MIRISim Documentation

Release 2.4.0

MIRI European Consortium

Mar 22, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

An depth description of MIRI (Mid-Infrared Instrument) can be found in this series of PASP papers. In short, MIRI is the Mid-Infrared Instrument on the JWST (James Webb Space Telescope), and it contains an imager, coronographs, and low and medium resolution spectrographs, the latter of which is an IFU (Integral Field Unit). These individual components have differing Fields of View (FOV) and wavelength coverages which are summarised in the MIRI JDOCS page.

`MIRISim` is a python package designed to simulate the instrumental effects through MIRI. It gives the user a sense of what the instrument is capable of, and returns data with the same format as the uncalibrated data coming from the telescope. To do this, it uses the Calibration Data Products (CDPs) created by the MIRI team which describe the team's best understanding of the instrument.

`MIRISim` is provided on a best-effort basis to reproduce the on-orbit performance of MIRI. It is not a full telescope simulator, nor is it complete. There are still some effects actively being studied by the instrument team, and as these effects are characterised and quantified, and related products are delivered to, and verified by, STScI, they will be added to `MIRISim`. In this sense, `MIRISim` is presented as representative of the current understanding of MIRI. Its outputs have been tested against both the JWST exposure time calculator (ETC), and the MIRI sensitivity model [Glasse et al. 2015].

`MIRISim` can be used from the command line, or from within python. Simulation parameters (such as whether to simulate the Imager, MRS or LRS, how many exposures, dither positions, etc) are specified using formatted input files, and `MIRISim` creates a set of detector images representative of the outputs of the instrument. For the MRS (Mid-Resolution Spectrometer) a cube representation of the user supplied input targets is also created. These output detector images are consistent with the expected on-orbit performance of MIRI itself, and contain the right metadata for JWST pipeline processing.

The general work-flow within `MIRISim` consists of the following steps:

- Create (from formatted object descriptions or imported .FITS file) an astronomical scene
- Add in flux from instrument optics, electronics and background
- Simulate image slicer (for MRS or LRS)
- Simulate geometric deformations
- Disperse the image slices (for MRS or LRS)
- Add in stochastic cosmic ray hits, photon noise and read noise
- Output simulated observations.

This user guide is designed to give an overview of how to use the simulator. The first few sections of this guide explain how to install `MIRISim` and quickly get started with it (see *Using MIRISim*). This is then followed by a description of the components of `MIRISim` (i.e. the underpinnings of `MIRISim`, see *Components of MIRISim*), and how they are used. There is then a discussion of the `MIRISim` outputs (see *Simulation Products*), and information on how to get them into the JWST pipeline. There are also appendices which describe in detail how to setup an astronomical scene

for simulation, and Jupyter Notebook walk-throughs for each of the available MIRISim modes (Imager, MRS, and for the LRS - both slit and slitless modes).

Effort has been made to make MIRISim user friendly. A working knowledge of python is therefore *not* required to use MIRISim, and to avoid having to learn a number of different file formats, the required input text files for MIRISim are designed to have the same formatting/structure as those to be used for the JWST Calibration pipeline.

Full MIRISim simulations are intended for those who want to comprehensively simulate data coming from the MIRI instrument on the JWST. This includes:

- The MIRI team during commissioning
- Observers planning observations
- The instrument team refining the data reduction pipeline
- Scientists understanding the data products
- Theorists who want to draw comparisons between their models and what an observer would detect on the sky
- ... and many more.

For MIRISim, we have followed the JWST nomenclature when defining terms used for variables for ease of use. We note that this includes defining the number of integrations, and number of frames within an exposure instead of an integration time (which is consistent with the ETC and APT). To put a MIRISim simulation into the context of the terminology used in the APT, MIRISim is able to simulate 'activities' within larger 'visits'.

1.1 Limitations of MIRISim

Development of MIRISim is done on a best-effort basis, and we list below the significant deviations from the capabilities of MIRI itself:

- No Coronography
- No Simultaneous Imager and MRS observations
- No Mosaics
- Output filenames are not consistent with the JWST file naming scheme
- Where MIRI Calibration Data Products do not exist, we cannot model that effect

For the last point, the Calibration Data Products (CDPs), which are described in more detail later, are the MIRI team's best knowledge of the instrument. Where a CDP is still under development (e.g. fringing flat field and stray light correction), we cannot introduce those effects into the simulator. The JWST pipeline is based on STScI derived calibration reference files that, for MIRI, are often based directly on the CDPs. Thus, if there is no MIRI team derived CDP to introduce into the simulator, that effect is not yet calibrated 'out' in the pipeline either. In this respect, the simulator and the data reduction pipeline are consistent with each other.

CHAPTER

TWO

GETTING AND INSTALLING MIRISIM

MIRISim is written in Python, and includes dependencies on a number of Python packages (such as astropy, numpy, scipy, and many others). To manage dependencies, MIRISim has been bundled into a software package, which sets up within an Anaconda environment and installs the required dependencies into that environment. Thus, the first step is to download Anaconda (for Python 3).

No portion of installing or using MIRISim requires root permission on the host computer. No working knowledge of Python is required to use MIRISim: it can be used as a stand alone program at the command line. However, MIRISim componeents can be imported into Python for use. Enabling interaction both at the command line and within Python (i.e. not implementing a graphical user interface) was chosen to allow for the ability to script MIRISim.

Note: New in Version 2: MIRISim has been updated to Python 3.

2.1 Anaconda pre-requisite

MIRISim development is being done in python 3.6. And as its package manager, having Anaconda installed on your system is a pre-requisite for installing MIRISim, just as it is for the JWST pipeline. The 'miniconda' alternative to Anaconda (which installs the minimum number of packages required to properly run python) has been tested, and will allow MIRISim to be installed. However, this alternative *does not save on disk space*, as the MIRISim install script downloads the remaining python packages required for a complete Anaconda installation. At this time, there is no support for installing MIRISim outside of an Anaconda environment.

note macOS users: please ensure XCODE is installed on your computer. Instructions can be found here.

warning With the recent updates to Anaconda, previous versions of MIRISim can no longer be installed. Anything older than v2.2.0 can no longer be installed because of fundamental changes to the Anaconda channels, and how they are referenced in installation. The MIRISim anaconda environment has been shown to be stable when using Anaconda 4.7.11

2.2 Downloading MIRISim

To access MIRISim, go to Miricle.org, and click on the link labelled 'mirisim install' to get to the page with the installation script and instructions. There is no password required for downloading and installing the public version of MIRISim. If prompted for a password, you're on the wrong page.

MIRISim is still actively under development. In light of this, a stable version has been frozen for release. The stable version is the default version to download, when running the installation script:

```
./mirisim_install.bash
```

To install a specific version of mirisim, use the `--version` option. In this case, the install script command takes the form:

```
./mirisim_install.bash  --version=N
```

Note the two dashes before 'version', with no spaces between the dashes and word.

> **warning** installation versions lower than 10 are no longer compatible with generating Anaconda environments.

The installation script also installs a number of potentially large required files (e.g. CDPs, and pysynphot data models) which can take a few minutes to download if not already stored locally. The above commands should install `MIRISim` appropriately for your configuration on either Linux or macOS/OS X. There is currently no support for Windows (as is also true for the JWST pipeline).

When installing `MIRISim`, the final output of the install script to the screen will note the setup commands required to export the `MIRISim` variables to your terminal. It is **highly recommended** that these commands are saved in your `.bashrc` or `.bash_profile` files. These outputs take the form:

> export MIRISIM_ROOT=/HomeDirectory/mirisim
>
> export PYSYN_CDBS=$MIRISIM_ROOT/cdbs/

2.2.1 Installing data files and data models in non-standard location

If you wish to install `MIRISim` to somewhere other than your home directory, set the MIRISIM_ROOT variable before installing using either:

```
export MIRISIM_ROOT='directory'
```

or:

```
setenv MIRISIM_ROOT 'directory'
```

Similarly, to specify where `MIRISim` stores the CDP files,:

```
export CDP_DIR='CDP_directory'
```

For a more in depth description of the calibration data products, please see the section on CDPs.

2.2.2 Activating `MIRISim` environment

To activate the MIRISim anaconda environment:

```
export PYSYN_CDBS=/YOUR/DIRECTORY/MIRISIM/cdbs/
conda activate mirisim
```

where *mirisim* is the name of the environment you created (As specified when your installed MIRIsim), and output at the end of the installation script. Note that this will only work in the current terminal, or one that has been opened/refreshed after the installation of the anaconda *mirisim* environment. Similarly, to de-activate it (e.g. switch back to your system python):

```
conda deactivate
```

If planning on using MIRISim in future terminal shells, it is recommended to copy the above export commands to your shell start up script (e.g. .bashrc or .bash_profile)

2.2.3 Installation Quick Check

To quickly confirm whether MIRISim has installed properly, please try the following:

After executing the source activate mirisim command above, the command prompt should change to include (mirisim) at the beginning of the next command line. If the installation appears to have been successful, these further checks can be done within python or at the command line (both requiring being in the *mirisim* environment).

Within Python (or IPython), import the miri package via:

```
>>> import miri
```

If this command returns the command line with no additional text, the installation was likely successful. If it however gives an ImportError, then the installation was unsuccessful. In this case, check the install logs to diagnose the problem. The location of these logs will have been printed to screen during installation.

From the command line, if MIRISim is run at the command line without any arguments, and the installation was successful, a help message will appear showing an explanation of how to run a simulation or generate configuration files.

2.3 Disk Usage

The outputs of MIRISim become large quite quickly. A fresh installation of MIRISim and its pre-requisites (including things like Anaconda, and CDP files) with the outputs of the default (i.e. included) simulator.ini, ima/lrs/mrs_simulation.ini and scene.ini, files results in 12 GB on disk. If outputs are kept from multiple MIRISim executions, the amount of disk space required increases.

2.4 Documentation

The most up to date version of the MIRISim documentation is on the MIRISim webpage. It is the reference document, and therefore when/if there are deviations or inconsistencies between it and any documentation built from the source code, the document on the website is considered accurate.

Because the documentation has been written in ReStructuredText, and compiled in Sphinx, the documentation can be compiled in pdf or html formats from the source code. Information on how to do this can be found on the sphinx website. A version of Sphinx is included in the MIRISim anaconda environment, however the user is required to have their own latex installation if they want to compile the documentation. LaTeX cannot be included in an anaconda environment.

CHAPTER

THREE

GETTING STARTED

MIRISim can be used either at the command line, or from within Python/IPython. From the command line, it can take (up to) three input files specifying the simulation, astronomical scene and simulator parameters to be used. These same files can also be used from within Python, or the same sets of parameters can be independently set within the Python environment (as described in the appendices). Example configuration files can be generated to give the user a sense for how to populate these files, as described further below.

