ExoVista v2.3 User Guide

Alex Howe & Chris Stark, NASA GSFC

Table of Contents

1. Summary

ExoVista 2 is a hybrid Python/C++ software package based on an earlier IDL/C iteration that generates synthetic exoplanetary systems. ExoVista models exoplanet atmospheres in reflected light, stellar spectra using Kurucz stellar atmosphere models, and debris disks in scattered light using realistic spatial distributions and optical properties. Planets can be drawn from measured/extrapolated *Kepler* occurrence rates (Dulz et al. 2020) and are checked for basic stability criteria; debris disks are dynamically quasi-self-consistent with the underlying planetary system. All bodies are integrated with a Bulirsch-Stoer integrator to determine their barycentric velocities, positions, and orbits. The output product is a multi-extension FITS file that contains all of the information necessary to generate a spectral data cube of the system for direct imaging simulations of coronagraphs/starshades, as well as position/velocity data for simulation of RV and astrometric data sets, and transit and eclipse times. A more detailed description of the scientific model used by ExoVista is given in Appendix A.

2. Philosophy & Numerical Approach

ExoVista was designed with two primary numerical goals. First, ExoVista was designed for speed, to rapidly generate randomized planetary systems for a large number of stars/scenarios. To enable this, only simple checks are performed for dynamic stability of the planetary systems, and analytic models are used for the debris disks. Second, ExoVista's output file size was minimized, such that all required information on the planetary system is contained in $\lesssim$25 MB. (Determined by the spectral resolution of the disk and the baseline of orbit integration.) To minimize file size, ExoVista does not save a simple spectral image cube. Instead, the output is a list of each point source's position, velocity, orbit, and contrast/flux, combined with a contrast data cube of the debris disk (which has a smooth wavelength dependence and can be saved at lower spectral resolution). Transit and eclipse times are saved separately. As a consequence of recording disk contrast instead of disk flux, ExoVista cannot include thermal emission from the disk. We have provided routines in Python (`load_scene.py`) that load the output FITS files to aid in converting it to a spectral image cube.

3. Installation and Use Cases

ExoVista is designed to generate a universe of planetary systems based on varying assumptions about the overall exoplanet population. This use case requires familiarity with the complete ExoVista code and moderate to large computing facilities. Most of this guide is designed to provide a detailed description of ExoVista, its modules, and its dependencies for this use case. Additionally, a full API for ExoVista is provided in Appendix B.

Many users will likely wish to interact only with the data products of ExoVista, using existing simulations of planetary systems to simulate direct imaging instruments, or for planet detection

simulations. These users should download the existing FITS planetary scene files from https://tools.emac.gsfc.nasa.gov/exovista/ which can be read by any FITS tool such as astropy. *Note: these files do not include transit detection.*
An example of an up-to-date output file (for our Solar system) is also provided in the "`output`" subdirectory.
A detailed description of the FITS files produced by ExoVista is provided in Section 7. The included `load_scene.py` and `readfits.py` routines also provide pre-programmed options for reading the FITS files. Meanwhile, the `parsefits.py` routine takes an output file for a single system and generates a new, matching input file. This can be useful for recomputing randomly-generated planetary systems with changes.

To run the main modules of ExoVista, you must have a Python interpreter (**Python 3.8 or higher** recommended). You will also need to have installed, in addition to the "standard" suite of Python modules, the Python packages **scipy, astropy**, and **cython**. The **multiprocessing** package is needed if you wish to use ExoVista with parallel processing, although ExoVista can run as a serial code without it.

ExoVista also requires a **C++ compiler**:
For Linux users, g++ is usually available.
For Mac OS users, it is recommended to install Apple's XCode to obtain a compiler.
For Windows users, it is recommended to use the Microsoft Visual C/C++ compiler. However, other compilers such as MinGW or Cygwin are likely to work.

To install ExoVista, download the current version of the ExoVista package from the Github repository into the desired directory on your local machine.

Open a terminal window, navigate to the "src" subdirectory in the directory containing the ExoVista code, and compile the disk imaging and N-body integration modules by typing the following command:

```
python setup.py build_ext --inplace
```

This should call your C++ compiler successfully and generate the files "`wrapImage.cpp`" and "`wrapIntegrator.cpp`" on your machine in the `src` subdirectory. If this fails, you may need to edit the lines:

```
os.environ['CC'] = 'gcc'
os.environ['CXX'] = 'g++'
```

in the "`setup.py`" file to reflect your local C and C++ compilers. Once the `wrapImage` and `wrapIntegrator` modules are built successfully, you will be ready to run all ExoVista modules.

This installation process has been tested on both Mac OS and Windows machines.

4. Running

ExoVista can be run from the Python command line, but we recommend using a wrapper script to make full use of its features. This is especially important for larger target lists that require parallel processing. You may write your own wrapper based on the API described in Appendix B or modify the provided "`ExoVista.py`" and "`ExoVistaSystem.py`" scripts.

ExoVista has two main modes of operation for its primary use case of generating simulated exoplanetary systems:

1. Generate a single, user-defined planetary system to simulate observations of known planets. An example is provided in "`ExoVistaSystem.py`."
2. Create a universe of stochastically-generated simulated planetary systems based on a list of target stars and their properties. An example is provided in "`ExoVista.py`."

4.1 Generating a Single, User-Defined System

"`ExoVistaSystem.py`" provides an example of how to create a user-defined planetary system. Most of the machinery in this script is designed to prompt for and check the format of an input file and should not be touched.

`ExoVistaSystem.py` can be run with an input file name as a command line argument, but it will prompt for one if it is not given, or if the requested file fails. The input file should define all aspects of the planetary system, and the format for this file is described in Section 5.1. The provided "`solar_system.dat`" input file defines our own Solar system (excluding Mercury, which is held to be below the detection limit).

The core functionality of the script is as follows: first, create a `Settings` object with the desired model parameters. Syntax:

```
    settings = Settings.Settings(<optional arguments>)
```

Next, call the `read_solarsystem()` function with the `Settings` object and an input file name as an argument. Syntax:

```
    read_solarsystem.read_solarsystem(settings,system_file=filename)
```

The `read_solarsystem()` function converts the input file into the data structures used by ExoVista. The function outputs five new data structures (here described as `s,p,a,d,c`) plus an amended `Settings` object, which are used as inputs for `generate_scene()`.

Next, call the `generate_scene()` function to produce observations of the system over time. Syntax:

```
generate_scene.generate_scene(s,p,a,d,c,new_settings)
```

The parameters set by the Settings object are listed in Appendix B.9. Notable parameters for ExoVista users are:

- **`pixscale`**: the pixel size of the image in arcseconds. Default is 0.002.
- **`iwa`**: inner working angle of the coronagraph in arcseconds. Default is 0.015.
- **`specres`**: the resolution of the star and planet spectra. Default is R=300.
- **`specresdisk`**: the spectral resolution of the disk contrast cube. Default is R=10.
- **`lambdamin`** and **`lambdamax`**: the minimum and maximum wavelengths of the spectra, respectively, in microns. Defaults are 0.3 and 1.0.
- **`imin`** and **`imax`**: the minimum and maximum orbital inclination of the planets, in degrees. Default is 0.0. *(Note: a near-zero inclination may cause numerical instability in the longitude of ascending node and argument of periastron, but the more important longitude of periastron will remain stable.)*
- **`timemax`**: the length of time to integrate the orbits of the planets, in years. Default is 1.e-10. Typical values for a survey are 5-10.
- **`dt`**: the time step at which to report the planetary positions and spectra, in years. Default is 10 days.
- **`output_dir`**: the (relative) directory where the FITS files will be output. Default is "`output`."
- **`ncomponents`**: the number of components in the disk. *(Note: this must remain at the pre-set value of `ndisk` for the debris disk to be handled correctly.)*
- **`randphase`**: if True, the Lambertian phase function for non-phase-resolved planetary albedos will be scaled by a randomized parameter based on the spread of phase functions in Solar system objects in Sudarsky et al. (2005). Default is `False`.
- **`randrad`**: if True, a random spread will be added to the mass-radius relation used to generate planets, based on the "fractional dispersion" hyperparameters in the model of Chen & Kipping (2017). *Note: not compatible with the* "`usebins`" *parameter described in Section 4.2.* Default is `False`.
- **`hires`**: if True, the disk spectral cube will be computed at the same resolution as the star and planetary spectra. This will take much more memory, but may be useful for generating emission and absorption disk spectra in future versions. It is recommended to use this only with single-system input files. Default is `False`.

The `generate_scene()` function is the main output function for ExoVista. It generates a FITS file for each planetary system in the data structure, which includes a spectral data cube for the exozodiacal disk, time-dependent spectra and orbital parameters for the star and each planet, and a list of transit and eclipse times. The output FITS format is described in Section 7. This FITS file may be used to model simulated observations as desired.

4.2 Generating a Universe of Randomized Planetary System from a Target List

"`ExoVista.py`" provides an example of how to generate a universe of planetary systems from a target list of stars. Most of the machinery in this script is used to set up Python's multiprocessing functions and should not be touched.

The input file for this script is the stellar target list, which must define all required parameters for the stars used by ExoVista. The format for this file is described in Section 5.2. The provided "`master_target_list-usingDR2-50_pc.txt`" input file includes a master list of roughly 8,000 target stars within 50 pc of Earth from the LUVOIR/HabEx design studies. We have also provided a short target list of 10 stars called "`target_list10.txt`," which makes testing code changes much easier.

The notable features of this script for ExoVista users are:

- **`parallel`**: if `True`, Python will distribute the stars evenly to all available cores on the system to speed the calculations in `generate_scene()`. Defaults to `False` if the "multiprocessing" module is not available.
- **`maxcores`**: allows the user to set a maximum number of cores used instead of using all available cores.
- **`Settings`**: as with the single-system script, this is an object that contains the settable code parameters. It is used as an argument by the `generate_planets()`, `generate_disks()`, and `generate_scene()` functions.
- **`target_list_file`**: the target list used by ExoVista to generate planetary systems. This is the main input file for ExoVista, the format of which is described in Section 5.2.
- **`load_stars()`**, **`generate_planets()`**, and **`generate_disks()`**: these three functions serve the same purpose as the `read_solarsystem()` function for a single system, generating the data structures used by the code to generate the scenes. These functions include optional arguments to change the configuration of the planetary systems, which are described in depth in the ExoVista API in Appendix B.
- **`addearth`:** if `True`, `generate_planets()` will add an extra Earth twin (except with zero eccentricity) to every planetary system (orbit scaled with stellar luminosity) with negligible mass (so as not to disrupt stability or orbit integration). This can be useful if, for example, you wish to calculate exposure times based on the detectability of an Earth twin. Default is `False`.
- **`usebins`:** if `True`, `generate_planets()` will use the `rbound` and `abound` arrays found in `defaults.py` to define bins based on planet types, and will ensure that exactly the expected number of planets of each type occur in each bin (subject to rounding). Default is `False`.

