LIFEsim

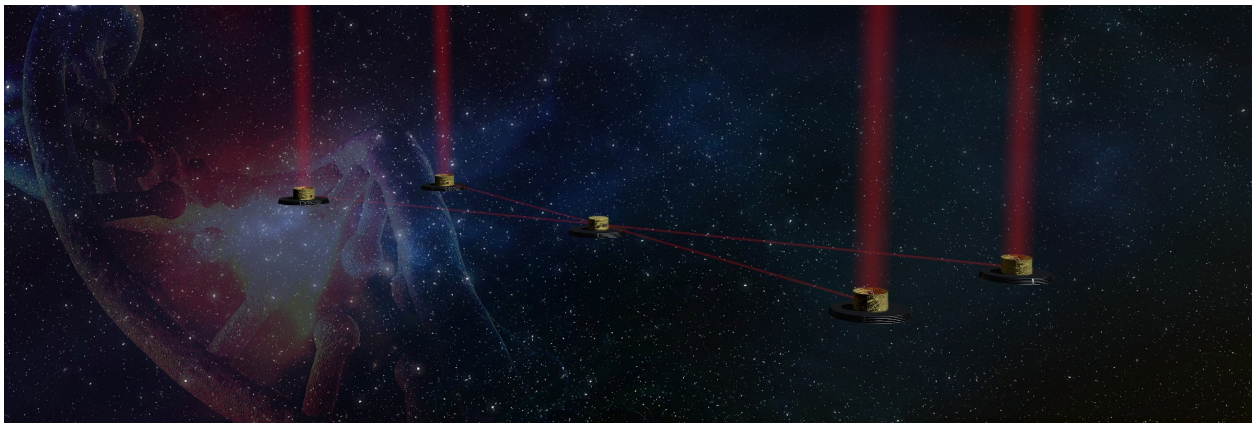
Felix Dannert, Maurice Ottiger and Sascha Quanz

Sep 20, 2023

GETTING STARTED

LIFEsim is the simulator software for the Large Interferometer For Exoplanets (LIFE). It simulates the interferometric measurement of LIFE to demonstrate the capabilities of the observatory. Coupled with on-sky source simulations, LIFEsim will infer the amount and diversity of exoplanets observable with LIFE. Inputting spectral information of exoplanets allows for a description of the spectral measurement potential of LIFE.

For more information, please visit the .



CHAPTER

ONE

INSTALLATION (LINUX AND MAC OS)

LIFEsim is available for installation from . It is compatible with Python 3.8.

Hint: Installation on Mac OS

We recommend the usage of a package manager for mac (e.g.). Follow the installation instruction given by the respective package manager, making sure that you use the correct version of python (3.8).

Open a terminal window and activate conda by running

```
$ conda activate
```

1.1 Virtual Environment

It is highly recommended to install LIFEsim in a *virtual environment*. This ensures installed packages and changes made do not affect other projects on a system. The package dependencies of LIFEsim are best managed with a package manager like pip.

As a first step, *virtualenv* is installed with pip

```
$ pip install virtualenv
```

Then a virtual environment in the folder `new_folder` is created in the current directory

```
$ virtualenv -p python3.8 new_folder
```

Hint: You might need to deactivate the conda base

```
$ conda deactivate
```

To activate and deactivate your new virtual environment use the following statement respectively

```
$ source path_to_new_folder/new_folder/bin/activate
```

Check that you are using the correct python version.

```
$ python --version
Python 3.8.X
```

```
$ deactivate
```

1.2 Download from PyPI (Recommended)

First, activate the virtual environment as shown above. LIFEsim can be installed from PyPI using the pip command.

```
$ pip install lifesim
```

LIFEsim need a modified version of the package to run. It can also be installed using pip. Run

```
$ pip install git+https://github.com/fdannert/SpectRes.git
```

1.3 Download from Github

Navigate to the directory in which you desire to place the LIFEsim repository. Then clone the repository from Github by executing

```
$ git clone https://github.com/fdannert/LIFEsim.git
```

Hint: If `git` is not installed on your system run

```
$ sudo apt install git
```

The dependencies required by LIFEsim can be installed with

```
$ pip install -r LIFEsim/requirements.txt
```

To upgrade already installed dependencies to LIFEsim requirements run

```
$ pip install --upgrade -r LIFEsim/requirements.txt
```

Important: LIFEsim need a modified version of the package to run. Please install it via the following procedure.

First, make sure that you are in the directory where you want to install SpectRes. Then run

```
$ git clone https://github.com/fdannert/SpectRes.git
```

The last step is point the Python install of your virtual environment to LIFEsim and SpectRes. Please do so by running

```
$ echo "export PYTHONPATH='$PYTHONPATH:/path_to_LIFEsim/LIFEsim/:/path_to_SpectRes/
↪SpectRes/'" >> path_to_new_folder/new_folder/bin/activate
```

1.4 Testing the Installation

To test the installation, open a new console and activate the virtual environment as above. Then open Python and import LIFEsim with

```
$ python
```

```
>>> import lifesim
```

If the import statement executes, the installation has been successful. As an extra test run

```
>>> lifesim.util.constants.c
299792000.0
```

This should return the speed of light in [m s $^{-1}$].