3.1 Configuration Files

To run a simulation MIRISim takes a minimum of one (max. of 3) files listing input parameters. The `simulation.ini` file specifies how to setup the simulation (e.g. MRS SHORT observations in SLOW mode with 100 frames in each of the 5 integrations in the single exposure), and optionally, which astronomical scene to simulate observing. This optional scene setting is done by specifying the name of a `scene.ini` file in the body of the `simulation.ini` file. If the `scene.ini` file is specified at the command line, any scene mentioned in the `simulation.ini` file is disregarded. The third file (`simulator.ini`) exposes expert user variables which are primarily used for testing (e.g. changing CDP version numbers). It contains the recommended values that represent MIRI, is only intended for internal MIRI team testing, and changing these values is not recommended for general users.

Note: The file naming convention is not strict (i.e. filenames can be anything), it is the internal formatting (as described below) that is important.

As a starting point, example input files are included with the code, and can be generated following the instructions below. These text files use the same syntax formatting as the JWST calibration pipeline inputs: with headings captured in square brackets, and sub-headings with double square brackets and indents to highlight their relationship to the primary headings (indents are not mandatory, but recommended for readability).

The user input (.ini) files follow the formatting of a Windows .ini or Linux .cfg file, with the formal definition of the syntax available here.

Note: New in Version 2: some duplicate (redundant) keywords have been removed from the scene definition files, which means that Version 1 scene files will no longer work with MIRISim v. 2.0.0 or newer. See the version 2.0.0 release notes for more details.

Within MIRISim there is the option to generate example configuration files. The general purpose of each of the files is described here. They can then be modified to suit the needs of the user with respect to integration times and other instrument specific parameters (in `simulation.ini` files), or with respect to targets in the input astronomical scene (in `scene.ini` files). Note, the input scene generation is done in the `scene.ini` file, which specifies either individual objects to be placed in the scene, or a user supplied FITS data cube to be interpreted and used for the scene.

3.1.1 Simulation input file

The `ima_simulation.ini`, `lrs_simulation.ini`, or `mrs_simulation.ini` file (which we refer to generally as `simulation.ini` throughout this document) sets up the observing parameters to be used in the simulation. These user definable parameters include:

- Pointing
- Whether to use the imager, LRS or MRS
- Filter or MRS channel to simulate
- Exposures, Integrations and number of frames
- Detector mode (e.g. SLOW or FAST)
- Dither patterns

The integration times for each frame (depending on, for instance sub-array of the Imager) are listed in Table 1 of [Ressler et al. 2015].

3.1.2 Scene input file

The `scene.ini` file gives an example of how to setup an astronomical scene. In the included example, there are three components within the scene:

- Background emission
- Two Point sources with spectral lines
- Galaxy with a specified SED, velocity map, and line of sight velocity distribution (LOSVD)

3.1.3 Simulator input file

The `simulator.ini` file sets up a generalised set of parameters for `MIRISim` to setup the simulation properly. **No parameters in this file should be changed by the general user.** Specifically, this file includes parameters such as mirror size and the version numbers of the MIRI Calibration Data Products (CDPs) associated with, and required by, the simulator to properly simulate distortions through MIRI. For those attempting to test the JWST calibration pipeline, the various detector effects processed in SCASim can be included or suppressed.

Note that unless this file is explicitly specified at the command line (or within python), `MIRISim` will use the internal default values for these parameters.

Generating Example Configuration Files

The default configuration files are produced either at the command line or within a python session. Both instances require the user to be in the correct Anacaonda environment. At the command line, the files are generated using:

```
mirisim --generate-configfiles
```

or, within python

```
from mirisim import MiriSimulation
MiriSimulation.generate_configfiles()
```

This command produces 10 files, all ending in *.ini*. There are five input files, with two copies of each: one with minimal comments, and a second with additional comments (those with *_commented* in their file names) which go into more detail with respect to the significance of each command.

All of the options available for setting up imager, LRS, or MRS simulations can be found in the commented versions of the simulation files (i.e. simulation_commented.ini). Users can modify these parameters at will, and indeed, the input filename as well.

In the case of the commented version of the `simulation.ini` files, all of the available options for a bound set of commands (e.g filter name) are listed in the extended comments. This includes, for instance, the names of all of the imager filters, the naming conventions for the MRS channels, etc.. It is not comprehensive for unbound variables such as number of frames. For guides on setting those parameters, consult the MIRI best practices guides.

Because of the wide range of parameters and options available for generating scene files, a similarly robust commented file is impossible to create. Instead, there are a number of resources available on the `MIRISim` website to help in creating custom scene files. There, a Jupyter notebook has been setup which highlights the available options for each of the types of parameters that are possible for a given type of object to be put in a scene (see also, the examples in appendix 2).

A scene can be generated in one of two ways; within a .ini file, or dynamically within python. To demonstrate this, there is a Jupyter notebook on the `MIRISim` website which gives an example of how to create a scene file within python, and gives the equivalent commands that would create the same objects using a .ini file. This notebook can also be used to output a derived `scene.ini` file. A general overview of setting up a scene file is given in the appendices. There is also an expert guide created by one of the `MIRISim` team members, available through this link

Latents are not well characterised through MIRISim, however if multiple simulations are run within a python session (which includes re-running a cell in a Jypyter notebook), some latent images could persist into subsequent simulator calls. `MIRISim` does not have the full telescope timing and decay models necessary for those residual images to be truly representative of the expected on-orbit latency behaviour, and thus, the residual images should not be taken as representative of the instrument latency. As a default, the preserve_latents toggle in **|simulatorini|** should be set to FALSE.

Expert users can experiment with preserving the latents from a previous simulation as follows for instrument characterisation experiments:

```
from mirisim import MiriSimulation

mysim = MiriSimulation.from_configfiles("ima_simulation.ini", "scene.ini",
                        "simulator.ini", preserve_latents=True)
mysim.run()
```

or alternatively, assuming config files have been loaded into config objects already:

```
mysim = MiriSimulation.from_configfiles(simulation_cfg, scene_cfg,
                      simulator_cfg, preserve_latents=True)
```

The above information is made available for expert users only. Within `MIRISim` there is no attempt to simulate exposures during wait times. Setting the latent preservation is not captured within the setup files, since it can only be used in sequential simulations. If the simulator is reset (as is done at each command line run), no latent information is captured.

CHAPTER

FOUR

USING MIRISIM

In this section, we demonstrate how to use MIRISim; both at the command line, and within python, using the default configuration files included with the code.

4.1 Setting up the MIRISIM environment

As discussed in the *Getting and Installing MIRISim* section, `MIRISim` is installed into a dedicated anaconda environment. As such, that environment needs to be activated anytime `MIRISim` is run. As with any anaconda environment, the `MIRISim` environment is activated using:

```
conda activate mirisim
```

where `mirisim` is the name of the environment (which would vary for other anaconda environments). To return to the system anaconda environment:

```
conda deactivate
```

4.2 Running `MIRISim` from the command line

The simplest, and most reproducable way of running `MIRISim` is from the command line, using a set of input files. As described earlier, there exists a set of example configuration files that can be modified to suit the simulation being run (e.g. setting the MRS channels, number of integrations, dither patterns, etc), or the user can build one from scratch. With these files (generalised to scene and simulation files) in the current working directory, and the anaconda environment set, mirisim is run using:

```
> mirisim simulation.ini
```

The command above assumes that the scene file is specified in the `simulation.ini` file. For the case where a scene file needs to be explicitly stated:

```
mirisim  simulation.ini --scene scene.ini
```

When in doubt, simply typing `mirisim` at the command line brings up a help file which shows the various ways of starting `MIRISim`. It is the equivalent of using `mirisim --help`.

One of the options listed in the help file is `--local`. This sets an override on the CDP fetching algorithms, and allows for the local copy to be used by default.

4.3 Running MIRISim from within Python

Running MIRISim from within Python allows for greater flexibility in terms of both running a number of similar simulations with slightly different properties (e.g. number of exposures), and allows for followup analysis within the same python session. Examples of setting up simulations directly in Python (i.e. without input files) are given in the Jupyter notebooks attached as appendices to this guide.

The input files used at the command line can also be used within Python, by either specifying them by name, or by setting their names as variables. These two methods are described here.

After having started Python or iPython within the MIRISim anaconda environment, the commands for running MIRISim using configuration files within python are:

```
from mirisim import MiriSimulation

mysim = MiriSimulation.from_configfiles('simulation.ini')
mysim.run()
```

where simulation.ini is the generalised form of the simulation file, which should include a link to the scene file to be used. Again, as at the command line, the inclusion of simulator.ini is implicit.

Alternatively, to parse each of the input files, and use them explicitly:

```
from mirisim import MiriSimulation
from mirisim.config_parser import SimConfig, SceneConfig, SimulatorConfig

simcfg = SimConfig('mrs_simulation.ini')
scenecfg = SceneConfig('scene.ini')
simulatorcfg = SimulatorConfig('simulator.ini')

mysim = MiriSimulation(simcfg, scenecfg, simulatorcfg)
mysim.run()
```

The output directory structure is described in detail in the next section, however it is worth noting here that the output directory name is time stamped, which means that the most recent directory name can be captured using:

```
import glob,os
outputdir = sorted(glob.glob('*_*_mirisim',key=os.path.gettime) )[-1]
```

The python equivalent of the command line '–local' command can be accomplished using:

```
from mirisim import cdp
cdp.set_cdp_host_to_local()
```

This sets the location for the rest of the python session. To revert back to the MIRISim defaults, the command is cdp.set_cdp_host_to_online().

4.3.1 Jupyter Notebooks

MIRISim can also be run using Jupyter notebooks. When using Jupyter notebooks, some users have occasionally experienced one of two issues: either they cannot connect to the CDP servers, or the CDPs downloaded to their system cannot be read in by the notebook. In both scenarios, clearing the CDP cache and restarting the simulation seems to solve the problem if the SFTP server is reachable.

4.4 Running MIRISim with an APT file

With MIRISim Version 2.3 and APT version 2020.3.1, it's possible to use MIRISim with the simulation properties provided by a complete APT file. Running MIRISim using APT files has also been tested using APT 27.2, and 2020.2 versions of the APT. Future proofing is not guaranteed because options may change inside the APT. If they do, the APT parser may break.