4.3 Post-Processing Routines

We have included several post-processing scripts in the ExoVista package. All of these scripts are backwards-compatible with earlier versions of the code, at least in that they will run successfully, although functionality may be limited. *Note: all of the scripts will ignore the "extra" Earth twin in the FITS file if it is present.*

"`parsefits.py`" is a script that reads an output FITS file and generates a new, matching single-system input file, which can be used to replicate the (possibly randomized) system described by the output file exactly. This will be useful for recomputing a scene for a given system with different parameters, especially for the (planned) disk absorption/emission spectrum upgrade.

`parsefits.py` may be run directly from the command line. It may be run with an argument of a valid FITS filename, in which case it will proceed directly to parsing. If it is run without a command line argument, it will produce a numbered list of the FITS files in the default output directory and prompt you for a file.

"`load_scene.py`" is a module containing the function `load_scene()`, which can process an ExoVista output FITS file into useful NumPy arrays. It is called with an input file and optionally a time (in years) as inputs with the syntax:

```
    load_scene.load_scene(inputfile, time=0)
```

This function returns a tuple containing 8 elements: (1) the stellar wavelength points, (2) the stellar coordinates and (3) spectrum at the specified time, (4) the planet coordinates and (5) spectra at the specified time, (6) the disk image cube interpolated to the stellar wavelength points, (7) the stellar angular diameter, and (8) the pixel scale. These data structures can then be used for image simulations and other applications. They are described in detail in Appendix B.11.

"`add_background.py`" is a module that generates a random data cube of extragalactic background sources for a given scene based on the Haystacks model found at
https://asd.gsfc.nasa.gov/projects/haystacks/downloads.html
This module requires you to download Extragalactic background cubes 0 through 5 (may change if you change the wavelength coverage of the scenes) from the Haystacks webpage. It is called with an input file and optionally a time (in years) as inputs with the syntax:

```
    add_background.add_background(inputfile, time=0)
```

This function returns a data cube of extragalactic fluxes with the same format as the disk data cube. *Note that this module is still in active development, with plans to add background stars, parallax, and target proper motion.*

"`readfits.py`" is a plotting routine that produces five plots of interest for a specified FITS file and prints the transit and eclipse times listed in the file. Similar to `parsefits.py`, `readfits.py` may be run directly from the command line. It may be run with an argument of a valid FITS filename, in which case it will proceed directly to plotting. If it is run without a command line argument, it will produce a numbered list of FITS files in the default output directory and prompt you for a file.

The plots produced by `readfits.py` are:

- An ideal simulated image of the scene in the 250x250 pixel frame at t=0. Disk contrast is indicated by relative brightness, with the region within 1.5λ/D of the star blacked out, similar to a coronograph simulation.
- A plot of disk contrast versus position on the x-axis, again with the central region within 1.5λ/D of the star blacked out.
- A plot of planet contrast spectra for all planets at t=0.
- A plot of planet contrast phase curves for all planets over the length of the orbital simulation.
- A plot of planet trajectories in the sky plane over the length of the orbital simulation.

Any transits and eclipses listed by the FITS file will have their ingress and egress times printed to the command line. It will also list any transits or eclipses that are in progress at the start or end of the simulation.

There are several user controls listed near the top of the `readfits.py` script, which may be edited in the source code. These controls are:

- **`lambda_ref`**: a reference wavelength in microns. The disk and planet fluxes will each be plotted at the closest wavelength point to this value. Default is 0.5.
- **`mirror_size`**: the size of the telescope mirror circumscribed diameter in meters. Default is 8.0 (based on the LUVOIR-B concept).
- **`disk_gain`**: the brightness of the disk will be multiplied by this value to highlight faint regions, at the cost of saturating bright regions. Default is 1.0.
- **`log_disk`**: if `True`, the disk brightness will be plotted based on a logarithmic scale rather than a linear one. Default is `False`.
- **`planet_bright`**: if `True`, the brightness of each planet will be scaled relative to the maximum of its phase curve. Default is `False`.
- **`color_code`**: if `True`, the planets will be color-coded based on type. (The colors are hard-coded at the top of the `readfits.py` script.) Default is `False`.
- **`fitsdir`**: the directory of FITS files that is to be enumerated for the input prompt. Default is "`./output`."

5. Input Files

5.1 The Planetary System File

The input file for a user-defined planetary system requires three sections to be specified in order: the star, the planets, and the (exozodiacal) disks, plus an optional fourth section for observational settings. (This order is required because the data structures for the later sections depend partially on the earlier ones.)

The first line of the input file should read:

```
Star
```

ExoVista expects to see this keyword before any parameters. After this, the parameters for the stellar model should be listed one per line. For example:

```
ID    999
Type  G2V
Lstar 1.0
```

Not all possible parameters are needed to build the stellar model, but a few are required. The required parameters are:

- **`Lstar`** (luminosity in M_Sun, must be <100)
- **`Dist`** (distance in pc)
- **`Vmag`** (apparent V-band magnitude)
- **`M_V`** (absolute V-band magnitude)
- Either **`Umag`** or **`Bmag`**
- At least one of **`Rmag`**, **`Imag`**, **`Jmag`**, **`Hmag`**, or **`Kmag`**
- **`Type`** (spectral type; must be a subgiant or main sequence)

Three optional parameters are also of particular interest.

- **`ID`** is an index number that is placed at the beginning of the output filename. If it is not included, it will default to zero for a single system and index starting from zero for a list of targets. *Note: the ID should be a positive number, as the hyphens in negative numbers cause parsing issues in the post-processing routines.*
- A **catalog number** (usually **`HIP`**) is not technically required, but it is likely to cause confusion in the output filenames if it is *not* included.
- **`Spectrum`** allows ExoVista's internal stellar model spectrum to be replaced by a real spectrum. Its value should be set to the spectrum file name. The spectrum file should have two columns of wavelength in nm and stellar surface flux in erg s^{-1} cm^{-2} Hz^{-1} sr^{-1}. The included `modelspectrum.dat` file provides an example spectrum for testing purposes, but it is based on ExoVista's own spectral model for a Sun-like star.

The second section is denoted by the keyword:

```
Planets
```

The next line is a header of the planet parameters in any order, which is then followed by a table of all of the planets in the system. *Note: a star cannot have more than 30 planets.* There should be 9 columns in this section. The planet parameters are:

- **`M`** (mass in M_Earth)
- **`R`** (radius in R_Earth)
- The six orbital parameters, **`a`**, **`e`**, **`i`**, **`longnode`**, **`argperi`**, and **`meananom`** (in AU and degrees where applicable)
- **`albedo`** (the albedo file for the planet found in the "`geometric_albedo_files`" subdirectory, excluding suffix)

The third section is denoted by the keyword:

```
Disks
```

The next line is a header of disk parameters in any order, which is then followed by a table of all of the disk components. *Note: a star cannot have more than 3 disk components.* The disk model is described in detail in Appendix A.3. There should be 12 columns in this section. The disk parameters are:

- **`nzodis`** (disk surface density at the location of an Earth twin, as a multiple of the zodiacal reference disk of 22 mag/arcsec^2)
- The four shape parameters, **`r`**, **`dror`**, **`rinner`**, and **`hor`** (in AU where applicable)
- The six scattering parameters, **`g0`**, **`g1`**, **`g2`**, **`w0`**, **`w1`**, and **`w2`**
- **`composition`** (the disk material found in the "`lqq_files`" subdirectory)

The fourth section may be omitted. If present, it is denoted by the keyword:

```
Settings
```

After this, similar to the "Star" section, the parameters should be listed one per line. Any parameters in the `Settings` dataclass may be included, but none are required. These settings are needed to recompute an existing scene with the same simulated observations and do not affect the planetary system directly. (The output of `parsefits.py` will include all the needed parameters automatically.)

5.2 The Stellar Target List

The stellar target list input file requires only the stellar parameters to be defined. The planets and disks will be defined procedurally. The first line of this file should be a list of columns in the star table. Each column name should be separated by whitespace and should have no internal spaces. For example:

```
ID HIP TYC2 WDS RA DEC Umag Bmag Vmag Rmag Imag Jmag Hmag Kmag
dist(pc) M_V Type Lstar(Lsun) WDSsep(") WDSdmag(mags)
pmRA(mas/yr) pmDEC(mas/yr)
```

The required column names in this file are the same as the required stellar parameters in the single system file listed in Section 5.1. However, some additional parameters may be required for specific use cases, such as sky coordinates and proper motions.

Note: there must be an additional line break after the header row to read the file correctly. This is a legacy issue due to formatting conflicts in some input files in development.

The table of stellar parameters should begin on line 3. In each row, the parameters should be separated by commas "`,`" or vertical bars "`|`". Omitted entries in the table should be listed as "NaN".

6. Data Files

A number of supplementary data files are used by ExoVista to generate planet and disk distributions and compute scenes. You may wish to modify, add to, or replace some of these files to customize your simulated universe. These supplementary files are summarized below. "`planetbins.dat`" and "`albedo_list.csv`" are the simplest files to modify and are thus described in detail.

`planetbins.dat` contains a grid of planet types in mass-semi-major axis-space. If `usebins` is enabled, the bounding boxes for this grid are used by `generate_planets()` to create exact expected numbers of planets for each desired planet type.

Note: ExoVista expects the smallest and largest bin edges in each array to be wider than the actual distribution of planets. If this is not the case, it may result in unstable behavior at the edges of the distribution.

This file should begin with the keyword **`Radius`** on the first line.

