array, in gigayears.

***bool* pera *** If True (default) the spectrum is computed in units of solar luminosities per Ångström. If False, the spectrum is computed in units of solar luminosities per Hertz.

***float*$_0$ lambdamin, lambdamax *** Upper and lower limits of the wavelength domain, in Ångströms. Default wavelength range is (2980, 11230), which encloses all SDSS bandpass filters. See waverange for further information.

Returns

***float*$_1$ wave** Wavelength domain of the spectrum, in Ångströms.

***float*$_1$ Spectrum** Spectral energy distribution, in units specified by the *pera* boolean argument.

3.6 spectra(sfh, zh, lookback, pera=True, lambdamin=2980, lambdamax=11230, verbose=True, numperc=100, string='Spectra')

Computes spectral energy distributions as in spectrum, for 2D arrays of multiple star formation and metallicity histories.

As this can take time with large datasets, this includes support for the verb function to print progress updates.

Input

***float*$_2$ sfh** 2D array of star formation histories, in solar masses per gigayear. Axes must be in order (samples, timesteps).

***float*$_2$ zh** 2D array of metallicity histories, in solar metallicity units. Axes must be in order (samples, timesteps).

***float*$_1$ lookback** 1D lookback time (age) array, in gigayears.

***bool* pera *** If True (default) each spectrum is computed in units of solar luminosities per Ångström. If False, each spectrum is computed in units of solar luminosities per Hertz.

***float*$_0$ lambdamin, lambdamax *** Upper and lower limits of the wavelength domain, in Ångströms. Default wavelength range is (2980, 11230), which encloses all SDSS bandpass filters. See waverange for further information.

***bool* verbose *** If True (default), verbose statements of the percentage of the task's completion are printed in terminal.

Two additional arguments *numperc* and *string* can be passed to this function, which have the same role as in verb (see verb for details). Defaults here are 100 and 'Spectra', respectively.

Returns

***float*$_1$ wave** Wavelength domain of all spectra, in Ångströms.

***float*$_2$ Spectra** 2D array of all spectral energy distributions determined by the star formation and metallicity histories. Units specified by the *pera* boolean argument.

3.7 waverange(lambdamin=2980, lambdamax=11230)

With specified lower and upper wavelengths, this returns the location indices of the FSPS wavelength array which contains them.

***float*$_0$ lambdamin, lambdamax *** Upper and lower limits of the wavelength domain, in Ångströms. Automatically switched if $\lambda_{\min} > \lambda_{\max}$. Default wavelength range is (2980, 11230), which encloses all SDSS bandpass filters.

Note that this function refers to the standard wavelength array of FSPS, and returns the subset of the array which encloses these limits, therefore the true limits are the nearest points below and above these arguments.

Returns

***int*$_1$ location** Ordered indices indicating the subset of the FSPS wavelength array containing these wavelength limits.

4 GriSPy Module

The GriSPy Module utilises the Grid Search In Python package for nearest neighbour searches within periodic boundary conditions, in this case the TNG simulation volumes.

The data required to run these functions comes directly from the TNG simulation, and can be obtained by running the subhistory function in the TNG Module.

4.1 Overdensity(loc, radius=1, sim=100, snap=99)

Computes the DM overdensity within spherical regions in TNG, specified by a list of coordinates and radii.

Input

***float*$_2$ loc** Coordinates of the centre of each spherical region, in comoving megaparsecs. Axes must be in order (samples, coordinate axes). Coordinates are Cartesian, in order x, y, z.

***float*$_{0,1}$ radius *** Radius or radii of the overdensity region(s). Default is 1Mpc.

***float/int/str*$_0$ sim *** Number specifying the TNG simulation. Can be either 100, referring to TNG100-1, or 300, referring to TNG300-1.

***int*$_0$ snap *** Snapshot number of the TNG simulation, where larger numbers are later snapshots. Must be an integer from 0 to 99, inclusive.

Returns

***float*$_{1,2}$ delta** Set of overdensities. If multiple radii given, this has shape (n_samples, n_radii), otherwise this is a 1D array.

4.2 Overdensities(locs, radius=1, sim=100, verbose=True, numperc=100, string='Overdensity History')

Computes dark matter overdensities as in Overdensity, now evaluated at all snapshots in TNG, thereby computing a full overdensity history.