As with MIRISim itself, the APT parsing variant can be run at the command line, or from within python. From the command line, simply give the APT file and a scene file as inputs when running:

```
mirisim_from_apt ####.aptx scene.ini
```

where ####.aptx is the name of the APT file to be parsed, and scene.ini describes the scene to be used for all observations executed in the APT file. When using the APT parser inside python there are a lot more options available, including the ability to use different scenes for each set of observations in the APT file.

Inside python, the equivalent to the command line code above looks like:

```
import miri.apt_parser
import mirisim.apt
miri.apt_parser.init_log()

from mirisim import config_parser as parser # SimConfig, SimulatorConfig, SceneConfig

# create a scene using the default parameters to simplify the example
scene_config = parser.SceneConfig.from_default()
# read the MIRI observations from the APT file
observations = miri.apt_parser.parse_apt("1232.aptx")
# run a MIRISim simulation using the observation specifications from the APT file
mirisim.apt.run(observations, scene_config)
```

Optional parameters enable the use of the `dryrun' functionality, which creates the .ini files in an output directory, but doesn't do the simulation itself. This gives the user the ability to see what the outputs would be (in terms of simulation.ini file setup). The simulator.ini file can also be specified if required:

```
simulator_config = parser.SimulatorConfig.from_default()

mirisim.apt.run(observations, scene_config, dryrun=True,simulator=simulator_config)
```

4.4.1 Specifying different scenes for the observations in an APT file

In the APT file, there will be a list of targets to be associated with the various observations. These targets will have a target number pre-pended to their names by the APT (i.e. :literal:`1 NGC-188-FTS107-1'). This number (and following space) is required to be included as part of the target name when specifying a dictionary of scenes.

```
# create a scene object from the defaults
scene_config = parser.SceneConfig.from_default()

# create a scene dictionary where each of the scenes to be observed comes
# from the scene_configuration defined above (i.e. they're all the same)
scene = {
        '1 NGC-188-FTS107-1':scene_config,
        '2 NGC-188-FTS107-2':scene_config,
        'NONE':scene_config,  # for an observation with not target defined
        }

# read the MIRI observations from the APT file
observations = miri.apt_parser.parse_apt("1232.aptx")
# run the simulation with the dictionary of scene configurations,
# rather than a single scene
mirisim.apt.run(observations, scene)
```

The limitations of the conversion of APT files to `MIRISim` include only using `MIRISim`'s default dither pattern for the given light path, only simulating the primary path (i.e. no parallel observations), and if the output folder for a given observation already exists in the working directory, that simulation will not be rung (to avoid duplication). To re-run a specific observation, the destination directory needs to be removed.

CHAPTER

FIVE

SIMULATION PRODUCTS

Here, the results of a MIRISim simulation are described. This includes an understanding of which files and directories are created, what each file contains, and what the names mean. As an example, we show an imager simulation using the 10 μm filter, with 100 frames in 1 integration. The scene we simulate consists of two disks with inner holes, and 15 point sources. The scene generation and simulation setup can be found on the MIRISim website.

5.1 Output Directory Structure

In the working directory, MIRISim will create a new folder based on the date and time the simulation was run, ending in _mirisim. The directory structure that follows is:

```
YYYYMMDD_HHMMSS_mirisim/
|--- det_images/
|--- illum_models/
|--- mirisim.log
|--- skycubes/       [MRS simulations only]
|--- simulation.ini
|--- scene.ini
`--- simulator.ini
```

scene.ini, simulation.ini, and simulator.ini are copies of the inputs used for the simulation.

The mirisim.log file captures the information displayed on the screen through the simulation. It holds, for instance, the name of the output directory, the total exposure time of the simulation, which (specific) detector effects were turned on or off (all are turned on by default), a tally of the number of point and extended sources included in the astronomical scene (or that a FITS file was used), and any warnings issued during the simulation.

5.1.1 Detector Images

The detector images stored in det_images are multi-extension FITS cubes, which are formatted to be consistent with the MIRI data product requirements, and for ingest into the JWST pipeline. This includes ensuring the metadata required by the pipeline is recorded properly in the headers

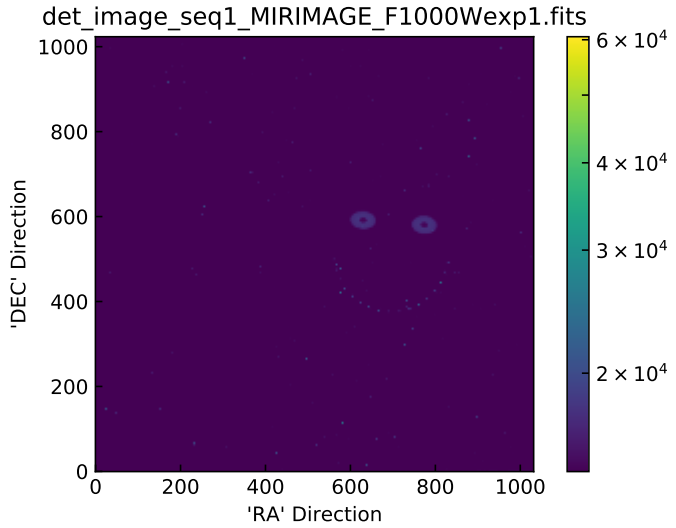
Each cube in this directory corresponds to a single exposure (given by the exp# in each filename). Thus, for four exposures, there will be four FITS cubes.

Warning: the JWST pipeline metadata requirements may change in the future. Our goal is to keep MIRISim output compatible with the pipeline, and hence we need to react to these changes which means outdated MIRISim outputs may not be compatible with newer JWST pipeline builds (and vice versa).

It is the first (not zeroth) extension (a 4D cube) that holds the science (SCI) information in the output FITS files, the other extensions relate to data quality, meta data, dither positions, etc.. Listing the `FITS` extensions using astropy (for example) shows:

```
Filename: det_images/det_image_seq1_MIRIFUSHORT_12SHORTexp1.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     116   ()
  1  SCI           1 ImageHDU        57   (1032, 1024, 5, 3)   float32
  2  REFOUT        1 ImageHDU        15   (258, 1024, 5, 3)   float32
  3  PIXELDQ       1 ImageHDU        11   (1032, 1024)   int32 (rescales to uint32)
  4  ASDF          1 BinTableHDU     11   6651R x 1C   [B]
```

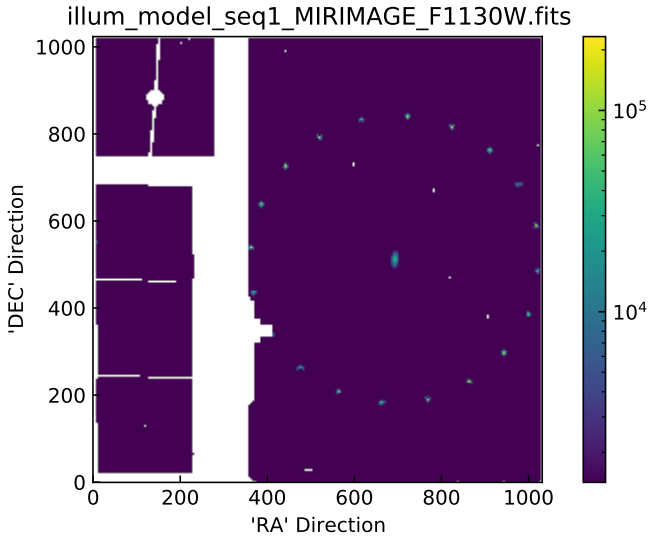
Figure 5.1.1 shows an example detector image using the simulation parameters specified at the beginning of this section. The last frame of the integration is shown to increase the contrast and highlight the signal from the targets in the field.



5.1.1: Example detector image (100th frame in a 10 μm imager filter simulation).

5.1.2 Illumination models

The `illum_models` directory contains the intermediate product illumination models created by the imager, MRS or LRS simulators. These files show how the specified MIRI detector is expected to be illuminated based on the user provided inputs. These intermediate products are then sent to the sensor chip assembly (SCA) simulator to create detector images, where things like exposure times are computed and sensitivities determined. There will be the same number of illumination models as there are detector images



5.1.2: Example illumination Model output from MIRISim (generated by ImSim)

5.1.3 SkyCubes [MRS simulations only]

In the case of an MRS simulation, the `skycubes` directory contains a set of `FITS` cubes with 3D representations of the data cubes used for the simulation (either processed versions of the input `FITS` files, or the cubes generated from the scene generator). It has dimensions consistent with the re-gridding required for an MRS simulation, and can be used to determine whether the input scene has been interpreted properly. Individual cubes are created for each of the 12 MRS sub-channels (e.g. 1 short, 2 long, 3 medium).

CHAPTER

SIX

WHAT MIRISIM DOES TO THE INPUT DATA

6.1 Distortions and transformations applied to the data.

The distortions and transformations applied to the data via CDPs include the effects listed below. For each, a short description of the meaning of the data product is given (often taken directly from the CDP description itself).

6.1.1 PSFs

The Imager PSF is derived from Zemax modelling of the JWST OTE (Optical Telescope Element) plus the (as built) imager optics. There are six diffraction spikes resulting from the hexagonal symmetry of the JWST pupil. These spikes are seen in the measured test data where the JWST pupil has been opto-mechanically simulated in the telescope test optics during all major test campaigns. In addition to the hexagonally symmetric pattern, there is a row and column direction cruciform pattern also seen in test data. These spikes are only seen when the brightest part of the image is falling on the detector, suggesting their origin is due to scattering in the detector itself.

For the LRS, The model PSF was created based on fitting PSF measurements made during testing. For LRS simulations, the user can specify either the LRS PSF CDP, or Webb PSF. This setting can be changed in `simulator.ini`.

To be able to provide a PSF calibration product with high signal-to-noise and for the full wavelength range of the MRS, the PSF is not generated using test data directly. This is because of a lack of point sources across the full MRS wavelength range in the various test campaigns. Instead, we use a simple model of the JWST pupil to simulate an idealistic PSF and modify it according our best knowledge for the MRS.

6.1.2 Distortions

Corrections for the distortion and wavelength calibrations for each light path require mapping known emission patterns (spatial and/or spectral) to what has been observed in test campaigns.

For the imager, the distortion transformations take the form of a number of matrix transformations that convert from the JWST focal plane, to the MIRI Entrance focal plane, through to the detector focal plane and then to the pixel columns and rows.

While a distortion CDP exists for the LRS, and is used by MIRISim, it was found in analysis that the best fit model consisted of a straight line.

For the MRS, once the spectral features of the edge filters and grating wheel transmissions have been identified, and located in the detector, their wavelengths can be assigned to the corresponding (x,y) pixel coordinates of their locations. The (x,y,wavelength) coordinates of the relevant feature define a “reference point” which is then used to identify the etalon lines around it and, by extrapolation, all etalon lines in the detector for that subband.