The second line is read in as the **`rbound`** array and contains a 1-D array of bin edges for planet types in radius space, in Earth radii. The included version of `planetbins.dat` defines bins of sub-Earths, super-Earths, sub-Neptunes, Neptunes, and Jupiters.

Note: it is not possible to include a bin edge between 12.28 Earth radii and the maximum planet size. This is necessary to convert radius bins to mass bins correctly.

The third line should contain a keyword either of **`Flux`** or **`Orbital_Distance`**. After this should be a table of corresponding values either of semi-major axes in AU (normalized to 1 L_sun) or of stellar instellations in F_earth. If the keyword `Flux` is included, the table is converted to semi-major axes by the equation `a = 1/sqrt(flux)`.

This table is read in as the **abound** array, and it should contain one fewer line than the length of `rbound` (that is, one for each radius bin). This allows each radius bin to have a different set of semi-major axis bins. The included version of `planetbins.dat` defines 5 bins with the middle 3 considered as "standard" definitions of hot, warm (habitable zone), and cold planets.

`albedo_list.csv` contains a list of albedo files to assign to planets and a bounding box for each in radius-semi-major-axis space in which they apply. When planets are generated by the code, each planet will be randomly assigned an albedo file from the set that overlap its position in *R*-*a*-space.

`albedo_list.csv` may be user-modified in order to define custom planet types. This file consists of a header followed by a list of planet types, and it requires seven columns:

- **`Files`**: the albedo filename associated with each planet type in the `geometric_albedo_files` subdirectory, excluding suffix.
- **`rmin`** and **`rmax`**: the minimum and maximum radius for that planet type, in Earth radii.
- **`amin`** and **`amax`**: the minimum and maximum semi-major axis for that planet type (normalized to 1 Lsun), in AU.
- **`prob`**: the probability weight for that planet type. When an albedo file is assigned, the weights of all of the eligible planet types for that planet will be summed and normalized, and a file will be randomly selected from that distribution.
- **`EEC`**: a Boolean value for whether the planet type is an exo-Earth candidate (EEC). EECs have an adjusted lower radius bound based on their instellation, changing r_{min} to r_{min}/sqrt(a), as small planets can more readily retain an atmosphere when they are colder. *Note: by default, EEC albedo files completely override non-EEC albedo files. Non-EEC albedo files can be assigned a non-zero weight by adjusting the* `eecprob` *parameter in* `Settings` *to less than 1.0.*

`phase_spread.dat` is a table of bounds for the scaling factor for the phase function. If `randphase` is enabled, for isotropic albedo files (Lambertian-defined phase functions), each planet's phase function will be randomly scaled from the Lambertian between upper and lower bounds described by this file. This table begins with the header "Phase_Function" and is followed by a table of three columns:

- Phase angle in degrees.
- Minimum scale factor on the Lambertian.
- Maximum scale factor on the Lambertian.

The included version of this file is based on the spread in phase functions of observed Solar system objects presented in Sudarsky et al. (2005).

`mamajek_dwarf.txt` is a table of median properties of Main Sequence stars by spectral type. ExoVista interpolates from this table to convert B-V colors from the stellar parameters input file to mass and effective temperature. Surface gravity is then computed from mass and radius. The citations for this table are listed in the file.

`fp00k2odfnew.pck` is a table of stellar spectrum models as a function of Teff and logg. These models are interpolated by ExoVista to generate the spectrum for each star in the input file. This file is based on the Castelli & Kurucz ATLAS9 stellar atmosphere models (IAU Symposium 210, 2003). *Note: this table does not include models for log(g)>5.25, nor for Teff<3500 K. Stars outside these limits will be excluded.*

Note: the stellar spectrum models are slated for updating in the near future.

`nominal_maxL_distribution-Dec2019.fits` is a random distribution of 300,000 exozodiacal light fluxes assigned to disks by ExoVista, based on the LBTI HOSTS survey (Ertel et al., 2020).

The **`lqq_dir`** subdirectory contains optical cross section tables for many exozodiacal disk compositions. Each file contains absorption and scattering cross sections as a function of wavelength for a specific particle size and composition. These are then compiled into complete tables for a single composition by the `generate_scene.load_lqsca()` and `generate_scene.load_lqabs()` functions.

Note: a wider selection of dust compositions is in development.

The **`geometric_albedo_files`** subdirectory contains albedo spectra for many planetary types. These files are assigned to planets by the `generate_planets.assign_albedo_files()` function, and they are used to compute the planetary spectra.

Three types of albedo files can be read by ExoVista:

1. **Isotropic albedo**: a simple 2-column file listing a spectrum of wavelength in the first column and albedo in the second column. (Isotropic albedos are converted to phase-resolved ones using a Lambert function.)
2. **Phase-resolved albedo**: these include a header listing the specific phase angles in degrees and the number of wavelength points at which the albedo is computed. *All* header rows must begin with a # symbol, and the list of phases must have the form:

   ```
   #PHASES: <phases separated by spaces>
   ```

   Each line after the header lists a wavelength followed by the albedo as a function of phase.
3. **Latitude-longitude-resolved albedo**: these include a header listing the specific longitudes (phases) and latitudes (inclinations) at which the albedo is computed, in degrees, along with the number of wavelength points. *All* header rows must begin with a # symbol, and the latitude and longitude lists must have the form:

   ```
   #LATITUDES: <latitudes separated by spaces>
   #LONGITUDES: <longitudes separated by spaces>
   ```

Each line after the header includes a wavelength and a linearized array of albedo as a function of latitude and longitude (longitude on the inner dimension).

Note: a wider selection of albedo files is in development.

The **`occurrence_rates`** directory contains tables of occurrence rates of exoplanets which are used to generate random planets with the correct distribution of parameters. Standard, optimistic, and pessimistic occurrence rates are included. The choice of occurrence rates used may be changed by setting the **bound** argument in `generate_planets.load_occurrence_rates()` to "`upper`" or "`lower`." The standard rates are used by default.

7. Output Files

For each star, `generate_scene.generate_scene()` produces a single FITS file. The sizes of these files vary with the spectral resolution of the planets and disks and the integration time for the N-body integrator. For the default spectral resolutions and an integration time of 10 years, the filesize is about 25 MB. Ideally, this will be roughly the largest file size produced by "standard" use cases.

The FITS files have multiple extensions. The data contained in each extension is as follows:

Extension #	Data description
0	Vector of wavelength values for star and planets
1	Vector of wavelength values for debris disk
2	3D debris disk contrast cube (x * y * lambda + noise map)
3	2D transit/eclipse events array (num. events * 4)
4	2D star data array (time * position + orbit + spectrum)
5…N_EXT	2D planet data array (time * position + orbit + spectrum)

Extension 0: Wavelength values for star and planets
Description: a 1D vector containing the wavelengths (in microns) at which stellar flux and planet contrast were calculated

Key header parameters:
NAXIS1: length of wavelength vector (# of wavelengths)
VERSION: the version number of the code (used to set up the post-processing routines)
N_EXT: maximum extension number (with planet data being in extensions 4 – N_EXT)
SPECRES: spectral resolution of wavelength vector
LAMMIN: minimum wavelength

LAMMAX: maximum wavelength

Extension 1: Wavelength values for debris disk contrast cube
Description: a 1D vector containing the wavelengths (in microns) at which disk contrast was calculated

Key header parameters:
NAXIS1: length of wavelength vector (# of wavelengths)
SPECRES: spectral resolution of wavelength vector
LAMMIN: minimum wavelength
LAMMAX: maximum wavelength

Extension 2: Debris disk contrast cube
Description: a 3D cube (xpix * ypix * wavelengths+1) of disk flux divided by star flux. To convert this back into a disk flux, interpolate the cube to the desired wavelengths, then multiply by the stellar flux at those wavelengths. Note: The number of entries in the last dimension is equal to the number of wavelengths+1 because the last entry is not a contrast map, but a 2D map estimating the fractional numerical noise in the contrast calculations.

Key header parameters:
NAXIS1: # of pixels in x dimension
NAXIS2: # of pixels in y dimension
NAXIS3: # of wavelengths+1
SPECRES: spectral resolution of wavelength vector
PXSCLMAS: pixel scale in milli-arcseconds
IWA: inner working angle of the coronagraph in arcseconds
DUSTBLOW: the dust blowout particle size in microns
TSUB: the dust sublimation temperature in kelvins
LNGND-N: longitude of the ascending node of the Nth disk component (degrees)
I-N: inclination of the Nth disk component relative to system midplane (degrees)
NZODIS-N: density in zodis of the Nth disk component
R-N: circumstellar distance of the peak density of the Nth disk component (AU)
DROR-N: value of the normalized Gaussian peak width of the Nth disk component
RINNER-N: value of the inner truncation radius of the Nth disk component (AU)
ETA-N: ratio of PR drag time to collision time for the blowout size for the Nth disk component
HOR-N: normalized scale height for the Nth disk component
G0-N – G2-N: 3 values of scattering asymmetry parameters for the Nth disk component
W0-N – W2-N: 3 weights for each HG scattering phase function for the Nth disk component
MINSIZE: minimum grain size considered
MAXSIZE: maximum grain size considered

Extension 3: transit and eclipse times
Description: a 2D array (number of events * 4) containing the time of each ingress and egress of a planet in transit or secondary eclipse, the number of the planet involved, and two values indicating the specific type of event. The data structure is organized as follows:

`data[0,i]`: the time of the event in days
`data[1,i]`: the numerical designation of the planet involved in the event
`data[2,i]`: the position of the planet before the event: `+1` for transit, `-1` for eclipse, or `0` for neither
`data[3,i]`: the position of the planet after the event: `+1` for transit, `-1` for eclipse, or `0` for neither

If a planet is in transit or eclipse at the start or end of the integration time, the third and fourth values will be `(+1, +1)` or `(-1, -1)`, respectively.
If there are no transits or secondary eclipses in the integration time (which is likely to be the case for most systems), this structure will contain a single event with all values set to `0`.

