The ScopeSim environment

A flexible astronomical instrument data simulation environment in Python

Doc. No.: ELT-TRE-MCD-56306-0060

Kieran Leschinski, Hugo Buddelmeijer, Oliver Czoske, Miguel Verdugo

September 21, 2020

Contents

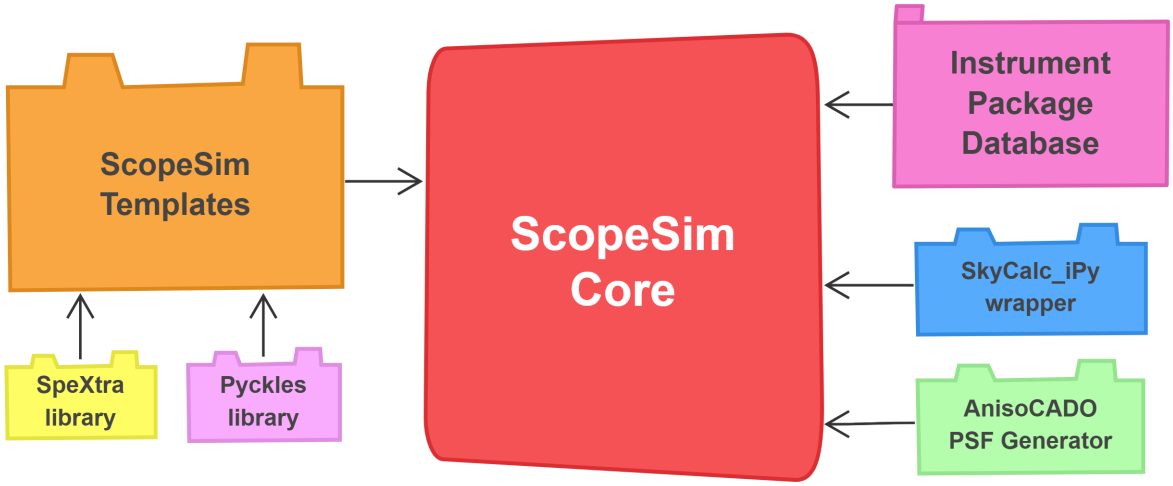

Figure 1: The packages belonging to the ScopeSim environment.

1 Introduction

1.1 An overview of the ScopeSim environment

ScopeSim is a modular and flexible suite of python packages that enable (almost) any astronomical optical (observatory/telescope/instrument) system to be simulated.

The suite of packages can be used by a wide audience for a variety of purposes; from the astronomer interested in simulating reduced observational data, to a pipeline developer needing raw calibration data for testing the pipeline.

ScopeSim achieves this level of flexibility by adhering to strict interfaces between the package, e.g: the ScopeSim engine package is completely instrument and object agnostic. All information and data relating to any specific optical configuration is kept exclusively in the instrument packages hosted in the instrument reference database (IRDB). Similarily, the engine has no clue about what it is observing until runtime. The description of the on-sky source is keep exclusively within the target `templates` package (see `scopesim_templates`).

The core of the ScopeSim environment are the three packages:

- `ScopeSim`: the simulation engine
- `ScopeSim_templates`: descriptions of the on-sky sources
- `IRDB`: the instrument reference database.

In addition to the core package, there are several support packages:

- `AnisoCADO`: simulates SCAO PSFs for the ELT
- `SkyCalc_ipy`: queries the ESO skycalc service for atmospheric spectral curves
- `SpeXtra`: provides easy access to many well-known spectral libraries
- `Pyckles`: a light-weight wrapper for the Pickles (1998) and Brown (2010) catalogues.

Each of these packages will be discussed in detail in the following sections.

1.2 Document Scope

This document is not intended to be a comprehensive description of the ScopeSim environment. Rather is aims to introduce the reader to the elements that make up ScopeSim and directs the reader towards the online documentation for each of the packages, should the reader wish to dive deeper into the material.