Hint: If the `import lifesim` command fails, the reason is likely that the the `PYTHONPATH` is not set correctly. To check for this please run in Python (started with the virtual environment active as above)

```
>>> import sys
>>> sys.path
```

If the path to LIFEsim `'/path_to_LIFEsim/LIFEsim/'` is not returned in the results, please open the file `path_to_new_folder/new_folder/bin/activate` with a text editor of your choice. Then make sure that the last line of the file reads

```
  export PYTHONPATH=':/path_to_LIFEsim/LIFEsim/'

The same test can be performed if SpectRes does not import.
```

CHAPTER

TWO

INSTALLATION (WINDOWS)

2.1 Download Conda

Hint: This step can be skipped if a python distribution and package manager is already installed.

First, a python distribution and package manager is required. For windows, we suggest , a lightweight variant of the popular Anaconda distribution. To install Miniconda, follow the .

2.2 Virtual Environment

It is highly recommended to install LIFEsim in a *virtual environment*. This ensures installed packages and changes made do not affect other projects on a system. The steps of creating a virtual environment with conda are described in the following.

First, open the Conda Prompt, navigate to where you want to create the virtual environment and type

```
> conda create --name new_folder python=3.8
```

Note that we specify the required python version in this command. Some conda commands will ask for confirmation, which can be affirmed by typing.

```
Proceed ([y]/n)?
> y
```

A virtual environment in the folder `new_folder` is created in the current directory

Hint: You might need to deactivate the conda base

```
> conda deactivate
```

To activate and deactivate your new virtual environment use the following commands respectively

```
> conda activate new_folder\
```

Check the python version with the following command

```
> python --version
> conda deactivate
```

2.3 Download from PyPI (Recommended)

Hint: You will likely need to install git first

```
> conda install git
```

With the new environment activated, install LIFEsim via pip.

```
> pip install LIFEsim
```

LIFEsim need a modified version of the package to run. It can also be installed using pip. Run

```
> pip install git+https://github.com/fdannert/SpectRes.git
```

2.4 Download from Github

Navigate to the directory in which you desire to place the LIFEsim repository. Then clone the repository from Github by executing

```
> git clone https://github.com/fdannert/LIFEsim.git
```

Hint: If `git` is not installed on your system run

```
> conda install git
```

The dependencies required by LIFEsim can be installed with

```
> conda install --file requirements.txt
```

Important: LIFEsim need a modified version of the package to run. Please install it via the following procedure.

First, make sure that you are in the directory where you want to install SpectRes. Then run

```
> git clone https://github.com/fdannert/SpectRes.git
```

The last step is point the Python install of your virtual environment to LIFEsim and SpectRes. To do so, please navigate to `site-packages` folder of your virtual environment, most likely located in `` C:Usersuser_nameminiconda3envsnew_folderLibsite-packages``. In this directory, create the file `lifesim.pth` containing the paths to LIFEsim and SpectRes separated by a new line

```
C:\path_to_LIFEsim\LIFEsim\
C:\path_to_SpectRes\SpectRes\
```

2.5 Testing the Installation

To test the installation, open a new conda prompt and activate the virtual environment as above. Then open Python and import LIFEsim with

```
> python
```

```
>>> import lifesim
```

If the import statement executes, the installation has been successful. As an extra test run

```
>>> lifesim.util.constants.c
299792000.0
```

This should return the speed of light in [m s $^{-1}$].

Hint: If the `import lifesim` command fails, the reason is likely that the the `PYTHONPATH` is not set correctly. To check for this please run in Python (started with the virtual environment active as above)

```
>>> import sys
>>> sys.path
```

If the path to LIFEsim `'C:\path_to_LIFEsim\LIFEsim\'` is not returned this is likely the source of the issue.

The same test can be performed if SpectRes does not import.

CHAPTER

THREE

FIRST EXAMPLES

The following examples will explore the current two main functionalities of LIFEsim: The simulation of single exoplanets with a given spectrum and the simulation of the LIFE search phase on a catalog of a simulated exoplanets populated.

3.1 Using the GUI

The following is a minimal working example of simulating a spectral observation of an exoplanet with the LIFEsim GUI.