Key header parameters:
NAXIS1: the number of ingress and egress events recorded
NAXIS2: the number of entries for each event (4)
BASELINE: the length of integration time for the file, in days

Extension 4: star data
Description: a 2D array (time x wavelengths+15) containing the time, position, orbit, and spectrum of the star for all time values. The data structure is organized as follows:

`data[i,j]`: ith time value, jth data value
`data[i,0]`: simulation time (years)
`data[i,1]`: x coordinate of star (in pixels) at ith time
`data[i,2]`: y coordinate of star (in pixels) at ith time
`data[i,3:9]`: heliocentric orbital elements at ith time (set to zero for the star, but used for the planets)
`data[i,9:15]`: barycentric x, y, z positions (in AU) and barycentric vx, vy, vz velocities (in AU/yr) at ith time
`data[i,15]`: placeholder value for phase angle
`data[i,16:16+nlambda]`: spectrum of star (in Jy) at ith time

Key header parameters:
NAXIS1: # of time values
NAXIS2: # of data values (wavelengths+15) for each time value
PA: position angle of system midplane (degrees)
I: inclination of system midplane (degrees)
STARID: an internal catalog ID # for the star
RA: right ascension of star (decimal degrees)
DEC: declination of star (decimal degrees)
*MAG: stellar empirical magnitude in the * filter band

M_V: absolute V-band magnitude of star
DIST: distance to star (pc)
TYPE: spectral type of star
LSTAR: bolometric stellar luminosity (solar luminosities)
TEFF: stellar effective temperature (K)
ANGDIAM: angular diameter of star (mas)
MASS: stellar mass (solar masses)
LOGG: log(stellar gravity) (cm/s^2)
RSTAR: stellar radius (solar radii)
WDS_SEP: most recent separation of companion in WDS catalog, if it exists (arcsec)
WDS_DMAG: delta mag of companion in WDS catalog, if it exists
PMRA: proper motion in RA (mas/yr)
PMDEC: proper motion in DEC (mas/yr)
PXSCLMAS: pixel scale (mas)

Extension 5 – N_EXT: planet data
Description: a 2D array (time x wavelengths+15) containing the time, position, orbit, and **contrast** spectrum of each planet for all time values. The data structure is organized as follows:

- `data[i,j]`: ith time value, jth data value
- `data[i,0]`: simulation time (years)
- `data[i,1]`: x coordinate of planet (in pixels) at ith time
- `data[i,2]`: y coordinate of planet (in pixels) at ith time
- `data[i,3:9]`: heliocentric orbital elements at ith time; semi-major axis (AU), eccentricity, inclination (degrees), longitude of ascending node (degrees), argument of pericenter (degrees), mean anomaly (degrees)
- `data[i,9:15]`: barycentric x, y, z positions (in AU) and barycentric vx, vy, vz velocities (in AU/yr) at ith time
- `data[i,15]`: phase angle of the planet at ith time in degrees
- `data[i,16:16+nlambda]`: **contrast** spectrum of planet at ith time

Key header parameters
NAXIS1: # of time values
NAXIS2: # of data values for each time value
M: planet mass (Earth masses)
R: planet radius (Earth radii)
ALBEDO_F: geometric albedo filename
PXSCLMAS: pixel scale (mas)

Appendices

A. Scientific and Computational Models

A.1 Stellar Target List

The stellar target list for ExoVista is intended to be user-defined. However, the `master_target_list-usingDR2-50_pc.txt` file provided with the code contains the HabEx/LUVOIR master target list generated by combining the *Hipparcos* and *Gaia* DR2 target lists within 50 pc. *Note: this target list is not intended to be accurate on a star-by-star basis*, and indeed, ExoVista internally uses model spectra rather than observed spectra to simulate the stars. However, it should be representative of nearby main sequence and subgiant stars.

A.2 Planet Generation

Planets are generated by ExoVista with orbital parameters defined relative to the system midplane orientation. Planets are drawn from the occurrence rate maps of Dulz et al. (2020) in mass-semi-major axis space (for a 1 L_sun star), converted to a probability distribution. If bins for planet types are not used, these maps are divided into a regular grid. Applying bins imposed an extra set of bin edges on this grid, and the occurrence rate in each "pixel" is scaled according to its size in log-log space.

The Monte Carlo draw defines a 3-D grid of random numbers, covering each pixel in the 2-D occurrence rate map and the total number of stars. Random numbers smaller than the occurrence rate in a given pixel assign one planet to that star with the corresponding parameters. *Note: this means that the occurrence rate map must be a high enough resolution that the expected number of planets per star is <1 for every pixel.* A planet's exact parameters within its pixel are assigned randomly in log-log space.

Planets are drawn from the mass grid even though radii are better known because M(R) is not a monotonic function, so there is ambiguity in converting radius to mass as opposed to the inverse. The default mass-radius relation is a strictly 1-dimensional function based on Chen & Kipping (2017). If `randrad` is enabled, random variation will be added to this function based on the "fractional dispersion" hyperparameters from that paper.

When generating the pixel map of drawn planets per star, if planet types are specified, ExoVista will check the number of planets generated for each type, and it will randomly add planets to empty (3-D) pixels or remove them from filled pixels until the number for that planet type equals the expected number. Orbital elements are then assigned randomly. However, eccentricities are set to zero by default, and inclinations are restricted to <5 degrees by default.

Once the planets are generated, they are randomly assigned to stars in the target list, with their semi-major axes being scaled according to the host star's luminosity. After adding the planets to stars, their orbital stability is checked based on a mutual Hill sphere heuristic. For any pair of planets with $\Delta > 6$, where Δ = separation/mutual Hill radius, the less massive planet is removed. After this stability check, new planets are drawn to replace the removed ones, and the process is iterated until all planets in the draw have been assigned and are stable. (If it is not possible to assign all of the planets in the draw, the loop halts after 50 iterations pass with no net planets successfully added, or after 200 total iterations.)

Note: Dulz et al. (2020) adopted a stability criterion of $\Delta > 9$, but this greatly slows the process of assigning planets due to the large number of rejections, so ExoVista adopts a more lenient criterion. The resulting systems are expected to be "plausible," but are not necessarily stable on Gyr timescales.

Note: ExoVista does not assume any correlation between planet types or multiplicities within a given system. All planets are equally likely to occur with any type of neighbors (subject to the stability criterion).

A.3 Disk Model

The debris disk data structure contains 3 parameters not listed in the individual planetary system files. These are **`n`**, the number of disk components, and **`longnode`** and **`i`**, which define the disk orientation. At present, `longnode` and `i` are not implemented, and all disks are assigned the same orientation as the system midplane.

Note: ExoVista assumes optically thin disks, such that surface brightness is proportional to density. By default, each star is assigned a 2-component debris disk, in which the inner component is required and corresponds to our Solar system's Zodiacal disk. This inner component is also assigned an absolute limit in extent from 0.5-5.0 AU. The outer component is assigned an extents limit of 5-50 AU, and a third component, if present, is assigned an extents limit of 50-500 AU. These values can be changed by using the **`r_min`** and **`r_max`** parameters in `Settings`.

The inner component of each disk is randomly assigned a density relative to our Zodiacal disk based on the exozodi distribution of on the LBTI HOSTS survey (Ertel et al., 2020), which has a median of 3 zodis. The outer component is assigned a density that is a multiple of the inner component between 0.2 and 5.0, randomly distributed in log-space. The disk composition is randomly selected from the available compositions in the `lqq_files` directory.

For each star, the disk model is structured with a belt shape in a location selected from the regions of stability between planets within that disk's extents limit. "Stable" regions are defined as all regions more than 3 Hill radii away from any planet. This belt is described by 4 shape parameters:

- **`r`**: the radius of peak density (i.e. the center of the belt), selected randomly (in log-space) from all stable radii within the disk's extents limits, subject to a minimum fractional belt width of dr/r > 0.05.
- **`dror`:** the fractional width of the belt, dr/r, randomly selected from the range 0.05-0.30, subject to the stability limits.
- **`rinner`**: the inner truncation radius of the disk caused by gravitational perturbations by interior giant planets. This is set to zero if there are no planets interior to `r` with a mass >100 Earth masses. If interior giant planets are present, it is set to `min(r,1.1*a*(1+e))` for the most massive interior planet.
- **`hor`**: the vertical fractional scale height of the disk, h/r, randomly selected from the range 0.03-0.2.

Note: if no value of `dror` *in the allowed range meets the stability criteria, the disk density is reset to zero, and no disk is included.*

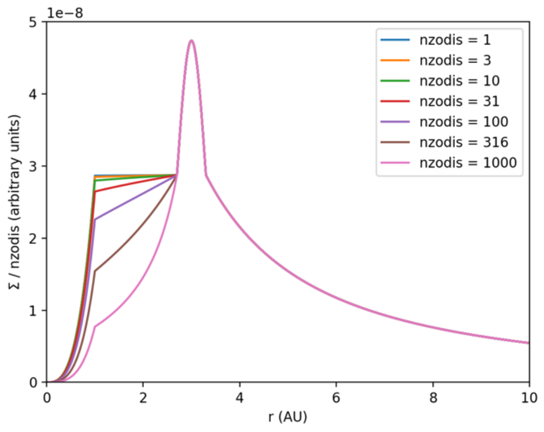
The disk density within the belt is assigned a Gaussian density distribution with circumstellar radius r, and a Dohnanyi particle size distribution. Exterior to the belt, the density falls off with a power law of r^-1.5. Interior to the belt, the density falls off based on a collisional Poynting-Robertson drag model from Wyatt. An additional parameter, `eta`, is the ratio of the Poynting-Robertson timescale to the collision timescale, and is computed analytically. Finally, interior to the truncation radius (if present), the density falls off faster, with a power law of r^3.

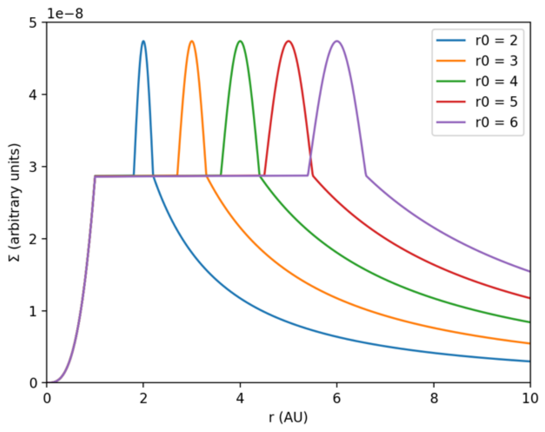
Note: ExoVista does not model mean motion resonant ring structures, and all disks are assumed to be circular and azimuthally symmetric. This is necessary in part to avoid the need to model the disk over time.