6.1.3 Imager Pixel area

As a result of the imager distortion, a photometric correction must be applied to account for variations in the field of view of the individual pixels. This correction is equivalent to the area of each pixel projected onto the sky. For the imager, this photometric correction is done using the PHOTOM CDP. For the MRS and LRS, this can be done either by the PHOTOM CDP, or using the PCE (prefered, and described below). Which file is used for the MRS and LRS can be set in the *_simulator.ini configuration file used for a simulation.

6.1.4 Photon Conversion Efficiencies

The PCE is equal to the number of electrons detected per photon incident on the MIRI entrance focal plane within the nominal science beam. The PCE tables are intended to be used on an input photon flux at the MIRI entrance focal plane (i.e. they exclude the telescope optics). They include detector quantum efficiency curves, as well as mirror reflectivities, and a wavelength independent transmission factor of 0.8 to represent the Beginning of Life (BOL) contamination, the End Of Life contamination is not included. The filter profiles have been clipped in wavelength to remove measurement artefacts. These calibration files are favoured over the PHOTOM CDP files for MRS and LRS simulations.

6.1.5 Read noise

This calculation needs to be performed in order to obtain the noise between any pair of consecutive frames in order to work with the pipeline. Once all of the files and integrations have been processed, the program has a large cube of all of the differenced frames. In order to obtain the read noise, we take the standard deviation of all of the differenced images on a pixel by pixel basis, creating a two dimensional image containing the read noise values for all pixels

6.1.6 Bad pixel masks

The bad pixel masks contains the best estimate of those pixels that are dead (do not respond to light), hot (accumulates charge quickly in low light or dark conditions), have an unreliable slope (signals that vary widely from the majority of the pixels) and pixels that have the characteristic RC signature.

6.1.7 Dark current

On the MIRI detectors a pixel's sample-up-the-ramp for exposures taken under dark or identical illumination levels are offset from one another. A pixel's first frame value from identical dark exposures does NOT occur at the same data number (DN) value (even after accounting for read noise and KTC noise) but drifts with time since anneal.

If the simulated results are nearing saturation, and the Dark current is added, MIRISim can produce a warning that the `data array is outside the range of unsigned 16-bit telemetry data`. This behaviour is expected when the addition of dark current pushes the simulation into saturation, hence MIRISim producing a warning, not an error. To avoid this warning, we suggest reducing the exposure time of the observations.

6.1.8 Flat field

Generally speaking, a flat field is a detector response correction image in the ideal case of perfectly spatially uniform illumination. That includes both pixel-to-pixel sensitivity and low-frequency corrections to the detector sensitivity.

For the imager, pixel-to-pixel gain variations are corrected for by using a pixel-to-pixel normalised gain map. The point source flat provides low frequency corrections to the detector sensitivity. It is obtained by moving a well known crowded field to different regions on the detector, with photometry performed at the different locations of the stars in the field. These measurements are used to measure changes in the detector response.

6.1.9 Gain

The gain reference files are used in conjunction with the read noise reference files in the step that finds cosmic rays and noise jumps on a pixel's sample-up-the-ramp values. The gain reference files are used to convert the DN value into electrons. Using the differences between adjacent frame values (converted to electrons using the gain), jumps in the ramp are found by using the expected Poisson noise and read noise.

In addition, in `SCASim`, the following detector effects are applied to the illumination maps:

- Cosmic ray events
- Poisson noise
- Detector non-linearity.
- Detector latency

Each of these detector effects can be turned off in `simulator.ini` for pipeline testing purposes. The latency effect will be overestimated for observation sequences where there are long pauses between observations.

6.2 Distortions and transformations NOT applied to the data

There are a number of known calibrations and transformations that are not applied in `MIRISim`. The list below captures those likely to be incorporated properly in future releases:

- Stray light (via CDP)

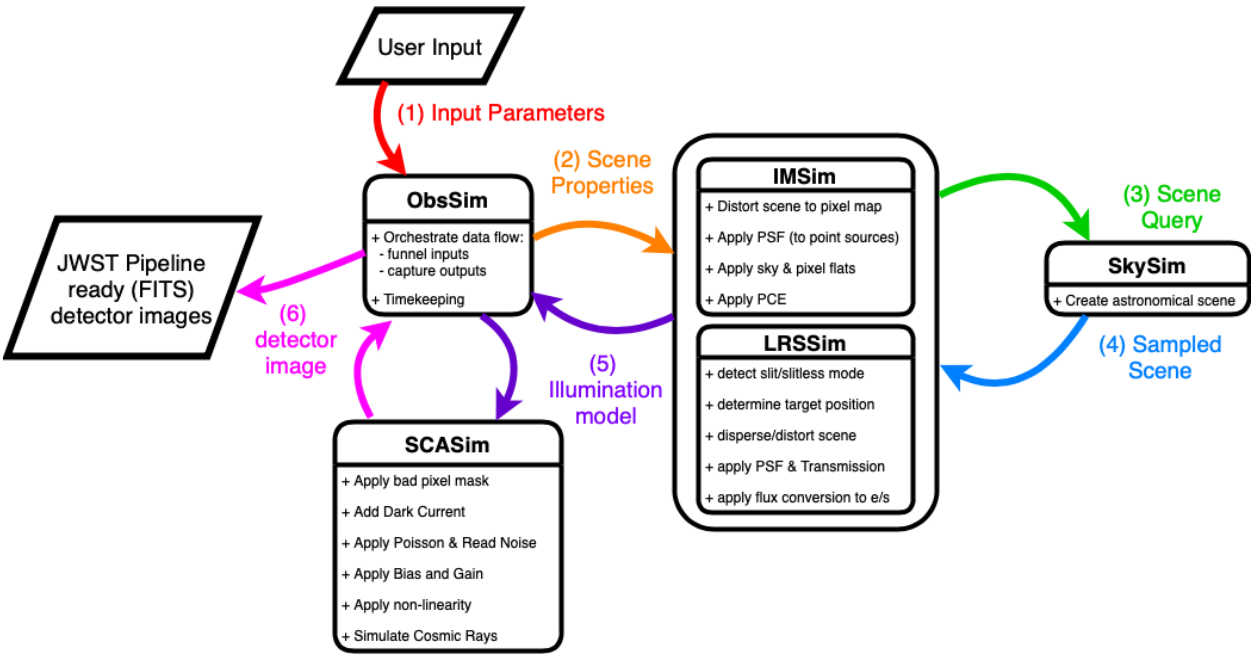
The Calibration Data Products for these calibrations are still under active development, and therefore are not yet corrected for in the JWST pipeline. Once the pre-flight data products have been finalised, they will be incorporated into both `MIRISim` and the JWST pipeline.

CHAPTER

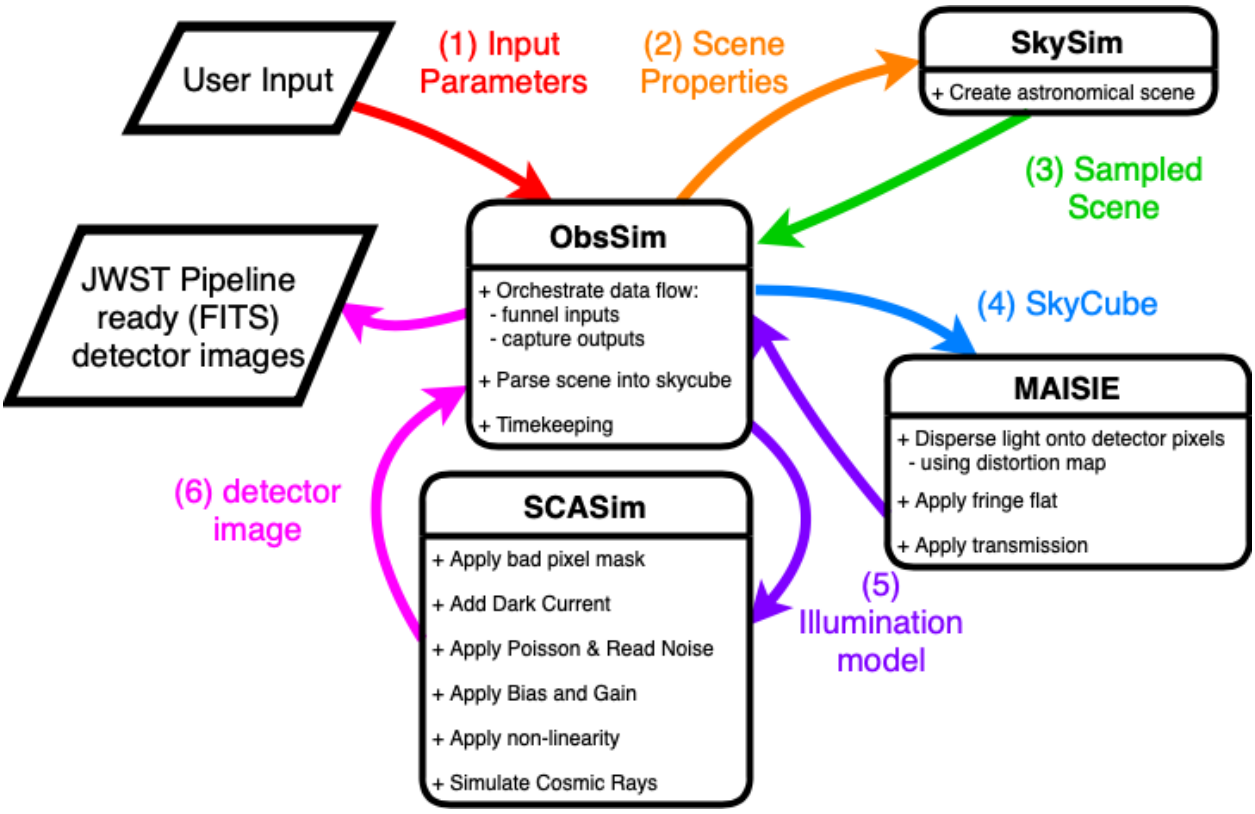
SEVEN

COMPONENTS OF MIRISIM

MIRISim is a python package comprised of a number of components. Many of these components are consistent across all light paths (i.e. the 'observation' simulator, the sensor chip assembly simulator, and the scene generator), however there are separate components for the LRS, imager and MRS specific portions of the simulator. How these components are organised, and the flow of information through a given simulation are shown here in the two flow diagrams (Figures 7.1 and 7.2)



7.1: Flow of information through an Imager or LRS simulation. The steps from user input (1) to final JWST pipeline ready FITS detector images (6) are labelled as they pass through the various components of MIRISim.