1.3 Rationale for the ScopeSim environment

Until now most instrument consortia have developed their own simulators. The general concensus is that every new instrument is sufficiently different from anything that has been previously developed, that it would make little sense to adapt already existing code. This statement is true to some extent. Every new instrument must differ in some way from all existing instruments in order for it to be useful to the astronomical community. However when looked at from a global perspective, every optical system is comprised primarily of elements common to all other systems. Atmospheric emission, mirror reflectivities, filter transmission curves, point spread functions, read-out noise, detector linearity, hot pixels, are just a few of the effects and artifacts that every astronomical optical system contains. Furthermore, while the amplitude and shape of each effect differs between optical systems, there are still commonalities in the way each effect can be described programmatically.

ScopeSim's main goal is to provide a framework for modelling (almost) any astronomical optical system by taking advantage of all these commonalities. What astropy has done for the general python landscape in astronomy, ScopeSim aims to do for the instrument simulator landscape.

1.4 Community involvement

The scopesim environment has been developed as an open source project so that it is possible for others in the astronomical community to be involved in the expansion of both core and perifery functionality. The interface of both the `Effect` and `Source` objects was kept as minimalistic in order to keep the barrier to entry as low as possible.

For the `ScopeSim_templates` package we encourage the submission of both basic and detailed custom `Source` objects for any astronomical source. The package is designed in such a way as to be highly scalable to accommodate descriptions of the myriad of astronomical objects.

For users who need custom `Effect` objects for their optical model, there is the possibility of adding "plug-in" `Effect` objects at run time. We however encourage anyone who has made these custom `Effects` to submit a pull request to the ScopeSim github page, so that we can include these `Effects` in the next major release.

1.5 Documentation and code bases

Table 1: Links to the open source doumentation and code bases

Package	Documentation	Code base
ScopeSim	https://scopesim.readthedocs.io/	https://github.io/astronomyk/scopesim
ScopeSim_templates	https://scopesim-templates.readthedocs.io/	https://github.com/astronomyk/ScopeSim_templates
IRDB	https://irdb.readthedocs.io/en/latest/	https://github.com/astronomyk/IRDB
AnisoCADO	https://anisocado.readthedocs.io/	https://github.com/astronomyk/AnisoCADO

SkyCalc_ipy	https://skycalc-ipy.readthedocs.io/en/latest/	https://github.com/astronomyk/SkyCalc_iPy
SpeXtra	https://spextra.readthedocs.io/en/latest/	https://github.com/miguelverdugo/speXtra
Pyckles	https://pyckles.readthedocs.io/en/latest/	https://github.com/astronomyk/Pyckles

2 The ScopeSim engine

Documentation: https://scopesim.readthedocs.io/

Code Base: https://github.com/astronomyk/ScopeSim

Continuous integration: https://travis-ci.org/github/astronomyk/ScopeSim

Author: Kieran Leschinski

2.1 How the ScopeSim engine works

The scopesim engine is the core of the scopesim environment. At the heart of scopesim are two major concepts.

- The observed output is the target input plus a series of optical artefacts
- The effect of each optical artefact is independent of all other artefacts

If we follow these to concepts to their logical conclusion we end up with a situation where optical artefacts can be treated as a series of "lego bricks" and any digital optical model can be constructed much in the same way as a lego model; by stacking the correct combination of effect "bricks" together in the right order.

The ScopeSim engine is therefore comprised of two types of objects: a collection of `Effect` object subclasses, and a series of "management" classes.

