Software Requirements Document for

EChOSim: A Simulator for the Exoplanet Characterization Observatory

ECHO-TN-002-CF-SRD
Version 2.0
November 26, 2013

Contributors:
Andreas Papageorgiou, Ingo Waldmann, Enzo Pascale, Carrie MacTavish, Alexander Amaral-Rogers, Jean-Philippe Beaulieu, Céline Cavarroc, Vincent Coudé du Foresto, Marc Ollivier , Locke Spencer, Bruce Swinyard, Marcel Tessenyi and Giovanna Tinetti

Table of Contents

1 Introduction

1.1 Purpose of Document

This document describes the software requirements and architecture of the simulation tool `EChOSim`. In the Introduction, the project scope, as well as useful references and acronyms are given. Section 2 provides an overview of the organisation of the `EChOSim` directory structure and briefly outlines each of the modules, classes and internal libraries. Finally, Section 3 provides a detailed description of the `EChOSim` architecture, including details of the algorithms involved for each module.

1.2 Project Scope and Objectives

The purpose of this project is to develop a simulation tool for the Exoplanet Characterization Observatory. The software, `EChOSim`, will simulate the astrophysical scene (star and transiting planet) as well as the stationary and dynamic characteristics of the instrument and observing strategy. The objective of the `EChOSim` software is to evaluate critical design elements of the EChO instrument, as well as, evaluate impact of variations in the system level design and implementation, setting benchmarks for the instrument parameters. `EChOSim` is intended to provide the "ultimate" performance prediction. Users of the tool are EChO collaborators.

1.3 Acronyms

SRD Software Requirements Document (this document)

URD User Requirements Document

TBD To Be Determined

FOM Figure of Merit

1.4 Useful References

- `EChOSim`: More about user requirements for software design can be found in the `EChOSim` User Requirements Document (URD).
- EChO Instrument and Telescope: More information about the EChO mission can be found in the documents *EChO Science Requirements Document*, *EChO Payload Definition Document* and *EChO Mission Proposal* available on

  http://echo-spacemission.eu/

2 `EChOSim` Code Organisational Overview

The main is called `echosim.py`. It initialises the parameter object and initialises and runs each module in turn. The program is organised according to the following structure. The main, `echosim.py` and the parameter file, `echosim.par`, are in the root directory. Then four subdirectories exist:

a data directory (with further self-explanatory subdirectories)

- `classes`: containing the object classes called on by the modules
- `data`: containing planetary and stellar data files

- `docs`: containing documentation files and images
- `library`: containing functions shared by several classes and modules.
- `modules`: containing the higher-level code, calling classes and library functions

2.1 Data

The data subdirectory stores all the useful data of the system (for instance quantum efficiency, spectra, ...). Currently there is:

`data/instrument`: containing dichroic emission and transmission files; quantum efficiency tables.

`data/planetary`: containing planetary SED files for primary and secondary eclipses

`data/telescope`: containing mirror emissivity and reflectivity files; solid angle tables

`data/stellar`: containing stellar SED files from `SP_Phoenix.zip` provided by Ignasi Ribas; stellar limb-darkening parameter by Claret (2000).

2.2 Classes

`classes/channel.py` This placeholder class is in the process of being removed as it's no longer required.

`classes/detector.py` Object to hold detector-specific parameters and provide methods for the calculation of detector relevant functions such as focal plane ilumination and PSF.

`classes/instrument.py` Object to hold instrument-specific specific parameters such as dichroic emissivity/reflectivity and provide convenience methods for calculatting the instruments emission.

`classes/internal.py`: Object to hold the simulation's main variables (spectra) and pass them from one module to the next.

`classes/noise.py`: Object to hold noise specific data and provide convenience methods for applying noise to the simuation's timelines.

`classes/parameters.py`: Object to hold the all the parameters defined in the parameter file

`classes/planet.py`: Object to hold planet-specific parameters and provide methods for calculation of planetary emission/transmission spectra, solve orbital motions and generate light curves.

`classes/star.py`: Object to hold star-specific parameters and read-in star spectra

`classes/telescope.py` Object to hold telescope-specific parameters and provide methods for the simulation of the thermal environment and to apply telescope related effects to spectra from the Astroscene and Foreground modules.

`classes/position.py` Object to hold right-ascention (RA), declination (Dec), and observing date of the observed planet-star system. This information is needed for a time and position dependent Zodi emission calculation.

`classes/progress.py` Object tracking and displaying calculation progress, providing estimated completion time of overall simulation.

2.3 Modules

All of the `EChOSim` modules are in the Modules directory (Note: the observation pipeline is supplied with the `EChOSim`code but not formally part thereof). Details of the functionality and I/O for each module are given in the Section 3. Currently defined modules include:

`modules/Astroscene_module.py` – Initialises star class, planet class and calculates light curves. Returns light-curves.

`modules/Foreground_module.py` – Calls zodi method and fills internal data with retrieved values.

`modules/Instrument_module.py` – Initialises the telescope and instrument classes. Applies instrumental effects to incoming light from Astroscene and Foreground modules. Provides basic detector functionality (SNR calculations)

`modules/Noise_module.py` – Module that calculates all different contributions of noise (pointing, temperature, gains) to the data timelines.

`modules/Output_module.py` – Module that collates all focal plane illuminations (per time stamp) and generating fits-standard image files. These fits files follow the standard astronomical format which can be read in and analysed by the observation pipeline of generic astronomical packages.

2.4 Library

This directory contains all the useful functions that are neither modules nor methods (for instance, functions which require different classes or are used by several classes). Currently defined libraries include:

`library/library_occultquad.py`: computes the light-curve for occultation of a quadratically limb-darkened source without microlensing. Based on Mandel & Agol (2002) and Eastman & Agol (2008).

`library/library_astroscene.py`: includes the eccentric orbital motion generator; a black-body function generator; and smaller multiply used functions for interpolation, array sorting, etc..

`library/library_foreground.py`: simple JWST-MIRI paramaterisation of the Zodiacal light.

`library/library_output.py`: supporting functions (array sorting, dictionary collating, etc.)for the Output_Module.

3 `EChOSim` Architectural Design

Figure 1 shows an overview of the entire `EChOSim` architecture. This tool will consist of several modules. The main program, `echosim.py`, initially calls `Astroscence_module.py` for a given input source, simulating the observed sky. Next each of the modules describing the EChO instrument is called, following the input radiation along it's journey to the final detector destination. Thus a simulated EChO observation is generated for a single source and the raw focal plane illumination data is provided as output. This can be read in by the external Observation Pipeline or other standard astronomical data reduction software packages. In this section the algorithmic details of each of the modules are given.