7.2: Flow of information through an MRS simulation. The steps from user input (1) to final JWST pipeline ready FITS detector images (6) are labelled as they pass through the various components of `MIRISim`.

7.1 MIRI Coordinate Systems

The internal coordinate systems used within MIRISim are those of the JWST, and all external coordinate systems (e.g. RA, DEC specified in an input FITS file) are converted to these internal (JWST focal plane based) coordinates as the illumination maps and detector images are made (through ImSim/LRSSim/MAISIE and SCASim). The JWST focal plane coordinates can be found here, with a zoom in (which additionally shows the across slice (α) and along slice (β) coordinates of the MRS) are available via the MIRI Pocket Guide.

7.2 Data Models and Inputs

7.2.1 Calibration Data Products

Where possible, MIRISim depends on the calibration data products (CDPs) of the instrument performance delivered by the MIRI team to STScI. Describing the full set of CDPs is beyond the scope of this document, however they are, in essence, the instrument teams best idea of how MIRI will perform. The individual CDPs capture the detector (and other) effects seen in testing. For example, a relevant subset of CDPs used in MIRISim includes photon conversion efficiencies, (fringing) flat fields, imager, LRS and MRS distortions, and PSFs. As instrument development has matured, new versions of the CDPs have been delivered to STScI, and incorporated into MIRISim as required.

The MIRISim installation will create a CDP directory on your computer to store the required versions of the CDP files. A BASH environment variable can be set (e.g. in your .bashrc or .bash_profile file) to create this directory in some location other than your home directory. The relevant line in your .bashrc file is:

```
export CDP_DIR='/your/chosen/path'
```

Not all CDPs are updated with each release, and some CDP changes may take significant effort to be implemented in MIRISim. Therefore, CDP versions used within MIRISim are frozen with each release of the simulator. If/when MIRISim is updated, the new version will automatically check whether the CDP versions required for the simulator are available on disk, and if not, it will retrieve them from the CDP database. The baseline CDP delivery used in MIRISim is currently CDP-7. Any CDP reference file marked with a previous CDP release version number has not been updated/modified since the version referenced.

There are two exceptions to the above. The MRD distortion and flat field CDPs have been updated to versions 8beta.

Notes on CDPs:

There are no CDPs for SLOW mode observations for the pixel flat-field.

Generally, in the event that a user wishes to simulate SLOW mode data, they should be aware that the FAST mode CDPs are being used when no SLOW mode CDP exists. It is not expected for there to be significant differences between these SLOW and FAST mode calibrations, which is why the FAST mode CDPs are being used as substitutes (when required). Because the CDPs do not exist for some SLOW mode calibrations it is unlikely they are being handled properly in the current version of the JWST pipeline either.

7.3 SkySim

SkySim is the MIRISim component responsible for generating an astronomical 'scene' for MIRISim to simulate. This 'scene' can take the form of a user supplied FITS file, or can be created internally from sets of individual target specifications including both spatial and spectral descriptions.

In addition to the brief overviews given in Appendix Section 9, users with a passing knowledge of python can use the jupyter notebook provided on the MIRISim website which gives an exhaustive listing of the types of targets that can be specified and the variables that need to be set for the target size, SED and velocities.

7.3.1 Libraries

Within MIRISim there are a number of built in library models that can be used for target source generation from a text file (using the scene.ini or equivalent text file).

The available models, along with their required/accepted parameters can be queried via the in-built python help system. The key libraries that will likely be of use include the source library (highlighted below), the SED library and the velocity map (velomap) library. The last two can be accessed in the same manner as the source library, chaning source_lib below to sed_lib or velomap_lib.

```
>>>mirisim.skysim.lib.source_lib
{u'Background': mirisim.skysim.background.Background,
 u'Catalogue': mirisim.skysim.catalogues.Catalogue,
 u'ExpDisk': mirisim.skysim.disks.ExpDisk,
 u'Galaxy': mirisim.skysim.astrosources.Galaxy,
 u'PlanetDisk': mirisim.skysim.disks.PlanetDisk,
 u'Point': mirisim.skysim.basicsources.Point,
 u'SersicDisk': mirisim.skysim.disks.SersicDisk,
 u'Skycube': mirisim.skysim.externalsources.Skycube,
 u'Star': mirisim.skysim.astrosources.Star}
```

This shows the types of sources that are pre-installed in MIRISim, and how to get more information on them. Using an example from above, typing mirisim.skysim.basicsources.Point? inside an ipython (not python) session gives the location of the point source code, as well as a list of required parameters.

7.3.2 Units in SkySim

Most quantities used as inputs to make SkySim objects are assigned default units to make specification easier to manage. Below we list the default units assigned to user input quantities, and whether there is functionality available to change those defaults.

7.3.1: SkySim Quantity Default Units

Quantity	Unit	Notes
Wavelength	micron	
Temperature	Kelvin	
Position/Offset	Arcsec	position angles given in deg
Velocity	km/s	Not to be used in combo. with Redshift
Redshift	z	Not to be used in combo. with Velocity
RA	RA—TAN	
DEC	DEC–TAN	
Time	minutes	for time constant

How input fluxes are defined depends on what the input method is.

7.3.2: Units for Flux inputs in SkySim

Input Method	Unit	Notes
FITS cube	μ Jy/arcsec^2	
ExternalSED	μ Jy	Peak Flux at specified wavelength
Line List	10^-20 W/m^2	Integrated Flux over specified FWHM at wavelength

The variations for the three input methods come from the expectations of SkySim. For an external SED specified in a text file, the requirements are to provide a list of wavelengths and peak fluxes. It is assumed that SkySim will be required to interpolate the fluxes between the specified wavelengths (i.e. that the SED is a continuous function). The case of a list of lines, the user is expected to provide lists of wavelengths, integrated intensities and line FWHM values. This results in the expectation that there will only be flux over the (gaussian) region specified by the integrated flux and FWHM of each specified line. In this case, specifying a 'constant' integrated intensity over a fixed FWHM (i.e. 0.001 μm) for each line will result in larger line peaks as a function of wavelength - because the width of the line relative to the rest wavelength is decreasing (i.e. $\Delta\lambda/\lambda$ decreases, so the peak increases for a given integrated intensity).

7.3.3 Reading FITS files

For `SkySim` to be able to interpret a FITS file, it has a minimum requirement on the header keywords present in the file. The header must include (but not necessarily in this order):

```
NAXIS = 3
NAXISi = #of pixels
CTYPEi = (see below)
CRVALi = reference values
CRPIXi = pixel for reference value
CDELTi = size of pixel
```

Where the CTYPEi values take the form 'RA—TAN','DEC–TAN', and 'WAVE'. While not required, the user should not that if the CUNITi keywords are not provided, `SkySim` assumes RA and DEC are in arcsec, and WAVE is in micron. When Skysim is interpreting the input grid, it calculates values relative to a pixel center.

7.4 ObsSim

As the simulation 'conductor', the primary role of `ObsSim` is to interpret inputs (from the user or other `MIRISim` components) and translate them into the required inputs for the next step of the simulation (See Figures 7.1 and 7.2). It also writes out all outputs to FITS files.

`ObsSim` orchestrates which information is given to the individual simulator modules, and in which order. It is also the module to which the others return their end products. `ObsSim` takes the target parameters from the user supplied scene.ini (whether in the form of a collection of targets or a `FITS` file), and presents that information to `SkySim` for processing. Additionally, `ObsSim` takes the observation parameters from simulation.ini to pass to either `ImSim`, `LRSSim` or `MIRI-MAISIE`. It quantifies where the telescope is pointing, what MIRI component is being simulated (e.g. imager or MRS), how long the integration is, and dither pattern information.

7.4.1 Dither Patterns

The dithering recommendations for MIRI observations can be found on the STScI JDOCs MIRI dithers pages.

Key recommended dither patterns, as implemented in the APT have been incorporated into `MIRISim`. Incorporating more APT dither patterns is on the development roadmap, but in the meantime, the included dither patterns are described here.

For the imager, the implemented dither pattern is the large CYCLING pattern, with the same definitions as in the APT. The pattern used in `MIRISim` is spaced in pixels, while that listed in the APT is in arcsec.

For the MRS, we have implemented the point source dither patterns as described in the APT, and are working on implementing the extended source patterns. The patterns optimised for the different channels are available by chosing a different 'ditherstart' argument, with the starting point for each channel listed in the table below. Note that the starting positions for the 'backwards' patterns are listed as well.

7.4.1: MRS Dither pattern starting positions

Channel	Starting Position (+)	Starting Position (-)
1	1	5
2	9	13
3	17	21
4	25	29

So, to optimise the dither pattern for channel 2 observations using the positive pattern, the `StartInd` variable in the `simulation.ini` file or simulation configuration in python should be *9*.

For the LRS, the user can select whether dithering is applied (Option `DITHER` set to TRUE) using the `lrs_recommended_dither.dat` file (2 positions: 7 pixels offset left and right of the slit center) or not. `LRSSim` reads the output of the `SkySim` generated scene defined in the scene.ini file and distinguishes between point and extended sources, background and fits file input (skyCubes).

7.5 LRSSim

While the `LRSSim` module uses the same primary optical path (POP) as the imager (IMA), it is selected by the ConfigPath options `LRS_SLIT` or `LRS_SLITLESS` depending on whether the simulation should be run on the slit position (full array) or the slitless position (SLITLESSPRISM subarray). The filter should be set to P750L which is the filter wheel position for the LRS prism.

The slitless mode only makes sense for point sources. `LRSSim` generates a MIRI illumination model as a 1032x1024 pixel array in slit mode and a SLITLESSPRISM subarray in slitless mode that is fed into `SCASim`. For extended and background sources the Ra/Dec for each pixel of the LRS area on the array is calculated by applying the imager transformation tasks with the present pointing information. The LRS area on the array is defined in the LRS DISTORTION CDP file, and the wavelength for each pixel in this area is calculated using this file. Inside this area there is a dispersion direction and a spatial direction. The current pre-launch distortion CDP file is defined in a way that the spatial direction is in row direction, and each row has a different wavelength. The background and extended sources are projected on the array by applying the skySim task `source.getext(ra,dec,wave)` and then the result is convolved with the LRS PSF.

For point sources, the given offset (`source.Cen`) from the reference position is calculated and the LRS PSF is placed at the offset pixel position using the imager transformation tasks with the present pointing information. The LRS PSF is scaled with the given spectral flux (`source.SED`) for each wavelength after convolving the SED with a normalized Gaussian kernel using a sigma that equals the pixel to pixel wavelength step.