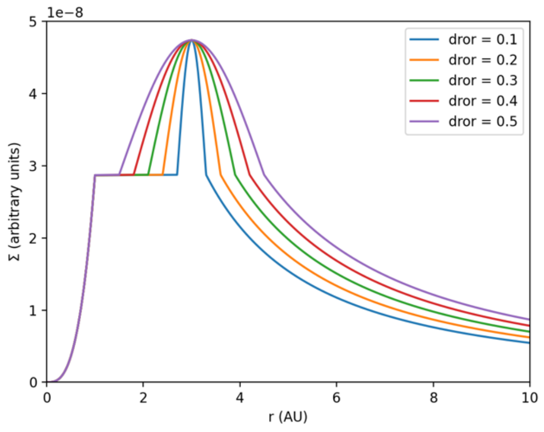
Asymmetric scattering of starlight off dust particles in the disk is modeled by a linear combinationsof 3 Henyey-Greenstein phase functions. The asymmetry parameters of these functions, **`g0`**, **`g1`**, and **`g2`**, are randomly selected from the ranges 0.8 to 0.995, 0.35 to 0.8 and -0.30 to 0.35, respectively. The weights on the scattering functions are randomly selected such that **`w0`** falls in the range 0.4 to 0.8; **`w1`** falls in the range 0.1 to `1-w0`, and **`w2`**=`1-w0-w1`.

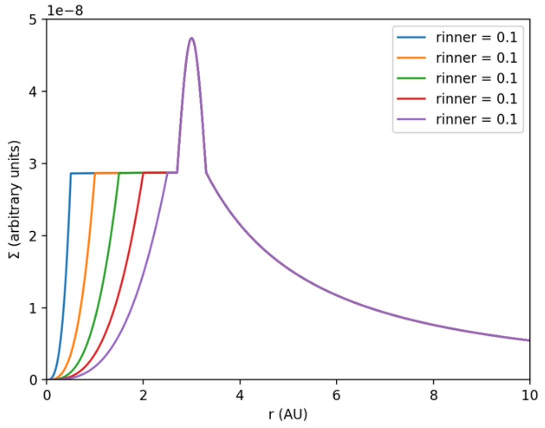
ExoVista models only scattering of light by the disk, not emission or absorption, due to the lower spectral resolution needed to accurately model scattering. Additionally, it does not rely on Mie scattering due to the computationally-intensive modelling required for it and instead reads scattering coefficients from files. The full scattering spectrum is assembled from coefficients for many different grain sizes with a size resolution of s/ds = 5. The range of grain sizes is determined by the files in `lqq_dir`, which extends from the blowout size (assumed to be 0.5 microns for all stars) to 100 times the maximum wavelength observed (thus, by default 100 microns).

Plots illustrating the effect of the four disk shape parameters (`nzodis`, `r0`, `dror`, and `rinner`) on the radial disk surface density profile are shown on the next page.


1e−8
Σ / nzodis (arbitrary units)
r (AU)
nzodis = 1
nzodis = 3
nzodis = 10
nzodis = 31
nzodis = 100
nzodis = 316
nzodis = 1000


1e−8
Σ (arbitrary units)
r (AU)
r0 = 2
r0 = 3
r0 = 4
r0 = 5
r0 = 6


1e−8
Σ (arbitrary units)
r (AU)
dror = 0.1
dror = 0.2
dror = 0.3
dror = 0.4
dror = 0.5


1e−8
Σ (arbitrary units)
r (AU)
rinner = 0.1
rinner = 0.1
rinner = 0.1
rinner = 0.1
rinner = 0.1

B. ExoVista API

B.1 Module `load_stars`

`load_stars.load_target_list(target_list_file)`

Internal function used by `load_stars()` to read in the target list file and convert it to a DataFrame for further processing. It also computes color values and angular diameters.

Parameters:

`target_list_file`: *string*; file name for list of target stars

Returns:

`target_list`: *DataFrame*; the table of stellar parameters for stars that pass the filter

`load_stars.load_stars(target_list_file)`

Reads in a table of target star parameters and converts it to the DataFrame format used by ExoVista. It filers out stars that have insufficient data or are outside the limits for ExoVista. It also computes necessary the derived quantities, `mass`, `Teff`, `rstar`, and `logg` for each star.

Parameters:

`target_list_file`: *string*; file name for list of target stars

Returns:

`target_list`: *DataFrame*; the table of stellar parameters for stars that pass the filter

`nexozodis`: *array (floats)* or *None*; a list of exozodiacal densities to replace the randomly generated ones. If the “nexozodis” is not included in the target list file, this has a value of None.

B.2 Module `generate_planets`

`generate_planets.mass_to_radius(M, rng, randrad=False)`

Converts planetary masses to radii using a three-part piecewise empirical function. This is used to convert occurrence rates from mass-space to radius-space.

Parameters:

> **`M`**: *array (floats)*; an array of planet masses.
>
> **`rng`**: `numpy.random.default_rng` *object*; random number generator created by `generate_planets()`, which carries the same random seed over to `mass_to_radius()`.
>
> **`randrad`**: *bool*; if `True`, random variation is added to the computed planet radii, carried over from the main `Settings` object; default is `False`. *Note: not compatible with* `usebins`.

Returns:

> **R**: *array (floats)*; a corresponding array of planet masses.

`generate_planets.radius_to_mass(R)`

Converts planetary radii to masses using the inverse of the piecewise mass-to-radius function. This is used to convert boundaries for planet types from radius-space to mass-space in order to divide the occurrence rate grid accurately. *Note: because the inverse function M(R) is not monotonic, this function cannot be evaluated for radii >12.28 R_J and will return a value of -1 for these radii.*

Parameters:

> **R**: *array (floats)*; an array of planet radii.

Returns:

> **`M`**: *array (floats)*; a corresponding array of planet radii.

`generate_planets.generate_planets(stars, settings, bound='', nomcdraw=False, addearth=False, usebins=False, subdivide=1)`

Creates a randomized array of planets and orbital parameters associated with the input star list. Each star is given a list of 30 planets to ensure that there are enough to cover all of the planets assigned to the star. Unassigned slots are set to zeros. The planets are generated iteratively, first creating a random sample of the expected number of planets and assigning them to the stars, then removing those that are dynamically unstable. This process is then repeated until the final number of planets matches the expected number, or no new planets can be added without creating dynamical instabilities.

Parameters:

`stars`: *DataFrame*; a table of stellar parameters

`settings`: `Settings` *object*; contains all of the settable code parameters.

`bound`: *string, optional*; set to 'upper' or 'lower' to use the optimistic or pessimistic bounds of the planet occurrence rates, respectively. Default is '' (empty string), which uses the nominal occurrence rates.

`nomcdraw`: *bool, optional*; if `True`, ExoVista bypasses the Monte Carlo draw and assigns planets to stars strictly based on occurrence rates. Default is `False`.

`addearth`: *bool, optional*; if `True`, ExoVista will add an Earth twin to every planetary system. These Earth twins have negligible mass, so they do not disrupt stability or disk structure. Their orbital distance is scaled to the luminosity of the star, and they have zero eccentricity so that the phase angle is exactly lined up with quadrature at t=0. Default is `False`.

`usebins`: *bool, optional*; if `True`, ExoVista will assign planets to bins in radius and semi-major axis based on planet types instead of randomly to the whole grid. These bins are read in from the file `planetbins.dat`. The occurrence rate will be adjusted from the Monte Carlo draw so that each planet type bin includes exactly the expected number of planets from the occurrence rate table (unless stability constraints prevent this). Default is `False`.

`subdivide`: *int, optional*; multiplies the grid size of the occurrence rate distribution used to assign planets. (The tables have a native resolution of 100x100 in M-a space.) Minimum value is 1; maximum value is 10. Default is 1.

Returns:

`plorb`: *3-D array (floats)*; planet parameters (nstars x 30 x 8)

`albedos`: *2-D array (floats)*; albedo file names assigned to planets (nstars x 30)

`generate_planets.load_occurrence_rates(subdivide=1, bound='', mass=True, usebins=False)`

Wrapper that calls the correct version of `load_occurrence_rates` to read the desired table of occurrence rates. (Currently, only the M-a-space table is implemented.) The bound and `mass` parameters are used to generate the name of the correct file. (*Note: this wrapper is not needed in the current version of the code, but it is included to allow the possibility of future expansion with different types of occurrence rate files such as those in R-P-space.)*

Parameters:

`subdivide`: see below

bound: *string, optional*; set to 'upper' or 'lower' to use the optimistic or pessimistic bounds of the planet occurrence rates, respectively. Default is '' (empty string), which uses the nominal occurrence rates.

`mass`: *bool, optional*; if `True`, the occurrence rates are read in M-a-space. If `False`, the occurrence rates are read in R-P-space and then converted to M-a-space for the purpose of generating planet parameters. Default is `True`.

`usebins`: *bool, optional*; if `True`, ExoVista will assign planets to bins in radius and semi-major axis based on planet types instead of randomly to the whole grid. Default is `False`.

Returns:

Output of desired `load_occurrence_rates` function.

`generate_planets.load_occurrence_ratesMA(filename, subdivide=1, usebins=False)`

Reads in a table of planet occurrence rates in radius-period space, which is used to distribute planets to the stars.

Parameters:

`filename`: *string*; the name of the occurrence rate file to be read. Generated automatically by `generate_planets.load_occurrence_rates()`.

`subdivide`: *int, optional*; multiplies the grid size of the occurrence rate distribution used to assign planets. Minimum value is 1; maximum value is 10. Default is 1.

`usebins`: *bool, optional*; if `True`, ExoVista will assign planets to bins in radius and semi-major axis based on planet types instead of randomly to the whole grid. Default is `False`.