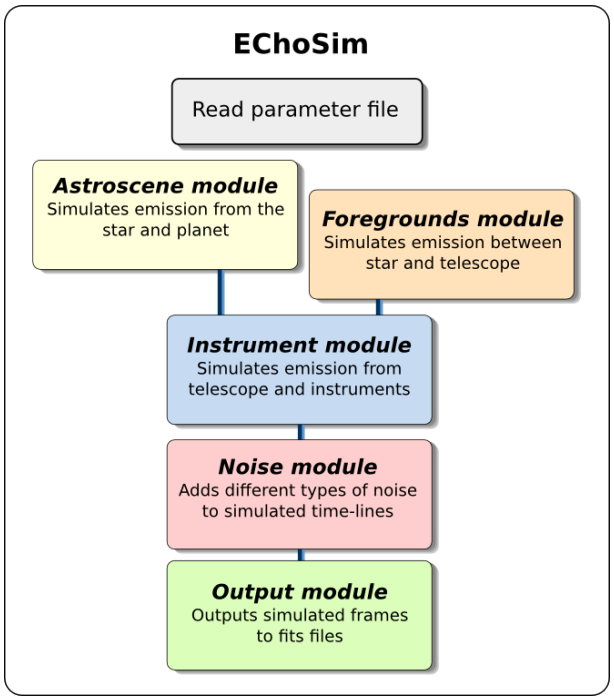

Figure 1: Overview of `EChOSim` Architecture. The main calls a number of modules describing the sky and the payload instrument parameters.

3.1 Astroscene Module

3.1.1 Module Aim

This module simulates the 'astrophysical scene' and generates the data to be 'observed' by `EChOSim`. Starting from an input of stellar SEDs and pre-specified planet, star and orbital parameters. Astroscene will generate a series of stellar spectra modulated over time with: 1) the planetary light curve, including the 2) planetary transmission/emission spectrum, 3) the planetary phase curve, 4) time correlated stellar noise and 5) exozodiacal effects. The calculations of the observables are summarised below with block diagram illustrating the architecture in figure 2.

1. Given the set stellar temperature, the appropriate SED file is read and scaled to a user specified apparent magnitude.
2. The average surface temperate of the planet is calculated (given user provided orbital parameters) and the appropriate planetary SED for primary or secondary eclipse is select or approximated by a black-body distribution.
3. Given user supplied parameters, the planetary orbit is simulated and the wavelength dependent planetary SED used to set appropriate transit depths of either the primary or secondary eclipse.
4. The stellar SED and the generated planetary lightcurves are combined to give a time-resolved observation of the planet-star system.

For more information on the algorithm, see below and on the underlying theory see URD [v2.0] Section 3.1.

3.1.2 Inputs

Input files:

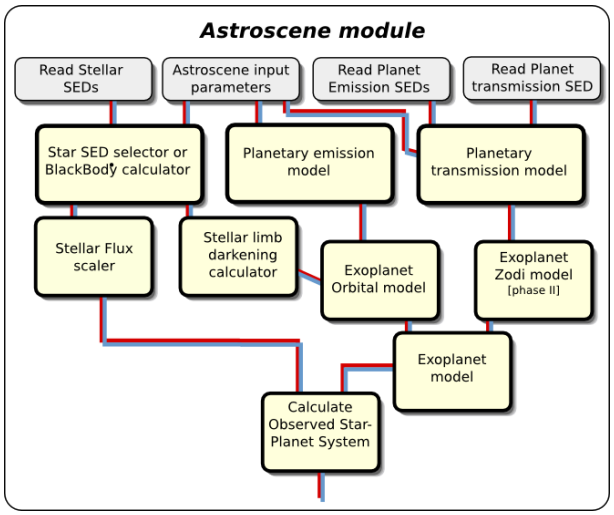

Figure 2: Block diagram of `Astroscene_module.py`. Simulates the astrophysical scene and includes a description of the star and transiting planet. All time-dependent effects (limb darkening, transit profile, stellar variability) are included. The output are stellar spectra modulated over time by the planetary primary/secondary eclipse.

Name	Type	Description
`params`	Parameters.class	This parameter object is initialized by the main script and contains simulation relevant parameters. The parameters contained in this object are read in from the '`echosim.par`' file.
`/data/stellar`	Folder	Containing stellar LTE spectra from 3070K - 7200K and solar gravity and metalicities. Spectra are generated using the Phoenix code and provided by Ignasi Ribas.
`SPTyp_KH.dat`	File	Stellar spectra index file used by `EChOSim`.
`quadratic.dat`	File	Containing limb-darkening coefficients for varying temperatures, metalicities and log$g$ from Claret (2000).
`/data/planet`	Folder	Folder containing planetary transmission and emission spectra for various planet types

Inputs in Parameter class:

Name	Units	Description
star_dist	pc	Observer-star distance
star_radius	$R_\odot$	Stellar radius
star_temp	K	Stellar effective temperature
star_logg	m/s^2	Stellar surface gravity
star_mh	Fe/H	Stellar metallicity
spectral_res	R	Spectral resolution (Note: legacy, now defined by detector module)
oversamplefact	double	Spectral oversampling (Note: legacy, now calculated internally)
star_ra	deg.	Stellar right ascension
star_dec	deg.	Stellar declination
star_date	JD	Julian date of observation
planet_radius	$R_{Jupiter}$	Radius of planet
planet_mass	$M_{Jupiter}$	Mass of planet
planet_albedo	NA	Exoplanetary bond-albedo
planet_period	Days	Orbital period
planet_A	AU	Orbital semi-major axis
planet_inc	degrees	Orbital inclination (0-90 deg.)
planet_ecc	NA	Orbital eccentricity (0-1)
planet_omega	degrees	Argument of periastron for an eccentric orbit
planet_nu	kg	Molecular weight of the exoplanetary atmosphere
planet_blackbody	string	Set to T: using blackbody emission, F: use provided emission spectra
planet_prisec	string	Set to 'pri' for primary eclipse and 'sec' for secondary eclipse
planet_specfile	string	Allows the read-in of a transmission/emission spectrum as atmospheric spectrum
planet_specconst	double	Conversion factor (if needed) to convert spectrum file into F_p/F_s or $(R_p/R_s)^2$
planet_sampling	seconds	Exposure time (including all overheads)
planet_obsrate	integer	Fraction of time observed with respect to total transit duration (typically: > 2)

3.1.3 Outputs

Outputs are written into the `intdata` object. *Ftot* is also returned into *main* as standard output.