As a first step, activate the virtual environment containing the LIFEsim install with

```
$ source path_to_new_folder/new_folder/bin/activate
```

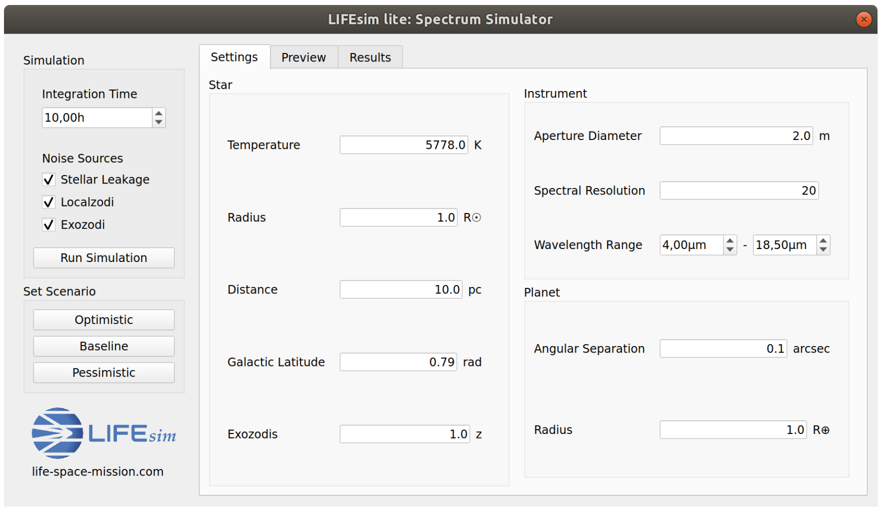
Open python

```
$ python
```

Then, open the LIFEsim spectrum simulator GUI by running

```
>>> from lifesim import Gui
>>> Gui()
```

With executing this command, the following GUI will open



Simulation

Integration Time 10,00h

Noise Sources
✓ Stellar Leakage
✓ Localzodi
✓ Exozodi

Star

Temperature 5778.0 K

Radius 1.0 R☉

Distance 10.0 pc

Galactic Latitude 0.79 rad

Exozodis 1.0 z

Instrument

Aperture Diameter 2.0 m

Spectral Resolution 20

Wavelength Range 4,00μm - 18,50μm

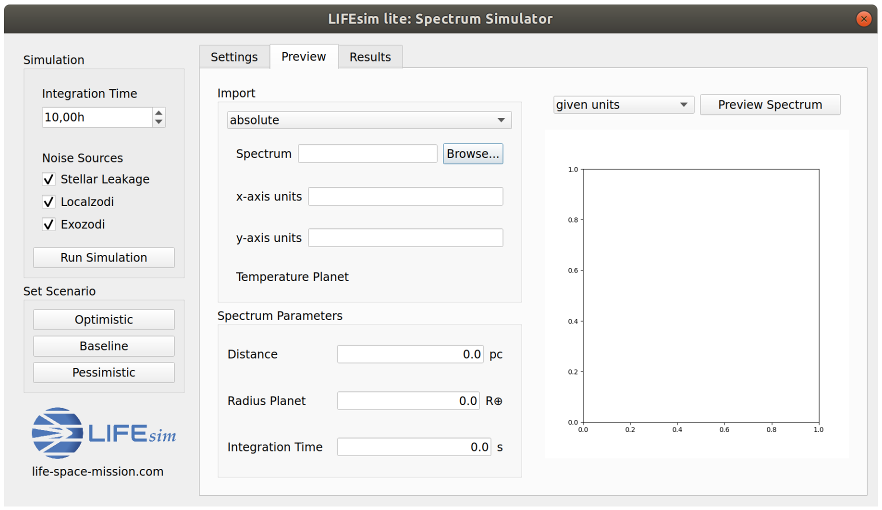
Planet

Angular Separation 0.1 arcsec

Radius 1.0 R⊕

Notice the three tabs for setting the instrument options, importing and previewing the spectrum and displaying the results. In the *Settings* tab, the required parameters for the target star and planet can be set manually. The parameters for the instrument can either be set manually or by pressing on a scenario button on the left side (e.g. *Optimistic*). This then automatically sets the instruments parameters to correspond with what the LIFE team currently views as an optimistic, baseline or pessimistic scenario.

For importing a spectrum, navigate to the *Preview* tab.


LIFEsim lite: Spectrum Simulator

Simulation

Integration Time: 10,00h

Noise Sources
- ✓ Stellar Leakage
- ✓ Localzodi
- ✓ Exozodi

Import: absolute

Spectrum:

x-axis units:

y-axis units:

Temperature Planet

Spectrum Parameters

Distance: 0.0 pc

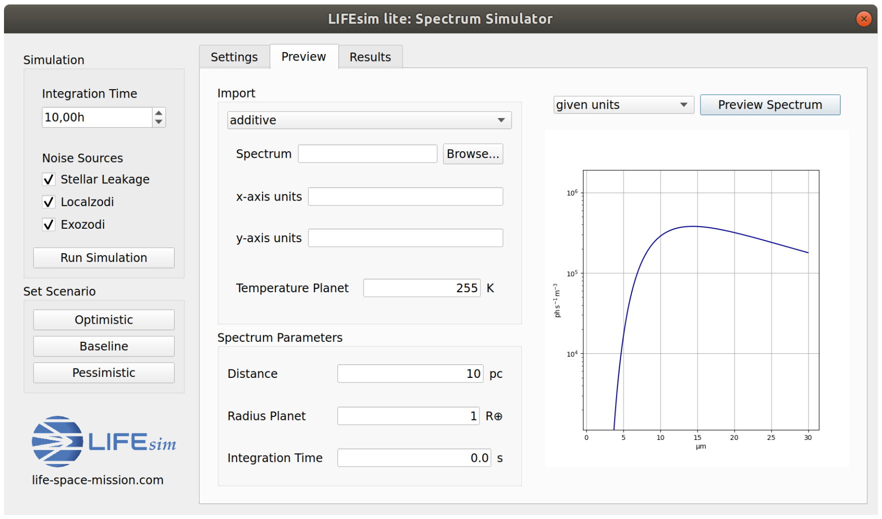
Radius Planet: 0.0 R⊕

Integration Time: 0.0 s

given units

Begin by choosing a spectrum to import (in .txt format) by clicking on *Browse...*. To complete this example, please download and open it in the pop-up dialog. Leave the option as *absolute* to only use the imported spectrum. Setting the option to *additive* will add the imported spectrum to the planets black body spectrum calculated according to given parameters.

Hint: A pure black body planet can be simulated by choosing the *additive* option and leaving the file dialog empty.


LIFEsim lite: Spectrum Simulator

Simulation

Integration Time: 10,00h

Noise Sources
- ✓ Stellar Leakage
- ✓ Localzodi
- ✓ Exozodi

Import: additive

Spectrum:

x-axis units:

y-axis units:

Temperature Planet: 255 K

Spectrum Parameters

Distance: 10 pc

Radius Planet: 1 R⊕

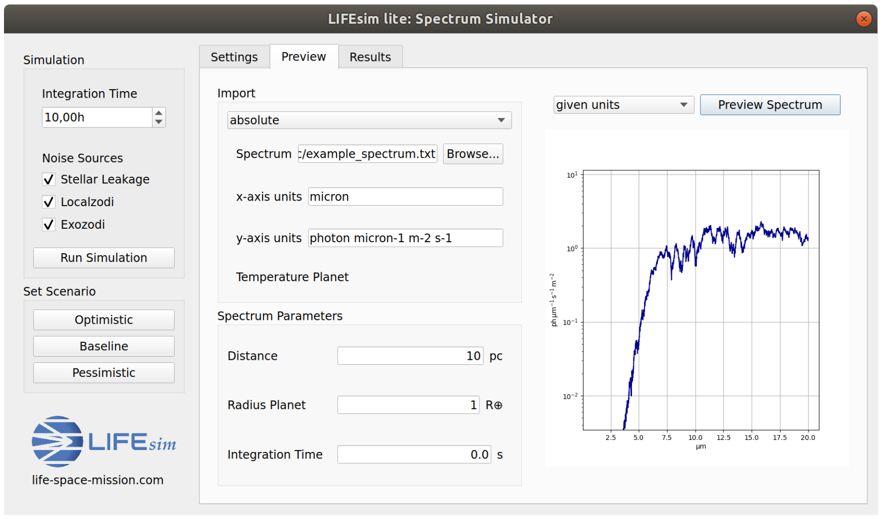
Integration Time: 0.0 s

given units

To specify the units of the spectrum you are importing, enter them in the fields *x-axis units* and *y-axis units*. For this example, please set *x-axis units* to `micron` and *y-axis units* to `photon micron-1 s-1 m-2 sr-1`.

In the *Spectrum Parameter* field the parameters used during the creation of the spectrum need to be given. Again, for the example please set *Distance* to `10pc`, *Planet Radius* to `1 Earth Radius` and leave *Integration Time* at 0.

Pressing *Preview Spectrum* will now show the spectrum in the units specified by the user. This can be used to check the correct import of the spectrum.



Simulation

Integration Time 10,00h

Noise Sources

✓ Stellar Leakage

✓ Localzodi

✓ Exozodi

Set Scenario

Import: absolute

Spectrum :/example_spectrum.txt

x-axis units micron

y-axis units photon micron-1 m-2 s-1

Temperature Planet

Spectrum Parameters

Distance 10 pc

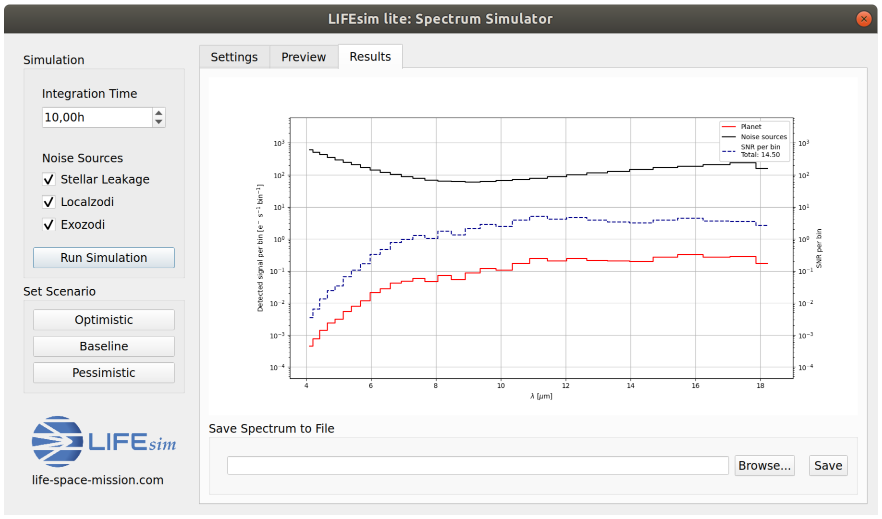
Radius Planet 1 R⊕

Integration Time 0.0 s

given units

Changing the drop-down menu to *converted units* will show the spectrum in the units used in LIFEsim. This completes setting up the simulator for a run.