Currently there are 2 possible LRS PSFs that can be chosen by correspondingly setting the boolean parameter `take_webbPsf` in the `simulator.ini` file: The measured PSF from the corresponding CDP file or the STSCI

model PSF (webbPsf). The CDP PSF was measured at MIRI instrument ground tests, the webbPsf is a modelled PSF at user defined wavelengths using the whole JWST optical chain. Applying the webbPsf software (which needs the optical propagation tool poppy) included in the MIRISim anaconda environment a data cube at LRS wavelengths is constructed, that is stored in the `LRSSim` data directory (`webbPsfCube.fits`). This data cube is summed over the slit or slitless pixel area over the first dimension for each wavelength, projected on the reference position on the array.

In case of a fits file input the file content is read as a data cube and a fits header including the WCS keywords. The unit is read from the fits header and the values are converted to microJy/arcsec^2 if necessary. The data cube is interpolated on the LRS wavelengths and convolved with the PSF from the webbPsf cube for each image plain at each wavelength after projecting the image plains on the webbPsf WCS using the software package reproject which is included in the MIRISim environment. The package is written to re-grid images from one world coordinate system to another (for example changing the pixel resolution, orientation, coordinate system). The software is also used to project the slit or slitless area on the image layers for each wavelength using the WCS coordinates of the slit/slitless area. The resulting data cube is summed over the slit or slitless pixel area over the first dimension for each wavelength and projected on the reference position on the array

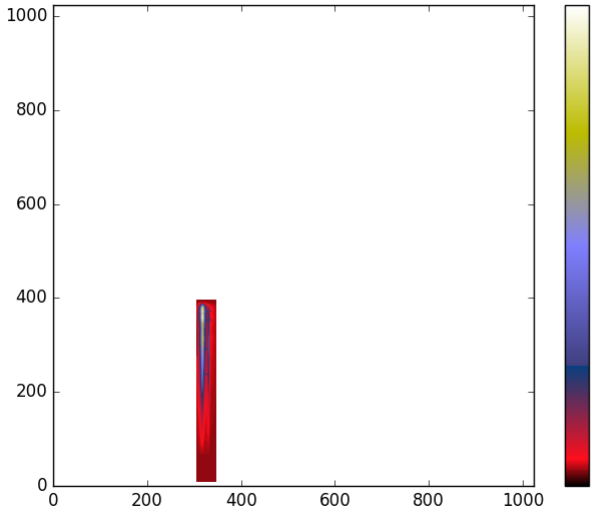
For all kinds of input sources, the given input flux (μ Jy/arcsec^2) is converted to electrons per second by dividing by the Planck constant times the spectral resolution, and multiplying with the photon conversion efficiency CDP file content (PCE, that contains the quantum efficiency), the time dependent JWST + MIRI transmission efficiency, the telescope area and the solid angle of a pixel. By setting the parameter :literal:`use_pce' to false (F) in the `simulator.ini` file, the flux conversion is calculated by using the LRS PHOTOM CDP file. This is done by dividing the flux values by the PHOTOM values with units of MJy/sr/DN/s taking into consideration the solid angle of a pixel. Subsequently, the flux values are then multiplied by the gain to get the final electrons/s unit.

The resulting MIRI illumination model is stored and also fed into `SCASim` (see the `LrsExposure` class in obssim/exposure.py), which adds the detector effects like flat-field, dark current, bad pixels, noise, cosmic rays, latency, drifts and nonlinearity (but not the quantum efficiency since this was already applied in LRSSim using the PCE CDP file) and generates the detector images.

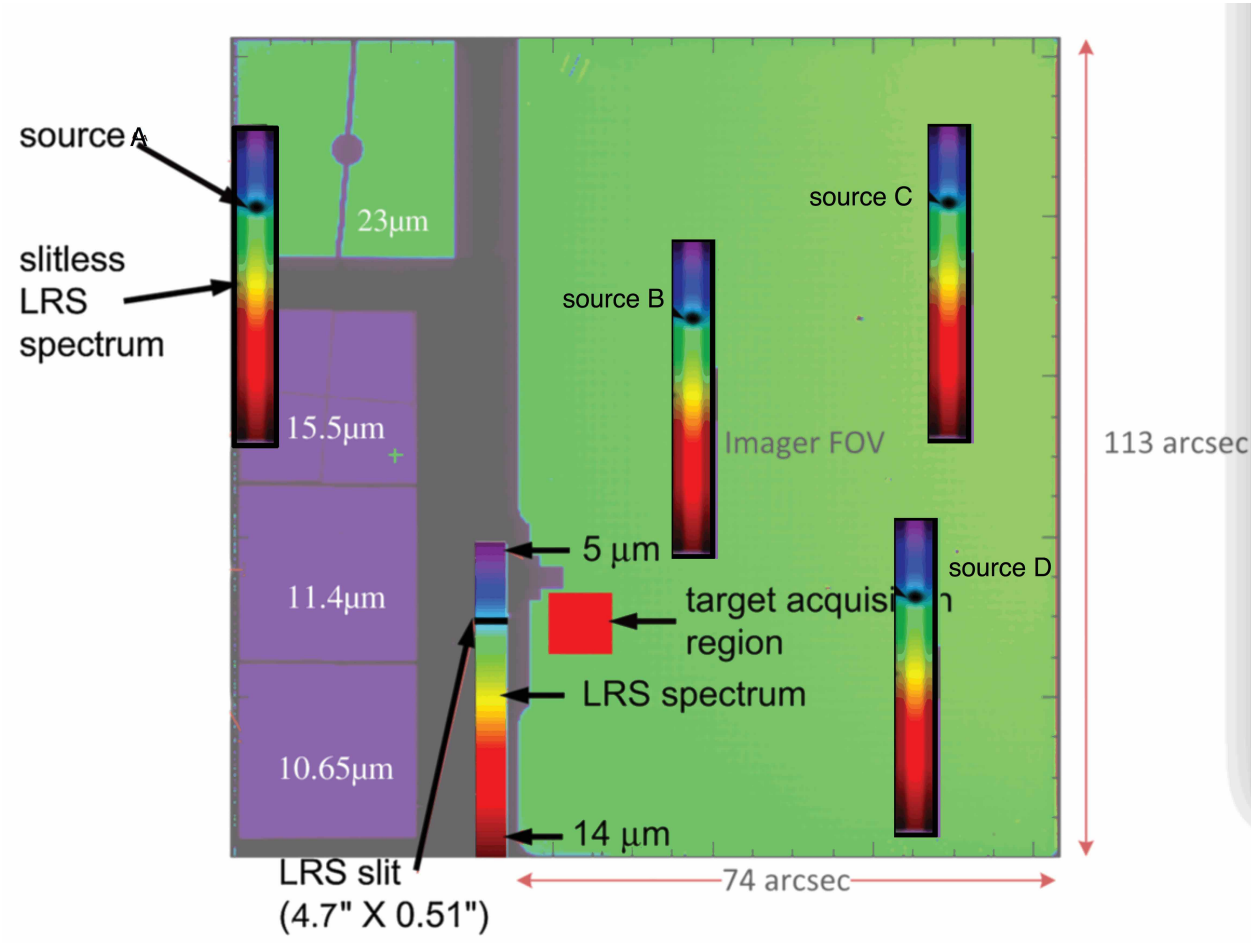
Limitations of the simulator:

1) In the mode LRS SLIT, the LRS simulator produces an illumination model with an image 1024*1024 (full array), filled with `NaN` with the exception of a subarray with the slit coordinates coming from the DISTORTION CDP file. This subarray has a width of 43 and a height of 388 pixels. Its coordinates are x=304 to 346 (first and last included), and y = 8 to 395 (first and last included) (see figure LRS Slit Simulation). In real slit observations, if there are other sources outside the slit their (slitless) spectra will also be visible on the imager array. See [Kendrew et al. 2015] : "There is no shutter or way to block light going into the slit while imaging nor is there a way to mask the imaging portion of the focal plane when taking a low-resolution spectrum. If there are other point sources in the imaging field at the time of an LRS observation, these will appear as slitless spectra on the imaging fields" (see figure LRS Spectral Sources). For slitless observations only a subarray of the upper left corner of the detector array is read (SLITLESSPRISM). This was pushed by the exoplanet community to enable faster readout to observe bright point sources. This subarray has an extent of 72 x 461 pixels and is therefore bigger than the LRS area defined in the DISTORTION CDP file. Therefore, for real observations sources placed outside the area defined in the DISTORTION file but inside the so called SLITLESSPRISM subarray also generate slitless spectra.

2) The LRS CDP DISTORTION file version 7 has a wavelength range from 3.57 micron to 14.39 micron. But for real observations in the mode slitless, the PCE CDP file has a value of 0.01 at 3.5 micron: it is not zero for high flux sources. So we have for wavelengths below 3.5 micron a bright pixel due to the folding of wavelengths. The dispersion curve is flat around 3.5 micron: all the wavelengths fall on the same pixel, which is then bright and eventually saturated (see figures LRS PCE and LRS Dispersion).