Name	Type	Description
astro_Ftot	2d array	'Observed' spectra, rows: wavelength axis, columns: time axis
astro_Fstar	1d array	Single stellar spectrum
astro_wavelengthgrid	1d array (μm)	Wavelength values for `astro_Ftot` and `astro_Fstar`
astro_lightcurve	1d array	Normalised light curve model
astro_phasegrid	1d array	Orbital phase values for `astro_lightcurve`
astro_timegrid	1d array (sec)	Time stamps for observations, first spectrum at $t = 0$
astro_totalobstime	seconds	Total observation time
astro_tempplanet	K	Average planetary surface temperature
astro_tempstar	K	Stellar surface temperature, may differ from parameter file

3.1.4 Algorithm

1. Both `star.class` and `planet.class` check the input parameters provided in `params` for consistency and either issue an error report and de-bugging information for fatal errors or issues a warning and set the faulty parameters to their lower or upper bounds.
2. The `star.class` is now initialised and input parameters are converted into SI units.
3. Given the stellar temperature in `params`, `star.class` will select the best matching stellar SED from the data archive. If stellar temperatures are outside the range of 3070 - 7200K, it will issue a warning and approximate the SED with a black-body curve of the given temperature.
4. The stellar SED is now interpolated to the spectral resolution set in `params` and trimmed to the spectral set in `params`. Should the spectral range set exceed the stellar SED spectral range, `star.class` will zero-pad the spectral edges.
5. The stellar SED is scaled to the user defined apparent magnitude and together with the corresponding wavelength grid returned by `star.class`.
6. When primary eclipse is specified in `params`, `star.class` will load quadratic limb-darkening coefficients for the specified stellar type and linearly interpolate between pass-band coefficients for all `EChOSim` wavelengths.
7. The `planet.class` is now initialised and input parameters are converted into SI units.
8. Given inputs set in `params`, `planet.class` estimates the total transit duration and calculates the time and orbital phase grids of the observation as well as transit mid-points.
9. The average planetary surface temperature is calculated and given the user input either steps 10 - 12 or steps 13 - 14 are executed.
10. *Primary eclipse case:* The planetary surface gravity is estimated and the atmospheric scale height is calculated.
11. Given the planetary temperature, `planet.class` reads in the appropriate transmission model and scales the atmospheric amplitude to 5× scale height. If specified in the parameter file, a pre-calculated transmission spectrum can also be read in.

12. The total apparent primary eclipse depth is now calculated as function of wavelength.
13. *Secondary eclipse case:* The appropriate planetary day-side emission model is read in (given the planetary temperature) and scaled by the star-planet surface ratios. If no emission models are found for the given planetary temperature, `planet.class` will approximate the day-side emission using a black-body distribution. If specified in the parameter file, a pre-calculated emission spectrum can also be read in.
14. The total apparent secondary eclipse depth is calculated as function of wavelength.
15. The `planet.class` now solves the Keplarian orbit of the exoplanet for user specified orbital parameters and uses outputs from steps 6, 12 and 14 to set the wavelength dependent eclipse depths.
16. The outputs of `star.class` and `planet.class` are now combined to form the 'observed' array with flux, time and wavelength as their axes.

3.1.5 Class methods:

Star class
Location: classes/star.py
Description: Class calculating stellar SEDs and stellar limb-darkening parameters

Methods:

- `paramcheck()`: Checks all class relevant input parameters contained in `params` for internal consistency and either issue an error report and de-bugging information for fatal errors or issues a warning and set the faulty parameters to their lower or upper bounds.
- `wave_grid()`: Calculates the correct wavelength grid for the observations given Detector_Module parameter settings.
- `select_star()`: Finds closest match between user specified stellar temperature and available stellar SED in 'data/stellar'. If stellar temperature in `params` exceeds limits of available stellar SEDs, it will issue a warning and set the default to a black-body curve instead.
- `do_spectrum()`: Loads the selected stellar SED from 'data/stellar' and interpolates it onto a given resolution grid. Alternatively it calculates the black-body function for given stellar temperature for the same resolution grid.
- `get_limbdarkening()`: When primary transit is selected in `params`, stellar limb darkening coefficients corresponding to the host-star are loaded and interpolated over the `EChOSim` wavelength grid.
- `mag_scaler()`: Scales the stellar SED to the user specified stellar distance.

Planet class
Location: classes/planet.py
Description: Class calculating planetary SEDs, the Keplarian orbits and primary/secondary eclipses

Methods:

- `paramcheck()`: Checks all class relevant input parameters contained in `params` for internal consistency and either issue an error report and de-bugging information for fatal errors or issues a warning and set the faulty parameters to their lower or upper bounds.
- `orbital_model()`: Calculates the total transit time and average planetary temperature; sets up the time and orbital phase arrays for the observation sequence; solves Kepler's equations and calculates the eclipse model.
- `exoplanet_model_pri()`: Calculates the planetary surface gravity; the atmospheric scale height; loads the appropriate stellar transmission spectrum for the given planetary temperature, interpolates the spectrum to the correct spectral resolution and scales it using the atmospheric scale height.
- `exoplanet_model_sec()`: Loads the appropriate planetary emission spectrum and interpolates to the correct spectral resolution. If no appropriate file is found, the planetary spectrum is approximated using a black-body distribution.

3.1.6 Libraries:

library_occultquad
Location: library/library_occultquad.py
Description: Library containing Mandel & Agol (2002) light curve modelling code

Methods:

- `ellke(k)`: Computes Hasting's polynomial approximation for the complete elliptic integral of the first (ek) and second (kk) kind.
- `ellpic_bulirsch(n,k)`: Computes the complete elliptical integral of the third kind using the algorithm of Bulirsch (1965).
- `occultquad(z,u1,u2,p0)`: Computes the light curve for occultation of a quadratically limb-darkened source without microlensing, Mandel & Agol (2002) and Eastman & Agol (2008).
- `occultquad2(z,p0, gamma)`: Quadratic limb-darkening light curve; cf. Section 4 of Mandel & Agol (2002).
- `occultuniform(z, p)`: Uniform-disk transit light curve (i.e., no limb darkening).
- `depthchisq(z, planet, data)`: Computes model chi-squared.

library_astroscene
Location: library/library_astroscene.py
Description: Library containing general functions and Keplarian orbit model

Methods:

- `tscompress(data,status)`: compresses the timelines using a jpeg-like compression.
- `find_nearest(arr,value)`: find nearest value in array.
- `sortarray(x,y)`: function sorting array to ascending values of x.
- `drange(start, end=None, inc=None)`: range function that does accept float arguments.

- `interpolator(xold,yold,xstart,xend,diff)`: interpolates values to given grid.
- `eccentric(phase,inc,aR,ecc,omega)`: function calculating the star planet separation,z, for the case of eccentric and circular orbits.
- `planck( wave, temp )`: Calculates a black-body function for a given wavelength and temperature.