Input

***float*$_3$ locs** Coordinates of the centre of each spherical region, in comoving megaparsecs. Axes must be in order (samples, snapshots, coordinate axes). Coordinates are Cartesian, in order x, y, z.

***float*$_{0,1}$ radius *** Radius or radii of the overdensity region(s). Default is 1Mpc.

***float/int/str*$_0$ sim *** Number specifying the TNG simulation. Can be either 100, referring to TNG100-1, or 300, referring to TNG300-1.

***bool* verbose *** If True (default), verbose statements of the percentage of the task's completion are printed in terminal.

Two additional arguments *numperc* and *string* can be passed to this function, which have the same role as in verb (see verb for details). Defaults here are 100 and 'Overdensity History', respectively.

Returns

***float*$_{2,3}$ delta** Set of overdensity histories. If multiple radii given, this has shape (n_snapshots, n_samples, n_radii), otherwise this is a 2D array.

4.3 Skew(loc, radius=3, sim=100, snap=99)

Computes the dark matter weighted radial skew within spherical regions in TNG, specified by a list of coordinates and radii.

Input

Same quantities as in Overdensity, except that the default radius is 3Mpc.

Returns

***float*$_{1,2}$ skew** Set of skews. If multiple radii given, this has shape (n_samples, n_radii), otherwise this is a 1D array.

***float*$_{1,2}$ min_d** Distance to nearest subhalo from target coordinates, in megapersecs. Same shape as *skew*.

4.4 Skews(locs, radius=3, sim=100, verbose=True, numperc=100, string='Skewness History')

Computes radial skews as in Skew, now evaluated at all snapshots in TNG, thereby computing a full skew history.

Input

Same quantities as in Overdensities, except that the default radius is 3Mpc, and the default verbosity string is 'Skewness History'.

Returns

***float*$_{2,3}$ skews** Set of skew histories. If multiple radii given, this has shape (n_snapshots, n_samples, n_radii), otherwise this is a 2D array.

***float*$_{2,3}$ min_d** Set of smallest distances, with same shape as *skews*.

5 TNG Module

This module is used to compute properties from the TNG simulation data. It is intented to be used in the IllustrisTNG Jupyter Workspace, as this contains direct access to the TNG data.

All functions in this module contain the following optional arguments:

***int/str*$_0$ simulation, order *** Numbers specifying the TNG simulation run. Simulation refers to TNG50, TNG100 or TNG300, and can be any one of these 3 numbers. Order refers to TNG100-1, TNG100-2, etc. and can be any integer from 1 to 4, inclusive. Default simulation run is TNG100-1.

bool*$_0$ Dark Boolean which, if True, queries the pure dark simulation equivalent to the chosen simulation. Default is False.

If no Input list is given in the documentation below, these are the only arguments of the function.

5.1 hostIDs(subIDs, snapnum=99, simulation=100, order=1)

For SubfindID(s), returns the FoF ID of its host(s), and the SubfindID(s) of the central subhalo(s). States whether the given SubfindID is the central subhalo or not.

Input

***int*$_{0,1}$ subIDs** SubfindID(s) of the target subhalo(s). Dependent on simulation run and snapshot number.

***int*$_0$ snapnum *** Snapshot number from which to draw the subhalo. Default is 99, i.e. $z = 0$. Can be any integer from 0 to 99, inclusive.

Returns

***int*$_{0,1}$ haloIDs** FoF ID(s) of the host halo(s) of the given subhalo(s).

***int*$_{0,1}$ centIDs** SubfindID(s) of the central subhalo(s) of the host halo(s).

***bool*$_{0,1}$ iscent** Returns True where the SubfindID is that of a central subhalo. Returns False otherwise.

5.2 IsCent(subIDs, snapnum=99, simulation=100, order=1)

States whether given SubfindID(s) are those of a central subhalo or not.

Input

***int*$_{0,1}$ subIDs** SubfindID(s) of the target subhalo(s). Dependent on simulation run and snapshot number.

***int*$_0$ snapnum *** Snapshot number from which to draw the subhalo. Default is 99, i.e. $z = 0$. Can be any integer from 0 to 99, inclusive.