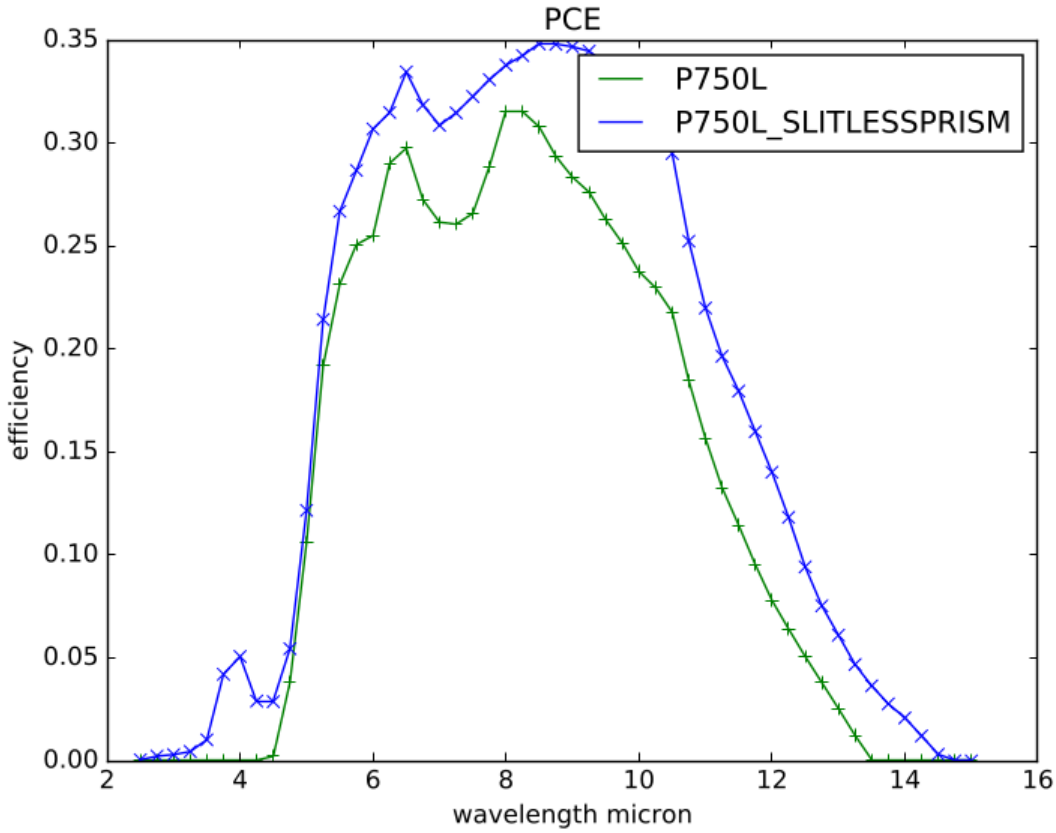
7.5.1: LRS Slit Simulation



7.5.2: LRS spectra of sources outside the slit

7.5.1: Photon conversion efficiencies

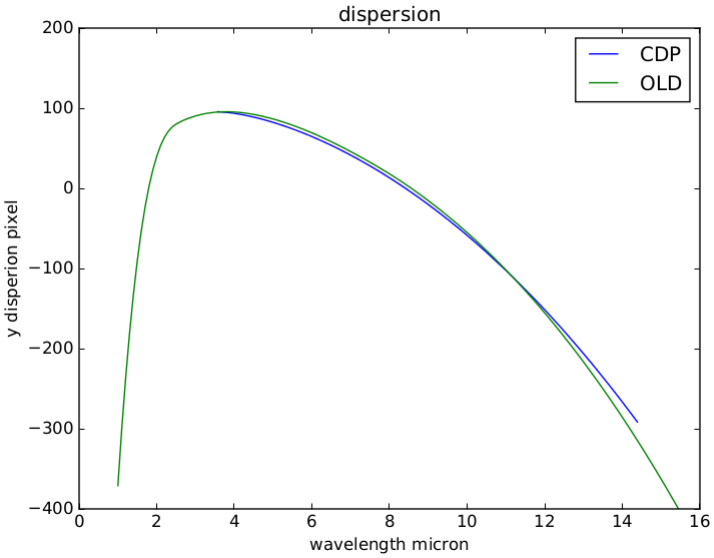
pce in %	0.23	0.30	0.45	1.02	4.20
wavelength (microns):	2.75	3.00	3.25	3.50	3.75



7.5.3: LRS Photon Conversion Efficiency

Limitation (1) can mostly be overcome by changing the parameter `add_extension=T` in the LRSSim seciton of the `simulator.ini` file. The CDP for the pixel flat is only defined in the slit area, which is why it needs to be disabled for now. With these modifications, all sources and the background on the full array for LRS_SLIT and on the SLITLESSPRISM subarray for LRS_SLITLESS observations are considered and calculated as dispersed slitless extended sources. The drawback is a much longer computation time since each row of the whole array has to be dispersed in column direction and then summed to the result of the other rows by considering the focal plane mask (see Figure 7.5.5). This mode is especially useful if a nearby bright source could scatter light into the slit or saturate the array.

As an example the simulation of a `pysynphot` model galaxy with an offset of its center of -40 arcsec in RA and 20 arcsec in DEC from the center FOV (reference position) is shown in Figure 7.5.6. The galaxy has a sersic index of 4 and an effective radius of 100 arcsec, the axial ratio is 0.5 and the position angle 35 degrees. The SED is linearly rising with wavelengths. The sed name is elliptical_template from the family kc96. Additionally, a centered point source with a Black Body spectrum (T=300K) is added. In Figure 7.5.7 the pure background simulation result is shown, there the effect of the focal plane mask can nicely be seen.



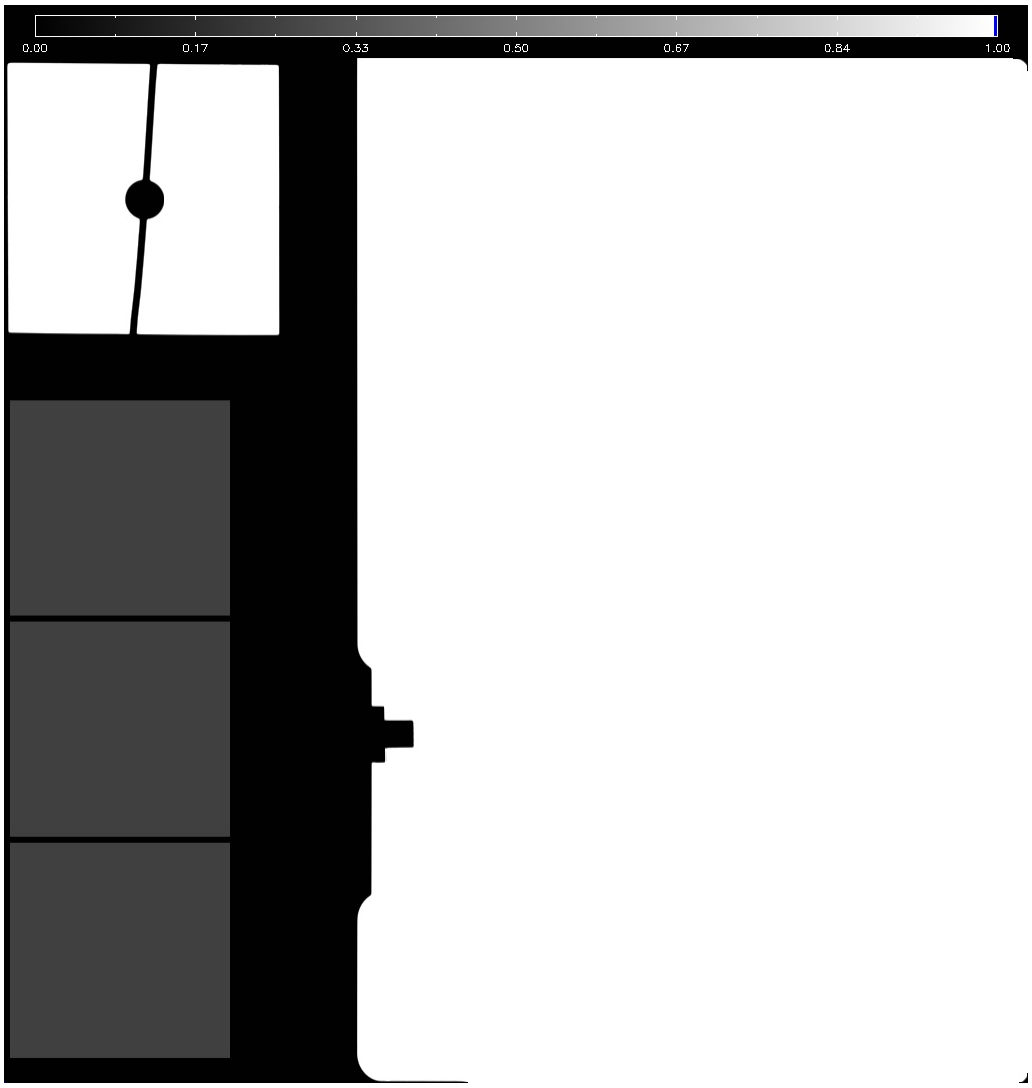
7.5.4: LRS Dispersion

7.6 MIRI-MAISIE

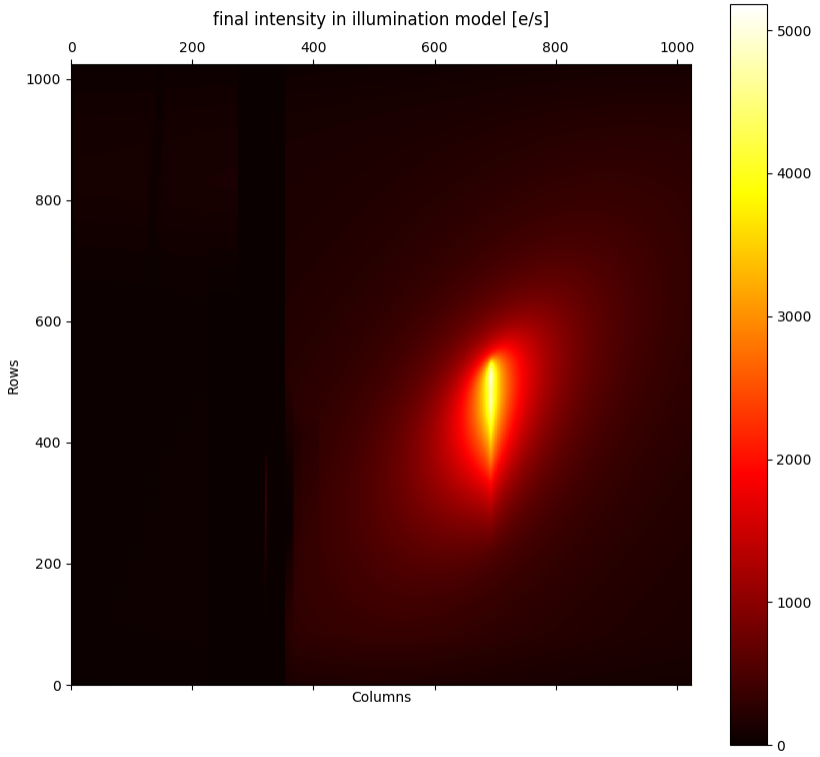
`MIRI-MAISIE`, a MIRI specific variant of MAISIE [O'Brien et al. 2016] simulates the optical path and image slicing through the MRS. It takes input from the orchestration module `ObsSim` and applies transmission and distortion coefficients in line with the sensitivity models of the MRS (see [Glasse et al. 2015]). It then disperses the target data, and produces illumination maps based on user inputs as to which channels and sub-channels are required (as specified in the `simulation.ini` file). The dispersed data are then passed to `SCASim` to simulate the sensor chip assembly through `ObsSim`. These final images are then ready for JWST pipeline processing.