3.1.7 Functional requirements:

Req. ID	Description
ASTROSCN-FUNC-010	The module will require a `parameter file` as input
ASTROSCN-FUNC-020	The module will require stellar SEDs as input
ASTROSCN-FUNC-021	The module will require a stellar SED index file as input
ASTROSCN-FUNC-022	The module will require a stellar quadratic limb darkening library as input
ASTROSCN-FUNC-030	The module will require planetary emission SEDs as input
ASTROSCN-FUNC-031	The module will require planetary transmission SEDs as input
ASTROSCN-FUNC-032	The module will require planetary SED index file as input
ASTROSCN-FUNC-040	The module will convert all parameters into SI units
ASTROSCN-FUNC-050	The module will read in the appropriate stellar SED given the user input
ASTROSCN-FUNC-051	The module will approximate the stellar SED using a black-body function if required
ASTROSCN-FUNC-060	The module will interpolate and trim the stellar SED to user defined resolution
ASTROSCN-FUNC-061	The module will compute a common wavelength grid for the given resolution
ASTROSCN-FUNC-070	The module will scale the stellar SED to user specified apparent magnitude
ASTROSCN-FUNC-080	The module will read in stellar limb-darkening parameters from Claret (2000)
ASTROSCN-FUNC-081	The module will select correct limb-darkening parameters given user specified stellar temperature and solar gravity and metalicity
ASTROSCN-FUNC-082	The module will interpolate limb-darkening parameters for the common wavelength grid
ASTROSCN-FUNC-090	The module will read in the appropriate planetary SED given the user input
ASTROSCN-FUNC-091	The module will approximate the planetary emission SED if required
ASTROSCN-FUNC-100	The module will read in the appropriate planetary SED given the user input
ASTROSCN-FUNC-110	The module will compute the planetary temperature
ASTROSCN-FUNC-111	The module will compute the planetary atmospheric scale height
ASTROSCN-FUNC-120	The module will compute the total transit time
ASTROSCN-FUNC-121	The module will solve the Keplarian orbit
ASTROSCN-FUNC-122	The module will compute observed primary and secondary eclipse light curves
ASTROSCN-FUNC-123	The module will compute one light curve model per spectral resolution point
ASTROSCN-FUNC-130	The module will read in time-series of correlated stellar noise taken from Kepler or CoRoT space mission archives
ASTROSCN-FUNC-131	The module will interpolate stellar noise time-series onto the common orbital phase grid
ASTROSCN-FUNC-132	The module will scale the stellar noise using a wavelength dependent function to be specified
ASTROSCN-FUNC-140	The module will modulate the stellar SED with the wavelength dependent planetary light curves to obtain a time-series of modulated stellar SEDs

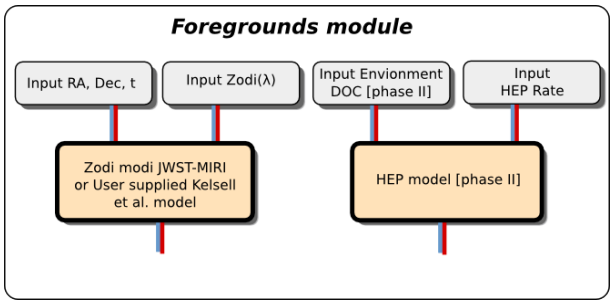

Figure 3: Block diagram of `Foregrounds_module.py`. The foregrounds module includes any static or dynamic signal sources which act as a source of noise between the exoplanet signal and the EChO telescope optics. Sources of such noise include high energy particles and/or zodiacal light.

3.1.8 Open Issues and Improvements:

1. ASTROSCN-FUNC-13* functions are not currently implemented and belong to a later development stage.

3.2 Foregrounds Module

3.2.1 Module Aim

Figure 3 shows a block diagram for this module. The aim of this module is simulate the foreground sources which act as a source of noise between the exoplanet signal and the EChO telescope optics. For more information on the underlying theory see URD [v2.0] Section 3.2.

3.2.2 Inputs

Name	Type	Description
`params`	Parameters.class	This parameter object is initialised by the main script and contains simulation relevant parameters. The parameters contained in this object are read in from the `'echosim.par'` file.
`intdata`	Internaldata.class	This is a common object containing the output data from all the modules. The input wavelength array grid from astroscene is loaded from this object. Zodi values are generated for each wavelength in the input grid.

Inputs in Parameter class:

Name	Units	Description
`star_ra`	degrees	RA of star
`star_dec`	degrees	DEC of star
`star_date`	days	Julian date
`run_foreground`	NA	Turns foreground module on(1)/off(0)
`zodi_level`	NA	Indicates zodi level; 1=min, 2=average, 3=max (default)

3.2.3 Outputs

Outputs are written into the `intdata` object.

Name	Type	Description
zodi	1d array	Zodiacal light value in $W/m^2/micron/sr$ for each astroscene defined wavelength grid point.

3.2.4 Algorithm

1. If the input parameter `run_foreground` is set (1) then `Foreground_module.py` is run. Otherwise there is no foreground inclusion.
2. The `params` and `intdata` objects are passed to the `Foreground_module.py` which in turn passes the source position (ra/dec) and date (from params) and the wavelength array (from intdata) along to the `zodi` method (part of the `library_foreground.py`). Note position and date info are not yet used.
3. The `zodi` method calculates the zodi light foreground contribution for each wavelength in the `astro_wavelengthgrid`. The model is based on a simple (position independent) JWST-MIRI model.
4. The zodi value is scaled by a predetermined factor (min, average, max) which is set in the parameter file (`zodi_level`).
5. The final output zodi array is returned in units of $W/m^2/micron/sr$ and written to the `indata.zodi` object to be passed onto the `Instrument_module` at a later point.

3.2.5 Libraries:

library_foreground
Location: library/library_foreground.py
Description: Library containing simple JWST-MIRI paramaterisation of the Zodiacal light.

Methods:

- `zodi(star_ra, star_dec, star_date, wave, zodi_level)`: Computes flux of zodiacal light foreground in $W/m^2/micron/sr$. Currently based on simple JWST-MIRI parameterization (position info not yet used).
- `foreground_file(wave, filename)`: Load pre-specified foreground emission files and interpolate the emission on a given wave grid.

3.2.6 Functional requirements

Req. ID	Description
~~FOREGND-FUNC-010~~	(Obsolete) ~~The module will require a parameter file as input~~
~~FOREGND-FUNC-012~~	(Obsolete) ~~The module will require astroscene wavelength array as input~~
FOREGND-FUNC-013	The module will require a parameter and an internal object as input
FOREGND-FUNC-020	The module will determine source position on the sky
FOREGND-FUNC-021	The module will determine zodi flux [at current source position] for each wavelength in ascroscene grid
FOREGND-FUNC-030	The module will require input parameters describing the high energy particle impact on pixels; including pixhit and *nonoppix*
FOREGND-FUNC-031	The module will require input parameters describing the detector specific timings associated with readout and integration; including t_{ramp} and t_{read}
FOREGND-FUNC-040	The module will calculate t_{eff}, the effective integration time, taking high energy particle hits into account.

3.2.7 Open Issues and Improvements:

1. A later version will include a more sophisticated zodi model which takes into account source position on the sky. This will involve implementation of FOREGND-FUNC-020 and an upgrade to FOREGND-FUNC-021.

2. FOREGND-FUNC-030/031/040 functions are not currently implemented and belong to a later development stage.

3.3 Instrument Module.

3.3.1 Module Aim

Figure 4 shows a block diagram for this module. This module is responsible for simulating the behaviour of the instruments with its main function to call the output from the Astroscene, pass through the telescope class, apply relevant optical-chain effects to the spectra, convolve with the instrument line function, re-bin to the passband spectral resolution wavelength grid and integrate across each finite element. For more information on the underlying theory see URD [v2.0] Section 3.4.