Returns:

> **`newoccrate`:** *2-D array (floats)*; an array of planet occurrence rates in M-a-space or R-a-space of the specified grid size.
>
> **`medge`** and **`aedge`**: *2-D arrays (floats)*; arrays of R- or M-values and a-values of the occurrence rate table, respectively. These two arrays combine to give the ordered pairs of the coordinates in the occurrence rate map.
>
> **`mmid`** and **`amid`**: *2-D arrays (floats)*; arrays of R- or M-values and a-values of the occurrence rate table, respectively. These two arrays combine to give the ordered pairs of the coordinates in the occurrence rate map.

`generate_planets.add_planets(stars, plorb, expected, orM_array, ora_array, hillsphere_flag, randrad, rng):`

Adds planets to the planets array based on the occurrence rates. This function fills out the array up to the expected number of planets randomly, without looking at the other planets in the system. All added planets are flagged to check their stability later.

Parameters:

> **`stars`**: *DataFrame*; a table of stellar parameters to which to assign the planets.
>
> **`plorb`**: *3-D array (floats)*; the table of current planets. New planets will be added to this table starting from the first vacant (all zeros) entry for each star.
>
> **`expected`**: *2-D array (ints)*; the 2-D histogram of total planets expected for each gridpoint in R-P space, rounded to the nearest integer.
>
> **`orM_array`** and **`ora_array`**: *arrays (floats)*; bin edge arrays of M-values and a-values of the occurrence rate table, respectively. These two arrays combine to give the ordered pairs of the coordinates in the occurrence rate map. Planets will be assigned to random coordinates between the bin edges of the appropriate "pixel" of the grid.
>
> **`hillsphere_flag`**: *2-D array (bools)*; a boolean table indicating where new planets have been added. The function will set the entries corresponding to all added planets to `True` and all existing planets to `False`.
>
> **`randrad`**: *bool*; if `True`, random variation is added to the computed planet radii, carried over from the main `Settings` object; default is `False`. *Note: not compatible with* `usebins`.

`rng`: `numpy.random.default_rng` *object*; random number generator created by `generate_planets()`, which carries the same random seed over to `add_planets()`.

Returns:

`plorb`: *3-D array (floats)*; the updated table of planet parameters.

`hillsphere_flag`: *2-D array (bools)*; the updated Hill sphere (new planet) flag table.

`generate_planets.remove_unstable_planets(stars, plorb, hillsphere_flag)`

Checks the stability of all newly added planets in the plorb table and removes those that show orbital instability due to overlap of mutual hill spheres. The function always removes the less massive planet of an unstable pair. When a planet is removed, it also re-checks the next planet inward, in case it is mutually unstable with a larger outer planet.

Parameters:

`stars`: *DataFrame*; a table of stellar parameters to which to assign the planets.

`plorb`: *3-D array (floats)*; the table of current planets. Unstable planets will be removed from this table and the table re-sorted for each star to remove gaps in the list.

`hillsphere_flag`: *2-D array (bools)*; a boolean table indicating where new planets have been added, which must therefore be checked for stability.

Returns:

`plorb`: *3-D array (floats)*; the updated table of planet parameters.

`generate_planets.assign_albedo_file(stars, plorb, rng)`

Assigns an albedo file to each planet in the population generated for the input list of stars. Albedo files are assigned randomly from the subset that apply to a given planet's radius and semi-major axis.

Parameters:

`stars`: *DataFrame*; a table of stellar parameters to which planets have been assigned.

`plorb`: *3-D array (floats)*; the table of planet parameters to be assigned albedo files.

`rng`: `numpy.random.default_rng` *object*; random number generator created by `generate_planets()`, which carries the same random seed over to `assign_albedo_file()`.

Returns:

`plalbedo`: *2-D array (strings)*; table of albedo files assigned to each planet

B.3 Module `generate_disks`

`generate_disks.generate_disks(stars, planets, settings, nexozodis=None, rand_components=False)`

Creates a randomized array of debris disks associated with the input star list. Each star is given a list of 3 disk components, corresponding roughly to the warm dust, cold dust, and high-inclination cold dust components of our Solar system. The warm dust component is always assigned, while unassigned slots are set to zeros. The disks are generated randomly, but are constrained by the dynamics imposed by the planets. All disk components in the same system are assigned the same composition.

Parameters:

`stars`: *DataFrame*; a table of stellar parameters to which planets have been assigned.

`planets`: *3-D array (floats)*; the table of planet parameters to be assigned albedo files.

`settings`: `Settings` *object*; contains all of the settable code parameters.

`nexozodis`: *1-D array (floats)*; the pre-assigned list of dust densities given by the stellar targets file, if present. If None, `nexozodis` is assigned randomly. Default is None.

`rand_components`: *bool*; if `True`, the number of disk components will be randomly selected for each star within the allowed range. Default is `False`.

Returns:

`disks`: *3-D array (floats)*; table of disk parameters (nstars x 3 x 15).

`compositions`: *array (strings)*; list of compositions assigned to disks for each star.

B.4 Module `generate_scene`

Generates the scene of a planetary system with stellar spectrum, planetary contrast spectra, per-pixel disk contrast spectra, and transit and eclipse times. Both stellar and planetary spectra are computed per-timestep, although the stellar spectrum does not change. This scene is the “real” or “exact” image with no PSF or throughput function applied, although it does require a pixel scale to be specified. It may be used to model detection, characterization, and other observation methods with a simulated pipeline.

`generate_scene.load_lqsca(lqq_dir, composition, rdust, rdust_boundaries, lam)`

Loads the scattering cross-section table for the specified disk composition. Absorption and scattering cross sections as a function of wavelength are listed for each composition and particle size in the `lqq_dir` directory. The `load_lqsca()` function reads in the files for each particle size for the given composition and builds a 2-D table of scattering cross sections.

Parameters:

`lqq_dir`: *string*; the name of the directory where the cross-section files are found.

`composition`: *string*; the name of the disk composition.

`rdust`: *array (floats)*; the particle sizes used to build the cross-section table. By default, this is based on an array defined in `defaults.py`.

`rdust_boundaries`: *2-D array (floats)*: the upper and lower bounds of each particle size bin (used to construct the filenames of the cross-section files).

`lam`: *array (floats)*; the array of wavelengths used to build the cross-section table.

Returns:

`Qsca_array`: *2-D array (floats)*; table of particle cross scattering sections (nsizes x nlambda).

`generate_scene.load_lqabs(lqq_dir, composition, rdust, rdust_boundaries, lam)`

Loads the absorption cross-section table for the specified disk composition. Absorption and scattering cross sections as a function of wavelength are listed for each composition and particle size in the `lqq_dir` directory. The `load_lqabs()` function reads in the files for each particle size for the given composition and builds a 2-D table of scattering cross sections.

Parameters:

`lqq_dir`: *string*; the name of the directory where the cross-section files are found.

`composition`: *string*; the name of the disk composition.

`rdust`: *array (floats)*; the particle sizes used to build the cross-section table. By default, this is based on an array defined in `defaults.py`.

`rdust_boundaries`: *2-D array (floats)*: the upper and lower bounds of each particle size bin (used to construct the filenames of the cross-section files).

`lam`: *array (floats)*; the array of wavelengths used to build the cross-section table.

Returns:

`Qabs_array`: *2-D array (floats)*; table of particle absorption cross sections (nsizes x nlambda).

`generate_scene.lambertian(beta)`

Simple function to compute the Lambertian reflectivity for a given angle.

Parameters:

`beta`: *array (floats)*; a list of angles to compute.

Returns:

`phi`: *array (floats)*; the corresponding reflectivity for each angle.

`generate_scene(stars, planets, disks, albedos, compositions, settings)`

The main function for `generate_scene` and the direct output function for ExoVista as a whole. This function takes all of the parameters of planetary systems generated by or read into the code, sets up the relevant arrays and constants, computes the corresponding scene for each star over time, and outputs the results as a FITS file. (This is also the function that loops over the individual stars.)

Parameters:

`stars`: *DataFrame*; the data structure of stellar properties.

`planets`: *3-D array (floats)*; the array of planet properties for each system.

`disks`: *3-D array (floats)*; the array of disk properties for each system.

`albedos`: *2-D array (strings)*: the albedo file assigned to each planet.

`compositions`: *array (strings)*; the disk composition assigned to each system.

`settings`: `Settings` *object*; contains all of the settable code parameters.

Returns: None (FITS files are written to disk.)

`distribute_diskpoints(s, disk, rdust, drdust, Qsca, xcen=0, ycen=0, xwidth=settings.npix, ywidth=settings.npix)`

Computes the disk contrast data cube for a given planetary system. This is an array of per-pixel contrast of the dust disk with the star over a low-resolution spectrum. A low spectral resolution is used to save memory because dust scattering spectra tend not to have narrow features. *Note: this function utilizes the Cython class* `PyImage`.

Parameters:

`s`: *DataFrame entry*; the data structure of stellar properties for a single star.

`disk`: *2-D array (floats)*; the array of disk properties for the planetary system.

`rdust`: *array (floats)*; the particle sizes used to build the cross-section table.

`drdust`: *array (floats)*; the widths of the `rdust` bins.

`Qsca`: *2-D array (strings)*: the dust cross section table, normally that returned by `load_lqsca()`.

`xcen` and **`ycen`**: *floats*; disk-relative coordinates of the center of the disk image to be calculated, **in AU**. *Note that these coordinates are relative to the star, not to the corner of the frame like the pixel coordinates.* They are rotated so that `xcen` lies along the disk's axis of inclination, and `ycen` lies perpendicular to that axis. Defaults are `0` and `0`.

`xwidth` and **`ywidth`**: *floats*; size of disk image to be calculated **in pixels**. This allows a subset of the full frame to be calculated to reduce execution time. (For some applications, such as calibration, only a "postage stamp' of the frame is required.) *Note that these coordinates are aligned with the image frame and not rotated.* Defaults are both `settings.npix`.

Returns:

`masterimg`: *3-D array (floats)*; the disk contrast spectrum data cube (npix x npix x nlambda_disk).

`rgen(numx, numy=0)`

Generates an array of radial coordinates associated with an x-y pixel coordinate grid, with the star at (0,0). This is converted to an array of angular separations from the star used by `distribute_diskpoints()` to compute the disk flux at each pixel.

Parameters:

`numx`: *int*; the size of the pixel array in the x-dimension.

`numy`: *int*; the size of the pixel array in the x-dimension. Default value is `0`, which is reset to `numx`.