Change to the *Results* tab and press *Run Simulation* on the very left. This will run the simulation and the display the results as shown below.



Simulation

Integration Time 10,00h

Noise Sources

✓ Stellar Leakage

✓ Localzodi

✓ Exozodi

Set Scenario

Save Spectrum to File

Above the *Run Simulation* button you can change the *Integration Time* of the simulation and select or deselect the inclusion of specific noise sources in the simulation.

At the bottom of the *Results* tab you can choose a location to save the results at by clicking on *Browse…* and then save the results by clicking *Save*.

3.2 Simulating the Search Phase

LIFEsim is capable of taking an artificial exoplanet catalog input from , calculate the signal-to-noise ratio for each planet and distribute the observation time available in the search phase to observe and detect an optimal number of exoplanets.

Hint: The following example mirrors the file `LIFEsim/lifesim_demo.py`.

3.2.1 Set-Up

To run such a simulation, create a new python file. First, LIFEsim needs to imported

```
import lifesim
```

LIFEsim is programmed such that all data and parameters are saved in a single location, which then distributes those to the relevant modules. Create an instance of this so-called `bus`

```
bus = lifesim.Bus()
```

The bus holds all parameters needed for the simulation (e.g. the collector aperture diameter, the duration of the search phase, etc.). They can all be set at the same time by using scenarios predefined by the LIFE team. These scenarios are the 'baseline' case, where the array configuration is as expected, as well as the 'optimistic' and 'pessimistic' case, where the array is set up in a more or less capable way. Set the parameters to the baseline case by running

```
bus.data.options.set_scenario('baseline')
```

Note, that options can also be set manually. For example, the collector aperture diameter can be manually increased to four meters by

```
bus.data.options.set_manual(diameter=4.)
```

A list of all available options and parameters can be found in the API Documentation for *lifesim.util.options*.

3.2.2 Downloading the Catalog

A example synthetic planet population based on statistics from the Kepler mission can be downloaded from the P-Pop github page with the following code. Make sure to replace the path with your local project folder.

```
data = requests.get('https://raw.githubusercontent.com/kammerje/P-pop/main/
↪TestPlanetPopulation.txt')

with open('path/ppop_catalog.txt', 'wb') as file:
    file.write(data.content)
```

3.2.3 Loading the Catalog

Now, the P-Pop catalog can be loaded in. An example catalog can be found in `LIFEsim/docs/_static`. Run

```
bus.data.catalog_from_ppop(input_path='path/ppop_catalog.txt')
```

Important: The given P-Pop catalog populates known stars in the solar neighborhood of up to 20 pc with artificial planets based on the Kepler statistics. This population is done not only once, but in a Monte Carlo approach 500 different universes are simulated. This needs to be kept in mind when the results are interpreted.

With the catalog loaded into LIFEsim, some selections of stars can be removed. The following will remove all A-type stars and every M-type at a distance larger than 10 pc away from earth.

```
bus.data.catalog_remove_distance(stype=0, mode='larger', dist=0.)
bus.data.catalog_remove_distance(stype=4, mode='larger', dist=10.)
```

Hint: LIFEsim uses the following numeric integer keys for stellar types:

0 = A, 1 = F, 2 = G, 3 = K, 4 = M.

With this, the setup for the simulation is complete.

3.2.4 Creating the Instrument

Now, an instance of the LIFEsim instrument module needs to be created.

```
instrument = lifesim.Instrument(name='inst')
```

To give any module access to the data and parameters used in a simulation, it needs to be connected to the bus.

```
bus.add_module(instrument)
```

Next, all modules needed for the instrument module to run need to be created. A list of the required modules can be found in the API Documentation of *lifesim.instrument.instrument*. First, create the module responsible for simulating transmission maps of a four-arm nulling interferometer and add it to the bus.

```
transm = lifesim.TransmissionMap(name='transm')
bus.add_module(transm)
```

Next, create the modules for the simulation of the astrophysical noise sources and add them to the bus.

```
exozodi = lifesim.PhotonNoiseExozodi(name='exo')
bus.add_module(exozodi)
localzodi = lifesim.PhotonNoiseLocalzodi(name='local')
bus.add_module(localzodi)
star_leak = lifesim.PhotonNoiseStar(name='star')
bus.add_module(star_leak)
```

Now, the instrument needs to be told to which modules it should connect to. Do so by running

```
bus.connect(('inst', 'transm'))
bus.connect(('inst', 'exo'))
bus.connect(('inst', 'local'))
bus.connect(('inst', 'star'))

bus.connect(('star', 'transm'))
```