The functionalities of the module can be summarised in three categories:

1. **Call telescope.class:** The telescope class calls data from the astroscene and the foreground modules stored in the intdata common object to calculate the telescope relevant effects and apply to the signal output. This class interpolates all relevant pre-calculated data to the astroscene wavelength grid, calculates the flux contribution from the scattered light of the baffles, zodiacal light, mirrors (assuming modified blackbody profiles) and returns Ptot and Psig.

2. **Call instrument.class:** This class (section 3.3.5) contains methods to call all relevant pre-calculated data contained in parameter files located in the '/data/instrument/' directory such as dichroic transmissions and emission profiles, and interpolates to the astroscene wavelength grid. The class contains methods to generate black body profiles when called in the body of the instrument module code, generate Instrument Line Functions (ILF) per channel and convolve with the spectra, a method to re-bins to spectral resolution grids, integrates across each finite element of the passband.

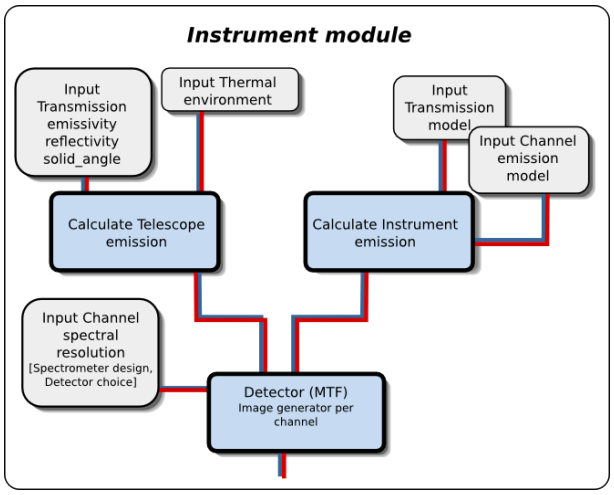

Figure 4: Block diagram of Instrument Module. This includes thermal emission from telescope as well as all of the EChO apparatus.

3. **Calculate the power passing through each channel:** This is the main functionality of the module where the dichroic transmission acts to split the input signals into channels, the emission from the dichroics are applied, the signal is convolved with the ILF and re-binned according to the channel spectral resolution. The 2D data arrays containing the Pchannel_tot_det and Pchannel_sig_det are stored in the intdata object along with the relevant output wavelength grid.

4. **Detector function:** Temporary function acting as the detector module contained in the instrument.class. Takes the output from the instrument module and converts Pchannel_tot_det, Pchannel_sig_det (W) to Ichannel_tot, Ichannel_sig (electrons / second / pixel) and calculates the SNR.

3.3.2 Module Inputs

Name	Type	Description
params	Parameters.class	This parameter object is initialised by the main script and contains simulation relevant parameters. The parameters contained in this object are read in from the 'echosim.par' file.
intdata	Internaldata.class	This is a common object containing the output data from all the modules. The input wavelength array grid from astroscene is loaded from this object. Zodi values are generated for each wavelength in the input grid.

3.3.3 Module Outputs

The output of this module is the updated `internal` object with all of the Q_tot data now containing contributions from telescope and instrument emission.

3.3.4 Algorithm

1. Calls holding temperatures of the mirrors from the parameters file and creates temperature profiles for the mirrors (assumes static temperature).

2. Loads mirror reflectivity, emissivity and solid angle data files stored in the /data/telescope file path. Generates transmission profile and interpolates parameters to the astroscene wavelength grid. Interpolator is zero padded such that where the spectral range is outside the wavelength range of the external files, returns zero values.

3. Calculates the emitted power assuming a blackbody profile modified by the emissivity of the mirrors for MNUM of mirrors in the telescope optics for all observation times, NTimeslices. Returns Ptel (W micron^{-1})

4. Loads Fzodi (W m^{-2} sr^{-1} micron^{-1}) from the foregrounds module if it has been selected and converts to power in the telescope beam by multiplying by the etendue. If the Foreground module has been turned off then this will return zero values. Returns Pzodi (W micron^{-1})

5. Loads pre-calculated scattered light flux, Fscat and interpolates with zero padding to the astroscene wavelength grid. Where the spectral range is outside the wavelength range of the external files, returns zero values. Converts the flux to power in the telescope beam, Pscat by applying the etendue of the system.

6. Combines Fsig (W m^{-2} sr^{-1} micron^{-1}), the output of the astroscene module with the transmission, tau and effective gathering area, A_eff to derive Psig: the power of the signal observed (W micron^{-1}). Psig = Fsig * A_eff * tau

7. Combines Fsig with Ptel, Pzodi, Pscat, and tau to calculate the total power transmitted through the telescope, Ptot (W micron^{-1}) for all observation times calculated in the astroscene module.
   Ptot = ((Fsig + Pzodi) * tau + Pscat) * A_eff

8. Stores interpolated parameters, system etendue, Ptel, Pscat, Pzodi, Ptot and Psig, data to intdata common object to be saved later.

9. Loads external dichroic transmission, emission data profiles stored in the /data/instrument file path and interpolates with zero padding to astroscene wavelength grid. Where the spectral range is outside the wavelength range of the external files, returns zero values.

10. Calls holding temperatures of the dichroics from the parameters file and creates temperature profiles for the mirrors (assumes static temperature for all observation times).

11. Creates 3D holding data cubes to store the spectra of each observation passing through each channel. Loops over the number of instruments, InstNum and applies the relevant transmission profile. Adds the component of emission from the dichroics assuming modified blackbody function to the power passed through the system:

    - Pchannel_1 - Ptot * (1-tau1) + eta1 * B1 * A_eff * Solid_angle
    - Pchannel_2 - (Ptot * tau1 + eta1 * B1 * A_eff * Solid_angle) * (1 - tau2) + eta2 *B2 * A_eff * Solid_angle
    - Pchannel_3 - [(Ptot * tau1 + eta1 * B1 * A_eff * Solid_angle) * tau2 + eta2 *B2 * A_eff * Solid_angle)] * (1 - tau3) + eta3 * B3 * A_eff * Solid_angle
    - Pchannel_4 - [(Ptot * tau1 + eta1 * B1 * A_eff * Solid_angle) * tau2 + eta2 *B2 * A_eff * Solid_angle)] * tau3 + eta3 * B3 * A_eff * Solid_angle

12. Generates an Instrument Line Function (ILF) per channel assuming a top hat function with width defined by the smallest lambda* R of the passband. Area of the top hat is normalised to 1.0

13. Convolves the ILF with Pchannel_tot and Pchannel_sig using convolve() function. Convolution acts to smooth the spectra.

14. Calculates a wavelength grid based on the spectral resolution of the channels starting from the lower wavelength edge of the passband and interpolates the convolved spectra to this new grid.