Returns

***bool*$_{0,1}$ IsCent** Returns True where the SubfindID is that of a central subhalo. Returns False otherwise.

5.3 Infall(subIDs, snapnum=99, simulation=100, order=1, save=None)

Computes an array of infall parameters for a set of subhalo IDs from a given simulation run.

Input

***int*$_1$ subIDs** SubfindIDs of the target subhalos. Dependent on simulation run and snapshot number.

***int*$_0$ snapnum *** Snapshot number from which to draw the subhalo. Default is 99, i.e. $z = 0$. Can be any integer from 0 to 99, inclusive.

***str/None* save *** If a string, the output is saved to a 2D npy file, whose filename is *string.npy*. If None (default), output is not saved. If querying one of the Dark simulations, the suffix *-Dark* is added to the filename, i.e. the filename is *string-Dark.npy*.

Returns

***float*$_2$ Infall** 2D array of infall parameters of the target subhalos. Each row of this array contains a single quantity. The array has shape (14, n_subhalos), and are given in the following order:

0. Simulation number.
1. Scaled infall time.
2. Scaled formation time.
3. Logarithm of infall DM mass ratio.
4. Logarithm of infall velocity, in km/s.
5. Host halo DM mass, in simulation units.
6. Subhalo DM mass, in simulation units.
7. Host halo stellar mass, in simulation units.
8. Subhalo stellar mass, in simulation units.
9. Host halo gas mass, in simulation units.
10. Subhalo gas mass, in simulation units.
11. Subfind IDs of input subhalos.
12. FoF IDs of host halos.
13. Subfind IDs of central subhalos of hosts.

Note that this array consists of floats, and some quantities may need to be converted back to integers.

If querying one of the Dark simulations, stellar and gas masses (7-10 inclusive) do not appear in the output as they do not exist in these simulations.

5.4 sfhzh(index, simulation=100, order=1, snapnum=99, time=TNGtime, NT=10000, radius=None)

Returns the SFH and ZH of a subhalo from the TNG database, with custom binning and optional stellar particle aperture.

Input

***int*$_0$ index** SubfindID of the target subhalo. Dependent on simulation run and snapshot number.

***int*$_0$ snapnum *** Snapshot number from which to draw the subhalo. Default is 99, i.e. $z = 0$. Can be any integer from 0 to 99, inclusive.

***float*$_1$ time *** Time domain of the desired SFH and ZH outputs, in gigayears. Default is the TNG snapshot times.

***int*$_0$ NT *** Number of equally spaced time bins to compute the fine age spectrum. This exists to avoid distorting effects when computing a star formation rate in unequal bins. Default is 10^4. The spacing between fine bins equates to the simulation Hubble time divided by the number of bins, therefore the default bin spacing is 1.3803 Myr.

***float*$_0$ radius *** If given, this limits the data to particles within the specified comoving radius of the galaxy's centre of mass. If None (default), all stellar particles which are gravitationally bound to this subhalo contribute to the calculation. Units are in megaparsecs.

Returns

***float*$_1$ SFH** Star formation history of the target galaxy, in solar masses per gigayear, corresponding to the input time array.

***float*$_1$ ZH** Stellar metallicity history of the target galaxy, in solar metallicity units.

5.5 subhistory(simulation=100, order=1)

Computes the mass, radius and location histories of all subhalos in the chosen simulation run. These are saved to 100 npz files; one for each snapshot number.

This algorithm can take time to complete, and will save a few dozen gigabytes per simulation run. Files are saved in the directory envsnaps (envsnaps-dark if Dark), which is created if it does not exist.

This data is necessary for running functions in the GriSPy Module.

Prints

***str* Progress** Prints a statement that the algorithm is x% complete.

Saves

***npz* envsnaps** Data of the environmental history of the full simulation. Data saved in 100 separate files for each snapshot, with the snapshot number and simulation run given in the file name. Fields in every npz archive include the following:

- **Parent** FoF ID of the host halo.
- **Location** Coordinates of the subhalos' centres of mass, in megaparsecs.
- **MSub** Dark matter mass component of the subhalos.
- **RSub** Dark matter half-mass radius of the subhalos.