Note, that not all noise sources need to be connected in order for the instrument simulation to run. If required, individual noise sources can be disconnected by running

```
>>> bus.disconnect(('inst', 'exo'))
```

3.2.5 Creating the Optimizer

The optimizer is responsible for distributing the available observing time onto the individual stars. Analogously to above, run

```
opt = lifesim.Optimizer(name='opt')
bus.add_module(opt)
ahgs = lifesim.AhgsModule(name='ahgs')
bus.add_module(ahgs)

bus.connect(('transm', 'opt'))
bus.connect(('inst', 'opt'))
bus.connect(('opt', 'ahgs'))
```

3.2.6 Running the Simulation

First, the signal-to-noise ratio needs to be calculated for every planet in the catalog. To do so, run

```
instrument.get_snr()
```

This function will assign every planet the SNR after one hour of observation. Since the simulation is entirely contained in the radom noise case, the SNR scales with square-root of the integration time. Therefore, the SNR for any integration time can be calculated by knowing the SNR of a specific integration time.

Note: It is not unusual for `instrument.get_snr()` to take up to an hour to complete.

Knowing the SNR for each planet, the integration time can be optimally distributed by

```
opt.ahgs()
```

In the baseline case, the observation time is distributed such that the number of planets in the habitable zone around their respective host stars is optimized. The optimization can be changed to respect all planets by setting

```
>>> bus.data.options.optimization['habitable'] = True
```

Check the API Documentation for *lifesim.util.options* to see all parameters used in the optimization process.

3.2.7 Saving the Results

After a simulation run, the results can be saved as a hdf5 file for later analysis by using

```
bus.data.export_catalog(output_path='path/filename.hdf5')
```

3.2.8 Reading the Results

A previously saved simulation can be read into LIFEsim by running

```
bus_read = lifesim.Bus()
bus_read.data.options.set_scenario('baseline')
bus_read.data.import_catalog(input_path='path/filename.hdf5')
```

3.2.9 Interpreting Results

All results are saved in the catalog located at `bus.data.catalog`. This pandas data frame contains rows representing the individual artificial exoplanets. The meaning of most columns can be found in the file `LIFEsim/lifesim/core/data.py`. The most important columns are listed in the following:

- `'nuniverse'` : The index for the universe the planet is located in. See 'Important' box above for further explanation.
- `'nstar'` : The unique index for the star.
- `'habitable'` : Is `True` if the planet resides in the habitable zone of its host star.
- `'snr_1h'` : The signal-to-noise ration the planet would have after one hour of integration time.
- `'detected'`: Is `True` if the planet is observed long enough to have an SNR larger than bus.data.option.optimization['snr_target']. In this case, the planet is counted towards the planets detected in the search phase.
- `'int_time'`: The amount of integration time spend on the system in [s].

Interpretation of this catalog is easily facilitated by the usage of selection masks. For example, the number of detected exoplanets in the habitable zone around M-type stars would be retrieved via

```
>>> import numpy as np
>>> mask_mtype = bus.data.catalog.stype == 4
>>> mask = np.logical_and.reduce((bus.data.catalog.detected, bus.data.catalog.habitable,␣
↪mask_mtype))
>>> result_number = mask.sum()/500
```

Note the division by 500 to factor out the 500 simulated universes.

CHAPTER

FOUR

ARCHITECTURE

4.1 Introduction

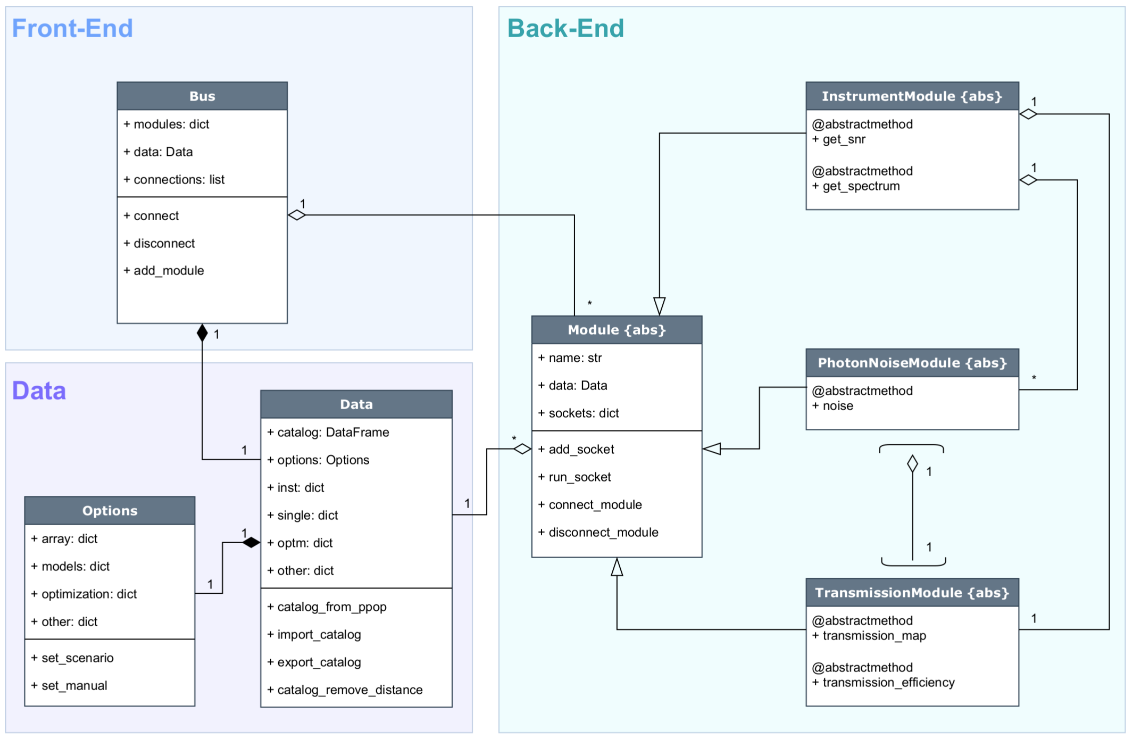
This section is meant to function as a starting point for anyone interested in contributing to LIFEsim or understanding the underlying programming ideas behind the package. First, the core of the package is described in some detail. This is then followed up by an example on how to program a new LIFEsim module.