15. Integrates the convolved spectra over each finite element of the spectral bins to return Pchannel_tot_det, Pchannel_sig_det (W). Takes the centre of each finite bin as the final wavelength grid. Writes Pchannel_tot_det, Pchannel_sig_det and the wavelength grid to the intdata object.

16. Loads external detector quantum efficiency data from data/instruments/ file path and interpolates with zero padding the quantum efficiency to the wavelength grid produced during the integration of the finite element. Loads pixel numbers illuminated by the psf and the integration time.

17. Converts Pchannel_tot_det, Pchannel_sig_det (W) to Ichannel_tot_det, Ichannel_sig_det (electrons/second/pixel) by multiplying by h * c / lambda and then multiplying by the interpolated quantum efficiency / number of illuminated pixels.

18. Calculated signal to noise ratio assuming photon noise. Where the signal to noise ratio is given by:
Ichannel_sig_det * integration time / SQRT(Ichannel_tot_det * integration time).

19. Writes the data to the intdata object to be saved further on in the code.

3.3.5 Classes, Methods, Libs Used

telescope.class: The telescope class is responsible for simulating the behaviour of the telescope and its main aim is to apply telescope-relevant effects to spectra produced by the Astroscene and Foreground Modules and combine them.
The functionality of the class can be summarised in four categories:

1. **Read telescope parameter files:** Most of the information describing the telescope is contained in parameter files located in the `/data/telescope/` directory. This information is often a function of wavelength and it has to be interpolated into the simulations wavelength grid. The class provides methods for reading the required data and interpolating it to the required wavelength grid.

2. **Simulate telescope thermal environment:** A factor that affects the telescope contribution to the observed data is the temperature of the telescope at a given time. The class provides a method(`get_Temperature`) that simulates the thermal environment. Currently the output temperature is a combination of constant with added normal random noise (to be updated to Brownian noise). Plans for the temperature time-line to incorporate temperature arising from the observation geometry (ra, dec, solar aspect angle) are TBI.

3. **Generate telescope emission:** The method(`calc_Ptel`)take the calculated thermal environment and calculates the emitted and transmitted telescope power assuming a modified blackbody.

4. **Apply telescope effects to input spectra from Astroscene:** This is the main functionality of the class where telescope effects are applied to the simulated spectra created by the Astroscene and Foreground modules. This function is performed by the `calc_Ptot_Psig` method of this class.

telescope.class methods:

- `__init__(params.class, intdata.class)`: Initialisation of the Telescope object requires as input a parameters object, containing the simulation parameters and the common data object, containing the wavelength grid used by the Astroscene module to produce the simulated spectra and the resultant spectra. Upon initialisation, all telescope parameters are read in from the relevant files and interpolated to the wavelength grid provided as input.
- `get_Temperature(intdata.class)`: This method takes mirror temperature parameters from the .par file and generates a temperature output array for the observation duration incorporating normal random noise (Brownian noise TBI) and temperature as a function of the geometry of the observation (ra, dec, solar aspect angle) (TBI). The output of this method is a 2D float array of temperature vs.time for each telescope mirror.
- `get_Emissivity(string File)`: This method loads the telescope emissivity parameters from a file and interpolates to the wavelength grid of the astroscene. The output is a 2D float array containing emissivity vs. wavelength for each telescope mirror.
- `get_Reflectivity(string File)`: This method loads mirror reflectivity parameters from a file and interpolates to the wavelength grid of the astroscene.

  The output is a two element list containing:

  1. a 2D float array storing reflectivity vs wavelength for each telescope mirror.
  2. a 1d float array which is the product of all the mirror components reflectivities.
- `get_Solid_Angle(string File)`: Reads telescope solid angle from a file using solid angle pathname stored in the params file and interpolates to the wavelength grid used by the astroscene. The output is a 1D float array containing the solid angle subtended by the beam vs. wavelength in units of steradians.
- `get_Scattered_Light (string File)`: This method reads in a file containing telescope scattered light as a function of time and wavelength, and to interpolate to time/wavelength grids. The ouput of the method is a 2D array containing the telescope scattered light as a function of wavelength and time.
- `calc_Ptel(params.class, intdata.class, 2d emissivity, 2d reflectivity)`: This method takes as input the mirror temperature profiles generated by the `calc_Temperature` method, the interpolated emissivity and reflectivity tables and models the telescope mirrors as modified black bodies.

  The output of this method is a 1D array containing the telescope emissivity. The telescope emissivity is calculated as:

  $((A_1\ E_1\ BB(T1)\ R_1 + A_2\ E_2\ BB(T2))R_3 + A_3\ E_3\ BB(T3))R_4 + A_4\ E_4\ BB(T4)$

  where A_i, E_i, R_i and Ti are the emitting area, emissivity, reflectivity and temperatures of mirror i.
- `calc_FZodi(intdata.class)`: This method takes as input zodi light produced by the Foreground Module and interpolates to the wavelength grid produced by the astroscene. The output of the method is a 1d array containing the interpolated emission of zodiacal light with respect to wavelength and is taken to be time independent.

- `calc_Ptot_Psig` : This method is the main function of the class and takes the `Fsig` output from the astroscene, the interpolated data from `Fzodi`, reflectivity, emissivity, solid angle, telescop scattered light and telescope emission to produce `Ptot` and `Psig`. All data is written to the `intdata` object to be called by users when required. The output from this method is a two element list containing:

  1. a 2D float array containing the product of all the inputs at the telescope aperture given by:
     $P_{tot} = ((F_{zodi} + F_{sig})\tau + P_{tel} + P_{scat})\Omega\ A_{eff}$
  2. a 2D float array containing the power of the signal that is collected by the telescope and passed through its optics and is calculate as:
     $P_{sig} = F_{sig}\ \tau\Omega\ A_{eff}$

instrument.class: The Instrument object takes input from the params file to perform various tasks required by the instrument module such as interpolating the dichroic transmissions/emissions profiles, generating the Instrument Line Function, interpolating to a spectral resolution grid and integrating across each bin.
instrument.class methods:

- `__init__(params.class, intdata.class)`: Initialisation of the Instrument object requires as input a parameters object, containing the simulation parameters and the common data object, containing the wavelength grid used by the Astroscene module to produce the simulated spectra and the resultant spectra. Upon initialisation, all instrument parameters are read in from the relevant files and interpolated to the wavelength grid provided as input.
- `Load_Dichroic_transmission()`: Reads dichroic transmission file and interpolates to astroscene wavelength grid.
- `Load_Dichroic_emission()`: Reads dichroic emission file and interpolates to astroscene wavelength grid.
- `get_Dichroic_Temperature()`: Returns the temperature of the diachroics as a function of time and wavelength.
- `Calc_Emission_Di`: Takes optics temperature derived in get_Dichroic_Temperature() and generates blackbody emission at astroscene wavelengths for the dichroic channel.
- `Generate_ILF`: Takes passband data from params and models the Instrument Line Function (ILF) as tophat of width equal to the finest spectral resoltion of the passband, with area normalised to 1.
- `Get_Pchannel_Components(2d Ptot, 2d Psig)`:Takes the emission and transmission profiles of the dichroics and passes Ptot and Psig through the system, effectively splitting the received power into channels. Returns 2 3D data cubes containing the transmitted power through the optics.
- `Convolve(2d Input, 2d ILF)`: Convolves Pchannel_tot, Pchannel_sig with the Instrument Line Function (ILF).
- `Generate_R_bins( 1d channel)`: Generates new bins for the spectra from the spectral resolution and the maximum/minimum wavelengths of the passbands.
- `Downsample( 2d P_det_tot, 2d P_det_sig, 1d wave, 1d channel)`: Interpolates total and signal spectra to the spectral resolution of the passband and integrates across the element.