Returns:

`r`: *2-D array (floats)*; the grid of radial coordinates of each pixel.

`x` and **`y`**: *arrays (floats)*; the lists of x- and y-coordinates used to build the grid.

`get_stellar_flux(s, lam, path='./', spectrum=None)`

Note: this function is slated for rewriting with an improved model. Computes the stellar spectrum for a given star from the Kurucz & Castelli ATLAS9 stellar atmosphere models. *Note that unlike the other spectra, which are contrast values, the stellar spectrum is computed in units of janskys.*

Parameters:

`s`: *DataFrame entry*; the stellar parameters to compute the spectrum.

`lam`: *array (float)*; the wavelength points to compute the spectrum.

`path`: *string*; the location of the Kurucz & Castelli model table. Default value is the current directory.

`spectrum`: *string*; if defined, sets a stellar spectrum file to be used in place of the built-in models. Default is None.

Returns:

`interplambda`: *array (floats)*; the output wavelength array on which the spectrum is computed. This is the union of the built-in wavelength array of the table, the requested `lam` array for the ExoVista output, and the `transition_lambda` transition wavelength values.

`fstar`: *array (floats)*; the stellar flux on the `interplambda` array.

`getkurucz(teff, logg, metallicity=0.0)`

Note: this function is slated for rewriting with an improved model, especially for temperatures <3500 K, and to add more metallicity values. Interpolates the Kurucz & Castelli spectrum table to the star's effective temperature, surface gravity, and metallicity.

Parameters:

> **`teff`**: *float*; the effective temperature of the star.
>
> **`logg`**: *float*; the log-surface gravity of the star.
>
> **`metallicity`**: *float*; the metallicity of the star in dex. Default value is `0.0`. *Note: currently returns an error if not set to zero.*

Returns:

> **`lam`**: *array (floats)*; the wavelength array of the Kurucz & Castelli table.
>
> **`Bnu`**: *array (floats)*; the stellar flux on the `lam` array in units of (erg s^{-1} cm^{-2} Hz^{-1} sr^{-1}).

`read_albedo_file(filename)`

Reads in the albedo file for a given planet. There are three types of albedo file: isotropic, phase-resolved, and latitude-longitude-resolved. These return 1-D, 2-D, and 3-D arrays of albedo values, respectively. This function handles all three formats, and there are corresponding sections of `thread_the_scene()` that convert each one to a planetary spectrum.

Parameters:

> **`filename`**: *string*; the name of the albedo file.

Returns:

> **`lam`**: *array (floats)*; the wavelength array of the albedo file.
>
> **`phi`**: *array (floats)*; the array of phase/longitude values of the albedo file. If the file is not phase-resolved, this is an array of length 1.
>
> **`lat`**: *array (floats)*; the array of latitude values of the albedo file. If the file is not latitude-resolved, this is an array of length 1.
>
> **`gI`**: *1-D, 2-D, or 3-D array (floats)*; the array albedo values from the file.

B.5 Module `read_solarsystem`

`read_solarsystem(settings, system_file='example_system.dat')`

Reads in a file containing the parameters for a single planetary system and converts it to the five data structures used by `generate_scene()`. *Note: each output array has one more dimension than the shape of its data structure would suggest. In each case, the outermost dimension has length one and is included for compatibility reasons, as* `generate_scene()` *expects a list of systems.*

Parameters:

> **`settings`**: `Settings` *object*; contains all of the settable code parameters.
>
> **`system_file`**: *string*; the file from which to read in the planetary system parameters. Default value is “`example_system.dat`”, which contains parameters for our Solar system.

Returns:

> **`stars`**: *DataFrame*; the data structure of stellar properties.
>
> **`planet`**: *3-D array (floats)*; the array of planet properties.
>
> **`disks`**: *3-D array (floats)*; the array of disk properties.
>
> **`albedos`**: *2-D array (strings)*: the albedo file assigned to each planet.
>
> **`compositions`**: *array (strings)*; the disk composition.

B.6 Module nbody

`nbody(cartin, GM, R, istar, curr_time, desired_time)`

Takes one timestep of the N-body integrator. Specifically, it sets up the `state` matrix and calls the integration functions repeatedly to reach the desired precision. It also calls `detect_transits()` to find transit and eclipse times.
Note: this function utilizes the Cython class `PyIntegrator`*.*

Parameters:

> **`cartin`**: *list of arrays (floats)*; the (barycentric) cartesian position and velocity coordinates of the star and planets at the stars of the timestep.

`GM`: *array (floats)*; the GM mass parameter for the star and planets.

`R`: *array (floats)*; the radii for the star and planets.

`istar`: *float*; the inclination of the system midplane.

`curr_time`: *float*; the current time at the start of the timestep.

`desired_time`: *float*; the desired time at the end of the timestep.

Returns:

`cartout`: *list of arrays (floats)*; the cartesian position and velocity coordinates of the star and planets at the end of the timestep.

`tlist`: *list (floats)*; a list of times of transit and eclipse events within the timestep, in years.

`transmaster`: *list of lists (ints)*; a list containing a list of status for each planet over the times in `tlist`: `+1` for transit, `-1` for secondary eclipse, `0` for no event.

`detect_transits(R, istar, state)`

Determines whether any planets are in transit or secondary eclipse at a given integrator step by measuring the coordinate difference between the planets and stars and comparing them with the sum of radii.

Parameters:

`R`: *array (floats)*; the radii for the star and planets.

`istar`: *float*; the inclination of the system midplane.

`state`: *2-D array (floats)*; compilation of star and planet coordinate lists at that integrator step.

Returns:

`mintrans`: float; a parameter used for integration step size control based on the minimum of the distance of a planet from the stellar disk divided by the planet's angular speed.

`translist`: array (floats); an array of the current status of each planet: `+1` for transit, `-1` for secondary eclipse, `0` for no event.

B.7 Module `coordinates`

`cartesian(GM, kepcoords)`

Converts the orbital elements of a list of planetary orbits to cartesian position and velocity vectors relative to the star. *Note that the coordinate transforms are heliocentric (and thus don't include the star in the vector) rather than barycentric like the N-body integrator. This is necessary for generating the image, which always centers the star.*

Parameters:

> **`GM`**: *array (floats)*; the GM mass parameter for the planets.
>
> **`kepcoords`**: *list of arrays (floats)*; the Keplerian orbital elements for the planets.

Returns:

> **`[x, y, z, vx, vy, vz]`**: *list of arrays (floats)*; the (heliocentric) position and velocity vectors for the planets.

`keplerian(GM, cartcoords)`

Converts cartesian position and velocity vectors relative to the star to instantaneous orbital elements for the planets.

Parameters:

> **`GM`**: *array (floats)*; the GM mass parameter for the planets.
>
> **`cartcoords`**: *list of arrays (floats)*; the (heliocentric) position and velocity vectors for the planets.

Returns:

> **`[a, e, I, longnode, argperi, meananom]`**: *list of arrays (floats)*; the Keplerian orbital elements for the planets.

B.8 Module `wrapImage` / Class `PyImage` / C++ Class `Image`

Note: this is a Cython class. `PyImage` is the Python class, which is a wrapper for the C++ `Image` class. They are implemented in Python with `wrapImage.pyx` and `wrapImage.pxd`. They are implemented in C++ with `Image.cpp` and `Image.h`. The API below describes the usage in normal Python code.

`wrapImage.PyImage()`

Constructor for the `PyImage` class. It holds the logical structures needed to call the C++ routines.

Parameters: None

Returns: `PyImage` object.

`PyImage.SetupImage(rs, Te, rdb, ts, di, rd, drd, Qs)`

Calls the constructor for the C++ Image object. This object holds the star and disk parameters needed to compute the disk image.

Parameters:

`rs`: *float*; the radius of the star.

`Te`: *float*; the effective temperature of the star.

`rdb`: *float*; the dust blowout grain size in microns. Grains smaller than this are assumed to be removed from the system by radiation pressure. Default value is 0.5, set by `defaults.py`.

`ts`: *float*; the sublimation temperature of the dust in kelvins. Grains interior to the radius where this temperature occurs than this are assumed to be removed from the system by sublimation. Default value is 1500, set by `defaults.py`.

`di`: *2-D array (floats)*; the disk parameters for the system.

`rd`: *array (floats)*; the array of dust sizes in the cross-section table.

`drd`: *array (float)*; the width of the grain size bins in the cross-section table.

`Qs`: *2-D array (floats)*; the optical cross-section table for the disk dust composition.

Returns: None

`PyImage.disk_imager(x, y, z, r, dv, cosscattang)`

Computes the disk brightness at a given pixel of the output image by a radiative transfer algorithm. First, the disk is rotated to be aligned with a cartesian coordinate grid. Then, the line of sight to the observer is subdivided in all three dimensions to account for local density variations in the disk. The radiative transfer calculation is performed over each 2-D subpixel, and reported as a spectrum. The spatial resolution of the radiative transfer calculation is determined by the size of the input arrays, and this resolution is iterated upon by `generate_scene.distribute_diskpoints()` to achieve the desired precision, them summed over the pixel. *Note that this assumes the exozodi disk is optically thin.*

Parameters:

`x`, **`y`**, and **`z`**: *3-D arrays (floats)*; the disk-relative x-, y-, and z- coordinates of each sub-voxel in the pixel addressed by the radiative transfer calculation.

`r`: *3-D array (floats)*; the distance from the star in AU of each sub-voxel.

`dv`: *3-D array (floats)*; the volume of each sub-voxel in AU^3.

`cosscatang`: *3-D array (floats)*; the cosine of the scattering angle from the star to the observer for each sub-voxel.

Returns:

`flux`: *3-D array (floats)*; the contrast spectrum of the disk with the star for each 2-D subpixel in the pixel (nx x ny x `nlambda`).

B.9 Module `wrapIntegrator` / Class `PyIntegrator` / C++ Class `Integrator`

Note: this is a Cython class. `PyIntegrator` is the Python class, which is a wrapper for the C++ `Integrator` class. They are implemented in Python with `wrapIntegrator.pyx` and `wrapIntegrator.pxd`. They are implemented in C++ with `Integrator.cpp` and `Integrator.h`. The API below describes the usage in normal Python code.