4.2 General Structure

Apart from the functional goals, the core of LIFEsim is programmed to achieve two major goals:

1. LIFEsim must be easy to use for users unfamiliar with programming and for programmers unfamiliar with the code
2. LIFEsim must be easy to extend.

The implicit aim of goal (1.) is that all members of the LIFE team, largely independent from their background and level of knowledge, should be able to run a LIFEsim simulation. The implicit aim of goal (2.) is that a student can contribute to LIFEsim in the scope of an ETH semester project.



The diagram above outlines the general core architecture of LIFEsim.

4.3 The Bus

The Bus class is the main interface between the user and a LIFEsim simulation. Hence, it is considered to be part of the LIFEsim front-end. All modules taking part in the simulation are aggregated by the Bus. Via the Bus, the user is able to connect modules via their sockets. Furthermore, the Bus holds a single Data class by composition and forwards it to any modules connected to the bus. This makes sure that there are no conflicting instances of the Data class in a single simulation.

4.4 Modules

The underlying idea behind the Module class is that every physical, (partially) self-contained model is written into its own Module. This simplifies the program layout and makes it easier to understand on an abstract level. If a new physical model is to be added to LIFEsim, the programmer should create an abstract class inheriting from the Module class. Then, the programmer should identify all functions that the new module should fulfill. These functions are implemented as a stub abstract method, forcing any inheriting modules to implement the respective functionalities. Note here, that on this level the modules should not contain any functional code, but are rather filled with placeholder pass statements.

4.5 The Data Class

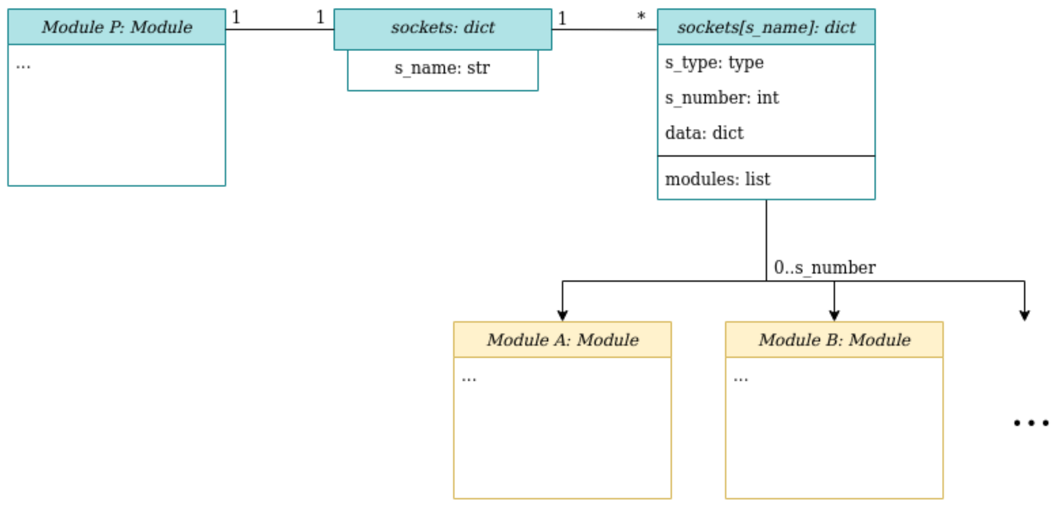
All data, parameters and options for LIFEsim are contained in the Data and Options classes. Located in the Data class is a catalog containing all target stars and exoplanets and additionally allocated space to save variables needed in the respectively used modules. The catalog can be directly imported from P-Pop.

The Options class saves parameters in dictionaries split after which part of the simulations the parameters concern. The parameters can be set according to existing baseline, optimistic and pessimistic scenarios for a future realization of the LIFE mission. A list of all options and the scenarios can be found in the tables at the bottom of the page. The Options class is connected to the Data class by composition, making sure that only a single instance of options can exist in any given simulation.