- `Detector(2d P_det_tot, 2d P_det_sig, 1d wave, 1d channel)`: Method acts as a temporary detector module. Converts the input power into photons per second falling on the detector. Multiplies by the instrument quantum efficiency to covert to electrons / second.

3.3.6 Functional requirements

Req. ID	Description
TELESCP-FUNC-010	The class will require a `parameter` as input
TELESCP-FUNC-020	The class will require a `internaldata` as input
TELESCP-FUNC-030	The class will read the emissivity of all telescope mirrors from a parameter file. The emissivity for each mirror will be interpolated to the waveleght grid provided by the Astroscene module
TELESCP-FUNC-040	The class will read the reflectivity of all telescope mirrors from a parameter file. The reflectivity for each mirror will be interpolated to the waveleght grid provided by the Astroscene module
TELESCP-FUNC-050	The class will read the transmission of all telescope mirrors from a parameter file. The transmission for each mirror will be interpolated to the waveleght grid provided by the Astroscene module
TELESCP-FUNC-060	The class will require the emitting area of all telescope mirrors as parameter input. The emitting area will be provided by the input `parameter` object.
TELESCP-FUNC-070	The class will require the effective aperture of the telescope as parameter input. The effective aperture will be provided by the input `parameter` object.
TELESCP-FUNC-080	The class will read the telescope solid angle from a parameter file. The telescope solid angle will be interpolated to the wavelength grid provided by the Astroscene module
TELESCP-FUNC-090	The class will simulate the telescope thermal environment for all 4 telescope mirrors as a function of time.
TELESCP-FUNC-091	It will be possible to simulate the thermal environment as constant temperature for the entire time-line. The temperatures to be used will be provided by the input `parameter` object.
TELESCP-FUNC-092	It will be possible to simulate the thermal environment as brownian noise. The temperatures to be used will be provided by the input `parameter` object.
TELESCP-FUNC-093	It will be possible to simulate the thermal environment as a function of the observation geometry (ra, dec, solar aspect angle).
TELESCP-FUNC-100	The class will read the telescope scattered light from a parameter file. The telescope scattered light will be interpolated to the wavelength grid provided by the Astroscene module
TELESCP-FUNC-110	The class will apply and output the effect of telescope optics on the signal produced by the Astroscene Module
TELESCP-FUNC-111	The class will combine and output the effect of telescope optics, telescope emission, telescope scattered light, foreground light on the signal produced by the Astroscene Module
TELESCP-FUNC-112	The spectra produced by the class will be placed in the `internaldata` object.

Req. ID	Description
INSTRUMENT-FUN-010	The Instrument class will require a parameter as input
INSTRUMENT-FUN-020	The Instrument class will require intdata object as input
INSTRUMENT-FUN-030	The Instrument will load dichroic transmission from an external file and interpolate to astroscene wavelength grid.
INSTRUMENT-FUN-040	The instrument will load dichroic emission from an external file and interpolate to astroscene wavelength grid.
INSTRUMENT-FUN-050	The instrument will load dichroic temperatures and generate temperature profiles.
INSTRUMENT-FUN-051	The instrument will incorporate Brownian noise temperature fluctuations in the dichroics.
INSTRUMENT-FUN-060	The instrument will calculate the emission profiles of the dichroics for all observation times based on a blackbody.
INSTRUMENT-FUN-070	The instrument will calculate an Instrument Line Function (ILF) that has a top-hat form, with width defined by the finest spectral resolution of the passband and area normalised to 1.0.
INSTRUMENT-FUN-080	The instrument will split the input of the astroscene into channels by combining with the transmission profiles of the dichroics and will include the emission from the dichroics themselves.
INSTRUMENT-FUN-090	The instrument will convolve the ILF with the channel spectral profiles.
INSTRUMENT-FUN-100	The instrument will generate spectral resolution wavelength grids.
INSTRUMENT-FUN-110	The instruments will down sample the convolved spectral profiles to that of the spectral resolution wavelength grids.
INSTRUMENT-FUN-120	The instrument will integrate across each finite element of the spectral resolution wavelength grid.
INSTRUMENT-FUN-130	The instrument will divide the down sampled spectra by h*c/lambda to convert from W to photons per second.
INSTRUMENT-FUN-140	The instrument will load the quantum efficiency of the detectors from an external file and interpolate to the down sampled wavelength grid.
INSTRUMENT-FUN-150	The instrument will multiply the resulting spectra by the interpolated quantum efficiency and divide by the number of pixels illuminated to return the current in the pixels. Units: electrons /second/pixel
INSTRUMENT-FUN-160	The instrument creates the signal to noise ratio of the observation from: SNR = I_sig* t / SQRT (I_tot * t) where t is the integration time of each observation.

3.3.7 Open Issues

1. Currently the `Telescope.class` simulates the telescope thermal environment a time-constant temperatures. Eventually, it will be possible to simulate the temperature as brownian noise. Also not implemented is simulation of temperature as a function of observation geometry. The latter will take some thinking on what is the best way to input the necessary information and how to calculate the temperature as a function of this input.

2. Currently the telescope scattered light is assumed to be zero.

3. During convolution of the ILF with Pchannel_tot, Pchannel_sig boundary effects are seen.

4. Current modelling method of the instrument is comparable to radiometric models. In the future a more detailed instrument modelling function will be implemented. See URD (section 3.4) for development goals of the instrument.

5. Basic detector methods are contained within the Instrument.class. A separate detector module will be created as the more complex instrument modelling is initialised.

3.4 Noise Module

3.4.1 Module Aim

The Noise Module is responsible for simulating different types of noise and adding them to the the simulated timelines3.4.7)

3.4.2 Inputs

Name	Type	Description
`params`	Parameters.class	This parameter object is initialised by the main script and contains simulation relevant parameters. The parameters contained in this object are read in from the `'echosim.par'` file.
`intdata`	Internaldata.class	This is a common object containing the output data from all the modules. The input wavelength array grid from astroscene is loaded from this object. Zodi values are generated for each wavelength in the input grid.

3.4.3 Outputs

The output of this module is the updated `internal` object with all of the Q_tot data now containing contributions from noise due to telescope jitter, thermal variations, detector relative gains and other.