The `Effect` object is a lightweight class that acts as a simple operator class. Object goes in, object comes out. Albeit with the flux distribution slightly altered in one way or another. Here the object in question can be any one of the 4 main flux distribution classes:

- `Source`: (x, y, lambda)
- `FieldOfView`: (x, y, lambda_0)
- `ImagePlane`: (x, y, sum(lambda))
- `DetectorPlane`: (x, y, e-)

> **Warning**
> finish section

2.2 Optical system capabilities

- Imaging
- Spectroscopy (LS, MOS, IFU)
- Simultaneous readouts on multiple image planes

2.3 Explanation of Effect objects

- effects are operators, what is passed, is returned
- **effect classes can alter the object in a variety of ways:**
  - intensity (mult, add, sub) in x,y and/or lambda
  - shifts
  - convolutions

    - spreads

effects are applied one after the other, first lam, then xylam, then xy - effects can also be electronic in nature, - examples of effects, psf, linearity, exposure

2.4 Explanation of observation workflow

- pseudo code for loop
- description of 4 objects, and why they are important

2.5 Documentation

- Tutorials on read the docs

3 On-sky target templates: ScopeSim_templates

Documentation: https://scopesim-templates.readthedocs.io/

Code Base: https://github.com/astronomyk/ScopeSim_templates

Continuous integration: https://travis-ci.org/github/astronomyk/ScopeSim_Templates

Author: Kieran Leschinski

3.1 What is included in the package

- Functions to generate `Source` objects.
- .basic subpackage written by us
- .advanced subpackage for community contributions
- .micado subpackage written by micado pipeline team (HB)
- can add packages for any other topic if anyone has code for this
- functions do not necessarily need spectra. stars for example need mags/fluxes

3.2 Source object interface

- **spatial description**
  - ascii
  - fits ImageHDU
  - fits cube (3D ImageHDU)
- **spectral description**
  - synphot.SourceSpectrum

3.3 Example

> **Warning**
> code to make images for here

- clusters
- galaxies with velocity gradients

3.4 Documentation

- scopesim_templates main documentation
- source object interface documentation from scopesim-templates
- converting from simcado to scopesim

4 Instrument reference database

Documentation: https://irdb.readthedocs.io/

Code Base: https://github.com/astronomyk/IRDB

Continuous integration : https://travis-ci.org/github/astronomyk/IRDB

Author: Oliver Czoske

4.1 What is contained in a packages

Ascii, fits files yaml config files

all effects modes

how to define an effect via yaml

4.2 Main packages vs support packages

- default.yaml
- packages dependencies
- modes_yamls in default

all other yamls just have properties and effects

4.3 List of available packages

4.4 Custom effects example

> **Warning**
> add code to show this (from astrometry notebook)

4.5 Documentation

- Reports of all packages
- repo
- tutorial on how to put together a package

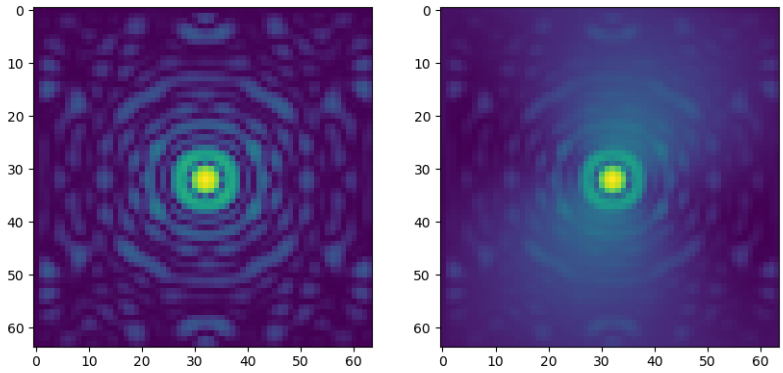

Figure 2: Left: the on-axis K-band (2.15mum) SCAO PSF for standrard atmospheric conditions. Right: the K-band SCAO-PSF at the position (15, -5) arcseconds from the natural guide star.