4.6 Sockets

The method of using sockets is implemented as a tool for interface management between the modules. This allows for independent implementation and re-implementation of modules. The the following is a description of the sockets functionality and how they should be used.

Each *Module* has an attributed dictionary called *sockets*. The elements of this dictionary is accessed by a string key, which doubles as the name of the socket. If *Module P* (see picture below) is to be connected to *Module A*, the following set-up needs to be done.



An entry in the *sockets* dictionary under a unique socket name `s_name` is created (e.g. `s_name = 'connection_a'`). This element contains itself another dictionary specifying

- the type of module the socket should accept.
- the number of modules that can be connected to this socket.
- a (currently empty) list tracking which modules are connected to the socket.

The set-up of the socket is now complete. Connecting the socket is done via the bus as follows

```python
1 bus = lifesim.Bus()
2
3 mod_p = lifesim.Module_P(name='module_p')
4 bus.add_module(mod_p)
5
6 mod_a = lifesim.Module_A(name='module_a')
7 bus.add_module(mod_a)
8
9 bus.connect(('module_p', 'module_a'))
```

The command in Line 9 searches all sockets of the instance of Module P if they accept connection from an instance of Module A, and connects them if suitable socket is found. The same is done vice-versa, meaning that all Module A sockets are search for connections to Module P.

Now, `module_a` is connected to the socket `connection_a` of `module_p`. It can therefore be accessed in `module_p` by calling `self.sockets['connection_a']['modules'][0]`.

4.7 List of Parameters

NAME	USE	UNIT	DEFAULT
diameter	The diameter of a single aperture	[m]	2 m
wl_min	Minimum wavelength accessible by the spectrometer	[μm]	4 μm
wl_max	Maximum wavelength accessible by the spectrometer	[μm]	18.5 μm
quantum_eff	Quantum efficiency of the detector	[%]	70 %
throughput	Photon throughput through the whole optical path inside the [%]	5 %	
spec_res	Spectral resolution of the spectrometer	[-]	20
bl_min	Minimum possible nulling baseline length	[m]	10 m
bl_max	Maximum possible nulling baseline length	[m]	100 m
ratio	Ratio between the imaging and the nulling baseline (imaging longer)	[-]	6
t_slew	Time needed to slew the array to another target	[s]	10 h
t_efficiency	Fraction of on-target time resulting in science output	[%]	80 %
image_size	Square root of the number of pixels used in the calculation of the transmission maps	[-]	256
wl_optimal	Wavelength for which the baseline is optimized	[μm]	15 μm
localzodi	Model used for the localzodi noise source	-	'darwinsim'
habitable	Model used for the habitable zone	-	'MS'

NAME	OPTIMISTIC	BASELINE	PESSIMISTIC
diameter	3.5 m	2 m	2 m
wl_min	3 μm	4 μm	6 μm
wl_min	20 μm	18.5 μm	17 μm

NAME	USE	UNIT	DEFAULT
`N_pf`	Number of samples per orbital period of a planet	[-]	25
`snr_target`	SNR of exoplanet to count as detection	[-]	7
`limit`	Maximum number of exoplanets per host star stellar type	[-]	∞
`habitable`	Optimize for planets in the habitable zone only	bool	`True`
`multi_visit`	Use multiple visits of stars in the scheduler	bool	`True`
`t_search`	Duration of the search phase	[s]	$2.5\,\mathrm{yr}$
`stat_size`	Size of prediction interval when estimating observation time of star	[-]	0.75
`wl_optimal_lz`	Wavelength for which the localzodi contribution is minimized	[m]	$15\,\mu\mathrm{m}$
`time_scaler`	Scales how important it is to schedule an observation with the correct time	[-]	1
`localzodi_scaler`	Scales how important it is to schedule an observation with minimal localzodi	[-]	0.6
`multi_scaler`	Scales how important it is to schedule an observation for frustrated stars	[-]	2

CHAPTER

FIVE

CONTRIBUTING TO LIFESIM

LIFEsim is a community project lead by the LIFE Collaboration. Any contribution to LIFEsim is much encourage and highly appreciated.

We ask that contributions follow the python style guide.

CHAPTER

SIX

LIFESIM.CORE PACKAGE

6.1 Submodules

6.2 lifesim.core.core module

6.3 lifesim.core.data module

6.4 lifesim.core.modules module

CHAPTER

SEVEN

LIFESIM.INSTRUMENT PACKAGE

7.1 Submodules

7.2 lifesim.instrument.instrument module

7.3 lifesim.instrument.pn_star module

7.4 lifesim.instrument.pn_localzodi module

7.5 lifesim.instrument.pn_exozodi module

7.6 lifesim.instrument.transmission module

CHAPTER

EIGHT

LIFESIM.UTIL PACKAGE

8.1 Submodules

8.2 lifesim.util.options module

8.3 lifesim.util.radiation module

8.4 lifesim.util.habitable module

8.5 lifesim.util.constants module