3.4.4 Algorithm

- `pointing jitter`: The pointing jitter can be calculated by one of the following ways

  1. From actual space telescope pointing data:
     In this case a file containing an ra,dec pointing timeline is loaded. The average of ra*cos(dec) and dec are removed from the ra and dec timelines respectively. The mean of the absolute FFT of both ra and dec timelines is returned as the power spectrum of instrument's pointing. The returned pointing power spectrum is multiplied by a gaussian random and along with and along with a normal random distribution as the complex phase, it is inverse Fourier transformed to generate a pointing timeline.
  2. From a set of nodes describing pointing power spectrum:
     In this case a set of nodes corresponding to [frequency, complex power] are provided from the parameters file. This nodes are used to construct the pointing power spectrum, by joining the nodes with an exponential function. The resulting pointing power spectrum, along with a normal random phase distribution is inverse Fourier transformed to create a pointing timeline, based on the input model.

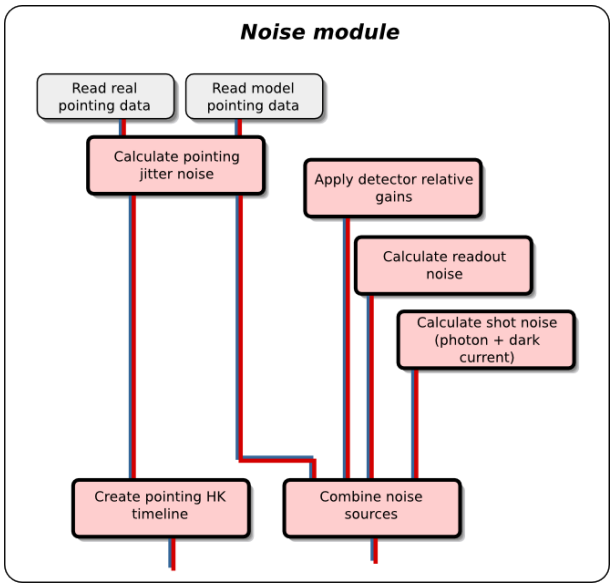

Figure 5: Block diagram of the Noise module, describing the different types of noise added to the simulated time-lines.

- `Brownian noise`: Brownian noise calculated using the Wiener process of the cumulative sum of white noise over the time domain.
  $1/\sqrt{(2\pi\sigma^2 t)} * exp(-(x - mean * t)^2/2\sigma^2 t)$
- `Photon noise`:It is assumed that the photon flux is high enough that a gaussian approximation is suitable.
- `Detector readout noise`: Readout noise is simulated as a gaussian random distribution.

3.4.5 Classes, Methods, Libs Used

noise.class: The noise class is responsible for holding pointing data (either real data, loaded from a file or simulated). In addition the class provides methods to apply noise to the simulated timelines, provided by the Instrument module.
noise.class methods:

- `loadDetectorRelativeGains()`:Method that loads a fits file containing relative gains for all the pixels in each detector.
- `loadHerscelPointingModel()`: Method that reads Herschel telescope pointing model from a fits file. The fits file is expected to have data for frequency and power spectrum. The method returns the power spectrum and the corresponding frequency array.
- `loadHerscelPointingData()`: Method that reads Herschel telescope pointing data from a fits file. The fits file is expected to have data for RA, Dec, and time. The method returns the average power spectrum of ra and dec, and the corresponding frequency array
- `getRandomHerschel(outputArrayLength, inJitterSeed)`: Method to return an array of random numbers based on the power spectrum of Herschel telescope pointing data.

- `makePointingJitterModel(inputNodes, maxFrequency, numSamples)`: Method to return two arrays [frequency, Complex Power], created from a list of (phase,power) points. The returned arrays (as per requirement) have length which is a power of 2.
- `getPointingJitterTimeline(inArrayLength, inDeltaS, inPointingJitterNyqFreq, inModelNodes, inJitterSeed )`: Method to create a simulated pointing jitter (delta theta) time-line, given a model describing the jitter. The model is provided as a list of (freq(Hz),Power) points, contained in inModelNodes. These are provided to the makePointingJitterModel() method in order to create a complex frequency and a complex power array. The method then returns:1. The jitter sampled at 1Hz to be used as Housekeeping. 2. The jitter summed every samplingTime (inDeltaS) to be used by Pointing_Jitter(). 3. The (jitter)^2 summed every samplingTime (inDeltaS) to be used by Pointing_Jitter()
- `Brownian()`: Brownian noise calculated using the Wiener process of the cumulative sum of white noise over the time domain.
- `counter()`: Takes the seed start point and the number of detectors and generates a seed counter which is utilises using the next() function in each noise function.
- `ShotNoise()`: Takes the detector timeline which has already been integrated by the detector time and passes the corresponding photon noise. It is assumed that the photon flux is high enough that a gaussian approximation is suitable.
- `Pointing_Jitter(F_eff, PSFy, PSFgrad,PSFgrad2, x, y, WidPix, Timeline, JitterSeed)`: Takes the detector timeline which has already been integrated by the detector time and passes the corresponding pointing jitter. It is given that the spectrometer has large stability in the wavelength axis and as such the jitter is modelled as fluctuations around the y axis only.
- `Dark_Current()`: Takes the dark current from the par file for the relevant channel and generates a constant dark current value given an exponential function of detector temperature.
- `Detector_Gain(timegrid, Lambda, Seed)`: Takes the mean expected gain from the par file for the relevant channel and generates a random Brownian noise gain.
- `Detector_Readout(Timeline, sigma_RE, Seed)`: Takes the mean expected gain from the par file for the relevant channel and generates a random Detector readout noise.

3.4.6 Functional requirements

Req. ID	Description
NOISE-FUN-010	The Noise module will require a parameter as input
NOISE-FUN-020	The Noise module will require intdata object as input
NOISE-FUN-030	The Noise module will be able to calculate pointing jitter timelines
NOISE-FUN-031	The pointing jitter timelines may be calculated from the properties of pre-existing timelines of earlier space telescopes.
NOISE-FUN-032	The pointing jitter timelines may calculated on the fly from jitter noise parameters provided in the parameter file.
NOISE-FUN-033	The Noise module will provide pointing jitter timelines sampled at a the simulated observation's sampling rate, as well as, at a higher sampling rate to simulate Housekeeping data.
NOISE-FUN-040	The Noise module will be able to apply relative gains to the timeline of pixel of each detector.
NOISE-FUN-050	The Noise module will be able to apply photon noise to the timeline of pixel of each detector.
NOISE-FUN-060	The Noise module will be able to apply relative gains to the timeline of pixel of each detector.
NOISE-FUN-070	The Noise module will be able to apply read-out noise to the timeline of pixel of each detector.
NOISE-FUN-080	The Noise module will be able to apply dark current noise to the timeline of pixel of each detector.

3.4.7 Open Issues

None currently.