5 AnisoCADO

A package for analytically generating SCAO PSFs primarily for the ELT

Documentation: https://anisocado.readthedocs.io/

Code Base: https://github.com/astronomyk/AnisoCADO

Continuous integration : https://travis-ci.org/github/astronomyk/AnisoCADO

Author: Kieran Leschinski

5.1 Introduction

- **Why AnisoCADO exists**
  - Quickly create SCAO PSFs
- **Limiting assumptions**
  - Long exposure PSFs (> D_M1 / v_wind)
  - No coherence between time steps

5.2 Examples

- Basic use case
- Off-axis

```
from anisocado import AnalyticalScaoPsf

psf = AnalyticalScaoPsf(N=64, wavelength=2.15)  # um
on_axis = psf.psf_on_axis
off_axis = psf.shift_off_axis(15, -5)   # arcsec
```

5.3 Functionality

- How AnisoCADO creates the SCAO PSFs
- Long / Short exposure PSFs
- Off axis
- Wavelength dependence

6 SkyCalc_iPy

An interactive python wrapper for the ESO SkyCalc_cli package

Documentation: https://skycalc_ipy.readthedocs.io/

Code Base: https://github.com/astronomyk/skycalc_ipy

Continuous integration : https://travis-ci.org/github/astronomyk/skycalc_ipy

Author: Kieran Leschinski

6.1 Functionality

6.2 Examples

7 SpeXtra

`speXtra` is a tool to manage and manipulate astronomical spectra.

Documentation: https://spextra.readthedocs.io/

Code Base: https://github.com/miguelverdugo/speXtra

Continuous integration: https://travis-ci.org/github/miguelverdugo/speXtra

Author: Miguel Verdugo

7.1 Functionality

`speXtra` packages several `synphot` workflows in simple-to-use methods that allow the user to manipulate astronomical spectra. For example, the user can extract the magnitude from an astronomical spectrum or scale the spectrum to that magnitude, the spectrum can be redshifted or blueshifted or smoothed with a velocity kernel, add emission or absorption lines, rebin the spectra or correct it for an extinction curve. Etc.

`speXtra` does not perform measurements, with the sole exception of extracting magnitudes within a pass-band.

`speXtra` also comes with a built-in database of spectral templates that cover many possible user cases. Loading these templates is as easy as typing `Spextrum('library_name/template_name')`

7.1.1 Database

The `speXtra` database contains libraries of spectral templates, extinction curves and filter systems. The data is downloaded on-the-fly when requested and kept cached in the local hard drive for future use.

The scheme is flexible and additional data can be added. In the following a short summary of the database contains is provided.

7.1.2 Spectral Templates

At the time of writing the following libraries are included in the `speXtra` database. Other can be added at request of the user.

- The Kinney-Calzetti Spectral Atlas of Galaxies
- Pickles Stellar Library
- SDSS galaxy composite spectra
- IRTF spectral library
- AGN templates
- Emission line nebulae
- Galaxy SEDs from the UV to the Mid-IR
- Kurucz 1993 Models (subset)
- Supernova Legacy Survey templates (subset)
- Flux/Telluric standards with X-Shooter
- High-Resolution Spectra of Habitable Zone Planets (example)

7.1.3 Extinction Curves

Extinction curves provided with the database.

- Gordon LMC/SMC extinction curves
- Cardelli MW extinction curves
- Calzetti starburst attenuation curve

7.1.4 Filter Systems

`speXtra` currently relies on the spanish SVO filter database to download the filters. This is done thanks to the `tynt` package. However this database is not complete and in particular filters for upcoming ELT instruments are missing. For that reason we have added some filter management to `speXtra`.

Filter system currently included

- MICADO filter system
- METIS filter system

7.1.5 Installation

To install `speXtra` simply type:

```
pip install spextra
```

7.1.6 Dependencies

`speXtra` require the following dependencies to properly work. If they are missing they are automatically installed.

- `numpy`
- `scipy`
- `astropy`
- `synphot`
- `PyYAML`
- `tynt`

7.2 Examples

8 Pyckles

A python package for quick easy access to the Pickles (1998) catalogue

Documentation: https://skycalc_ipy.readthedocs.io/

Code Base: https://github.com/astronomyk/skycalc_ipy

Continuous integration : https://travis-ci.org/github/astronomyk/skycalc_ipy

Author: Kieran Leschinski

8.1 Functionality

- accessing as if spectra are properties
- multiple spectral libraries available

8.2 Examples

- Pickles spectra in various formats
- Brown spectra