`wrapIntegrator.PyIntegrator()`

Constructor for the `PyImage` class. It holds the logical structures needed to call the C++ routines.

Parameters: None

Returns: `PyImage` object.

`PyIntegrator.SetupIntegrator(GMin, stin, tin, dtin)`

Calls the constructor for the C++ Integrator object. This object holds the mass, coordinate, and time states that need to be tracked by the integrator.

Parameters:

> **`GMin`**: *array (floats)*; the GM mass parameter for the star and planets.
>
> **`stin`**: *2-D array (floats)*; compilation of the `cartin` coordinate list into an array used by the integrator.
>
> **`tin`**: float; the current time.
>
> **`dtin`**: float; the integrator timestep (smaller than the output timestep)

`PyIntegrator.integrate()`

The core integration function for the N-body integrator, based on the Bulirsch-Stoer integrator written by Henon & Wisdom.

Parameters: None

Returns: None (The integrator state is returned by utility functions.)

`PyIntegrator.getState()`

Utility function returning the planet and star coordinates in a way that can be read by Python.

Parameters: None

Returns:

> **`state`**: *2-D array (floats)*; compilation of the current star and planet coordinates.

`PyIntegrator.getTime()`

Utility function returning the time in a way that can be read by Python.

Parameters: None

Returns:

> **`time`**: *float*; the current time in the integrator, in years.

`PyIntegrator.getDeltaT()`

Utility function returning the timestep size in a way that can be read by Python.

Parameters: None

Returns:

`official_delta_T`: *float*; the current timestep size of the integrator, in years.

`PyIntegrator.setDeltaT()`

Utility function allowing the timestep size to be set from Python.

Parameters:

`newdeltaT`: *float*; the new timestep size of the integrator, in years.

Returns: None

`equations_of_motion(GM, state)`

Computers the "equations" of motion of the star and planets, i.e. the 3-D velocities and accelerations because on Newton's law of gravity.

Parameters:

`local_state`: *array (floats)*; the position and velocity vectors for the star and planets.

Returns:

`local_deriv`: *2-D array (floats)*; the derivative of the state vector, i.e. the velocity and acceleration vectors for the star and planets computed by the function.

B.10 Dataclass `Settings.py`

A Dataclass is a Python class that contains only a structure of data. The `Settings` Dataclass contains all of the code parameters that are intended to be user-settable, as well as their default values. A `Settings` Dataclass with the desired values must be passed to the functions that use any of these parameters. The included parameters are listed below.

Imaging and spectroscopy parameters

`pixscale`: the pixel size in arcseconds. Default value is 0.002.

`iwa`: the inner working angle of the coronagraph, in arcseconds. Default value is 0.015.

`iwa_tol`: the fractional tolerance in precision required for the disk image interior to the inner working angle. Default value is 0.1 (compared with 0.05 or less outside the IWA).
`npix`: size of (square) image in pixels. Default value is 250.
`specres`: resolution of the star and planet output spectra. Default is 300.
`specresdisk`: spectral resolution of the disk contrast cube. Default is 10.
`lambdamin`: minimum wavelength of spectra in microns. Default is 0.3.
`lambdamax`: maximum wavelength of spectra in microns. Default is 1.0.
`hires`: if `True`, resets `specresdisk` to equal `specres`. Default is `False`.

Planet population parameters
`seed`: the (integer) seed of the random number generator. Default is `None`.
`emin` and **`emax`**: minimum and maximum eccentricity, respectively. *Note: eccentricity must be <1.* Defaults are both 0.0.
`imin` and **`imax`**: minimum and maximum orbital inclination relative to the system midplane, respectively, in degrees. Zero inclination may produce unstable behavior in the orbit orientation. Defaults are 0.0 and 5.0, respectively.
`sysimin` and **`sysimax`**: minimum and maximum system inclination relative to the plane of the sky, respectively, in degrees. (I.e. zero is face-on, 90 is edge-on.) Defaults are 0.0 and 180.0.
`sysPAmin` and **`sysPAmax`**: minimum and maximum position angle of the system midplane (rotation relative to the sensor), respectively, in degrees. Defaults are 0.0 and 360.0.
`eecprob`: the probability that a planet in the EEC bounding box will be an EEC. Default is 1.0.
`randphase`: if `True`, adds random variation to the Lambertian phase functions. Default is `False`.
`randrad`: if `True`, adds random variation to the mass-radius relation. Default is `False`.

Disk model parameters
`ncomponents`: number of disk components per star. Default value is 2, but is reset base on the size of the disk array (including zeros).
`diskoff`: if `True`, the disk contrast cube will not be calculated by `generate_scene()` and will be set to zero instead. This is useful to speed up testing. Default value is `False`.
`minsize` and **`maxsize`**: the minimum and maximum sizes of dust grains considered, respectively, in microns. Default values are 0.1 and 1000.0, respectively.
`rdust_blowout`: the minimum dust grain size in microns based on blowout by radiation pressure. Default value is 0.5.
`tsublimate`: the sublimation temperature of dust grains in kelvins. Default value is 1500.

Disk profile parameters
`density_ratio`: the maximum ratio between the densities of disk components. Default value is 5.
`stability_factor`: the minimum separation between a planet and a dust belt, in Hill radii. Default value is 3.
`rinner_mass_threshold`: the minimum mass of planet needed to truncate the disk to its interior, in Earth masses. Default value is 100.

dror_min and **dror_max**: the minimum and maximum fractional widths of dust belts, respectively. Default values are 0.05 and 0.3.
hor_min and **hor_max**: the minimum and maximum fractional scale height of the disk, respectively. Default values are 0.03 and 0.2.
r_min and **r_max**: lists of the minimum and maximum circumstellar distances, respectively of the three disk components, in AU. Defaults values are [0.5,5.,50.] and [5.,50.,500.].

Scene parameters
timemax: the end time used by the N-body integrator in years. Default value is 1.e-10 (single image only). Typical values for a survey are 5-10.
dt: timestep for the nbody integrator in (Julian) years. Default value is 10/365.25 (10 days).
output_dir: output directory. Default value is "output".

One parameter is computed by the `__post_init__()` function that runs automatically upon initialization:

pixscale_mas: the pixel size in milliarcseconds, based on pixscale.

specrdisk is also reset by `hires` in this function, if applicable.

B.11 Module `constants.py`

This module does not contain any functions, but does contain a list of constant values that are needed by various functions in the code. These constants are listed below.

Physical constants
grav_const: the gravitational constant in units of AU^3 yr^{-2} Msun^{-1}.
c: the speed of light in AU yr^{-1}.
planck: Planck's constant in cgs units.

Data structure parameters
maxnplanets: number of planets per star. Default value is 30, but is reset based on the size of the planets array (including zeros).
mincomponents: minimum number of disk components. Default value is 1.
maxcomponents: maximum number of disk components. Default value is 3.

File paths
exovistapath: path to the main ExoVista directory, where the relevant data values should be. Default value is the current directory.
lqq_dir: directory where the dust cross-section tables are stored. Default value is "lqq_files/".

Dust model parameters. These are used to define the dust cross-section filenames procedurally from a minimum and maximum grain size, and a resolution in log-grain size. However, this method consistently failed to produce the correct filenames due to rounding errors, and the commented-out variables are not used.

The arrays **`master_rdust`** and **`master_rdust_boundaries`** are hard-coded with the specific decimals used in the cross-section filenames. These should not be changed unless the cross-section tables do, or else the code will crash.
`master_nsizes`: the length of the `master_rdust_boundaries` array.
`master_drdust`: widths of the grain size bins, generated automatically from the `master_rdust_boundaries` array.

Table headings. These describe the valid column names for star, planet, and disk properties. In many cases, they are used for indexing, and they are used outside of `generate_scene` where the `**kwargs` reassignment does not apply, so they should not be changed, or else the code will crash or exhibit unstable behavior.

`starbase`: list of valid columns for stellar properties (excluding `nzodis`, since `nzodis` may need to be randomized).
`intlist`: stellar properties that have int rather than float data type.
`strlist`: stellar properties that have string rather than float data type.
`keplist`: list of orbital elements of stars and planets.
`pllabel`: list of planet properties.
`dlabel`: list of disk properties.

FITS file comments. These will not need to change unless you substantially change the format of the output files.
`scomments`: comments in the heading of the stellar data extension (Extension 4).
`pcomments`: comments in the headings of the planetary data extensions (Extension 5+).
`dcomments`: comments in the headings of the disk data extension (Extension 2).

B.12 Module load_scene.py

`load_scene.load_scene(inputfile, time=0)`

Reads in a FITS file produced by ExoVista and outputs a tuple of numpy arrays containing the important data for image simulations.

Parameters:

> **`inputfile`**: *string*; FITS file to be read.
>
> **`time`**: *float*; time at which the system parameters should be interpolated and output, in years. Default value is `0`.

Returns: an 8-element tuple containing the following:

- `lambdas`: *array (floats)*; the wavelength points at which the spectra are calculated, in microns.
- `xystar`: *array (floats)*; 2-element array containing the coordinates of the star, in pixels.
- `fstar`: *array (floats)*; the spectrum of the star, in Janskys.
- `xyplanet`: *2-D array (floats)*; the coordinates of the planets, in pixels.
- `fstar`: *2-D array (floats)*; the spectra of the planets, in Janskys.
- `diskimage`: *3-D array (floats)*; the disk contrast data cube of brightness relative to the star, except interpolated to the same wavelengths as the stellar spectrum.
- `angdiam`: *floats*; the angular diameter of the star, in milli-arcseconds.
- `pixscale`: *floats*; the pixel scale of the image, in milli-arcseconds.

B.13 Module add_background.py

`add_background.add_background(inputfile, time=0)`

Reads in a FITS file produced by ExoVista and generates a data cube of extragalactic background sources based on the Haystacks model of Roberg et al. (2017).

Note: this module requires Extragalactic background cubes 0 through 5 from https://asd.gsfc.nasa.gov/projects/haystacks/downloads.html

Parameters:

- `inputfile`: *string*; FITS file to be read.

- `time`: *float*; time at which the system parameters should be interpolated and output, in years. Default value is `0`.

Returns:

- `Background_final`: *3-D array (floats)*; data cube of extragalactic background fluxes in janskys per pixel, with the same format as the disk data cube.