The medium resolution spectrograph (MRS) simulator for the Mid Infrared Instrument (MIRI) of the JWST is the first instrument simulator to be developed using MAISIE. MIRI-MAISIE can simulate all four Integrated Field Unit (IFU) channels, with each channel having three sub-bands. The MIRI MRS simulation accepts a SkyCube object and a data object read from a file. It then applies the relevant combined transmission values and a general transmission value. The main component of MIRI-MAISIE is contained within the IFU detector transformation Effector. The resulting illumination map is combined with data from the paired channel and, finally, a MIRI Illumination Model is created. MIRI-MAISIE treats MIRI as if it had 12 distinct light paths and is mostly constructed from the built in classes from MAISIE. The new classes include a setupData object and a simulation file, which all new simulators need. A new reader class to allow access to the latest MIRI Calibration Data Products and new detector transformation class.

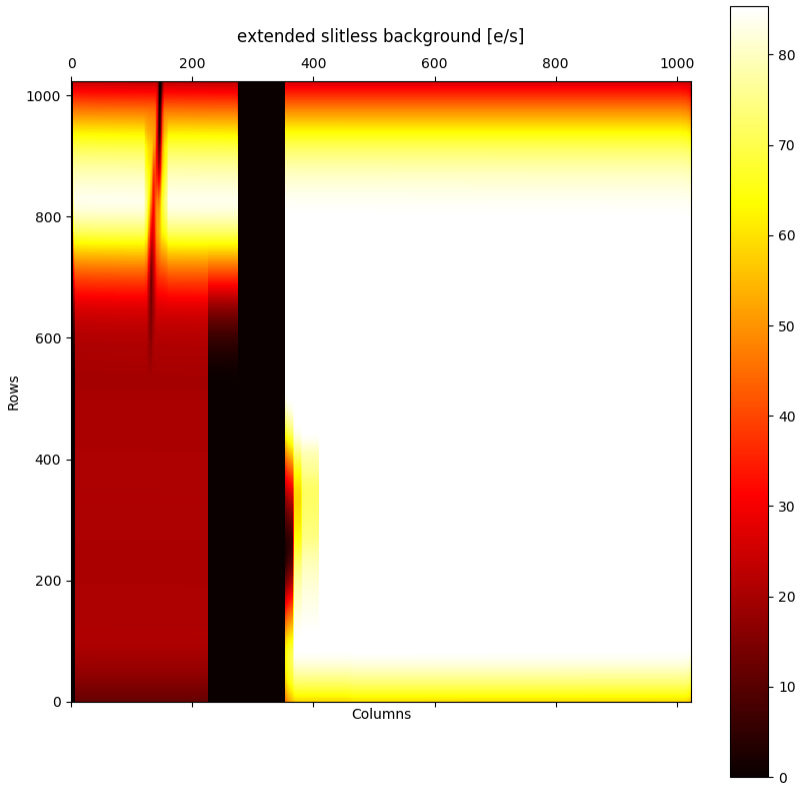
This section highlights how `MIRI-MAISIE` processes the incoming target information; what transformations are made to the data and what the outputs of the simulator are.



7.5.5: Focal Plane Mask



7.5.6: LRS illumination model of an offset galaxy and a centred point source



7.5.7: LRS illumination model of the background

7.6.1 Effectors

The key components of MIRI-MAISIE are 'effectors', which are self-contained sub-routines that cause some form of effect to be performed on the sky cube being manipulated; whether simulating the effects of the telescope background emission, the instrument itself, or any other manipulation required to the data. These effectors can be called in isolation, but generally work in a sequence based on an order of operations. Within MIRI-MAISIE the effectors are split into two types; Instrument effectors and Detector effectors. Which category an effector falls into is determined by whether the data from the CDPs is better interpreted in sky coordinates or detector coordinates and hence, whether it should be used before or after the transformation from sky space to detector space.

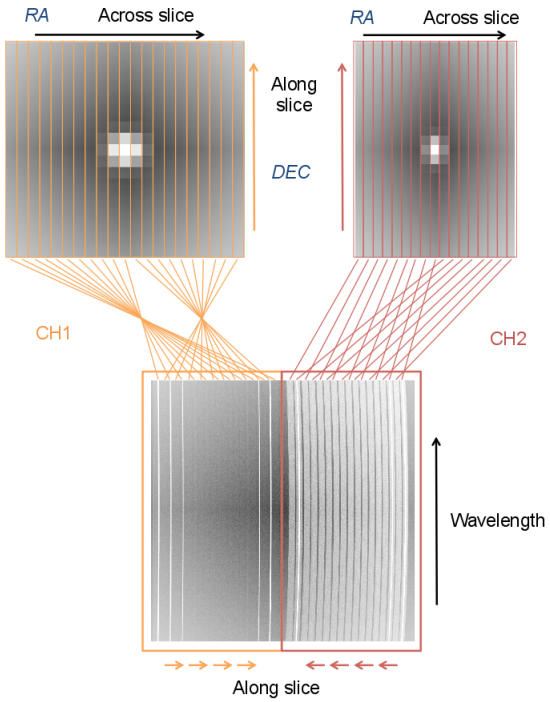
MIRI-MAISIE uses CDP files to access the data needed to simulate the MRS light path, including the MRS distortion, photometry (PHOTOM) and gain files. For a description of these files, see the section on *What MIRISim does to the input data*. The exception to this is rule is the current contamination of the telescope, which is based on the user configuration.

ObsSim provides MIRI-MAISIE with the scene provided in the form of a data cube, information on the channel and band being simulated, and the current level of contamination. MIRI-MAISIE then simulates the MRS light path for each channel individually, combining the two outputs into a single detector image. The first effect simulated is the contamination, a single flat number multiplied across the data cube. The data cubes are then converted from photons/sec/pixel to mJy/pixel before the conversion from sky coordinates to detector coordinates. The distortion CDP provides polynomial coefficients for each slice in a given channel, which transform the coordinates. Carefully chosen Beta coordinates in ObsSim means that there is one sky slice per detector slice. With the new coordinates the flux from the sky spaxels is resampled onto the detector. A slice mask makes sure that no flux goes into a detector pixel that should be dark and protects from an oversized data cube. With the scene in detector space, the PHOTOM CDP can be applied. This takes the form of a detector image with the response of each pixel, and hence each wavelength. The image is multiplied with the detector illumination map. At this stage the illumination model has units of DN/sec/pixel, to bring the model to the correct units for SCASim the Gain CDP is used. The Gain CDP is another detector image, which is also multiplied with the illumination model. Finally the illumination model has the units e/sec/pixel, and the illumination models for the two channels are added together. The model can now be passed to SCASim, with Quantum Efficiency turned off as that information was included in the PHOTOM CDP.

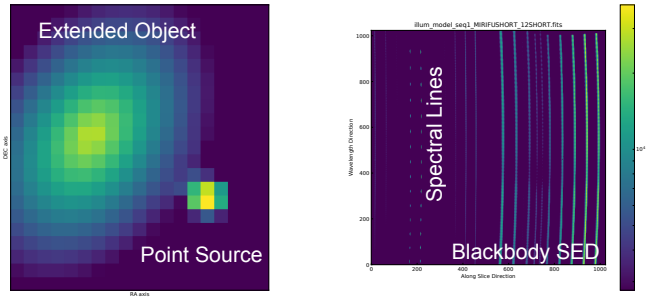
7.6.2 Outputs

MIRI-MAISIE creates illumination maps which are then saved by ObsSim as FITS files. These files can then be read into SCASim. There will be one illumination map per detector pair (i.e. 'Short' or 'Long' wavelength observations in channels 1 & 2 or channels 3 & 4) and sub-channel (i.e. 'A','B','C'). Figure 7.6.1 shows the form of the final illumination map (bottom panels) and how the dispersed data corresponds to the original input cubes. Horizontal lines (or bands for broad spectral features) in the output images correspond to spectral lines.

Figure 7.6.2 shows the point and extended source examples of Appendix Sample_MRS_Simulation, in which the left panel shows a slice of the skycube, and the right panel shows the illumination model produced by MIRI-MAISIE. Channel 1 contains spectral lines which were attached to the point source, and channel 2 shows the continuous black-body spectrum of the extended object.



7.6.1: Overview of the mapping between the image plane and an illumination map.



7.6.2: *Left:* Single slice of an MRS skycube showing the positions of the extended (and truncated) disk and point source in the MRS Field of View. *Right:* Illumination model produced by the simulated objects in the skycube. Channel 1SHORT is shown on the left, and Channel 2SHORT is shown on the right.

7.7 ImSim

The ImSim module directly takes the outputs from SkySim and simulates defocussing, jitter, transmission, and dispersion through the optics. No attempt is made at dispersion of the incoming signal.

Note: MIRISim does not simulate coronography.

7.7.1 Reference coordinates

The coordinates of targets placed in scenes (and indeed, centres of input FITS cubes) are referenced to the centre of the imager array. This varies depending on which type of imager observations are being made (e.g. whether a sub-array is being used). The pixel references for the centering are presented below, with the first two columns representing the array start location, the second two columns represent the size of the array being used (i.e. the FOV in pixels), and the last two columns represent the center pixels of the FOV:

```
'FULL': (0, 0, 1024, 1024, 688.5, 511.5)
'MASK1065': (0, 18, 284, 224, 115.0, 131.0)
'MASK1140': (0, 244, 284, 224, 114.5, 353.5)
'MASK1550': (0, 466, 284, 224, 114.5, 574.0)
'MASKLYOT': (0, 716, 316, 304, 137.5, 883.5)
'BRIGHTSKY': (456, 50, 512, 512, 707.5, 305.5)
'SUB128': (0, 888, 132, 128, 65.5, 951.5)
'SUB256': (412, 50, 256, 256, 535.5, 177.5)
'SUB64': (0, 778, 68, 64, 33.5, 809.5)
'SLITLESSPRISM': (0, 528, 68, 416, 33.5, 828.0)
```

In the simulation.ini file, where to center the FOV is defined in the 'ConfigPath' variable, while which sub-array to use is defined in the 'ReadDetect' variable.

7.8 SCASim

Both the MIRI imager and spectrometer use the same design of Sensor Chip Assembly (SCA) and the same kind of detector chip. The MIRI Sensor Chip Assembly Simulator (SCASim) provides a common utility which can be used by all MIRI simulator modules. The end result of the instrument components of MIRISim is a description of how the instrument is illuminating its detector, which is saved to a FITS file in a standard format common to all simulators. The SCA simulator reads that FITS file and finishes the simulation by adding the detector effects.

The outputs of ImSim, LRSSim and MIRI-MAISIE are models of how light is presented to the detectors. These illumination models are processed by SCASim to add detector effects to those models. The illumination models are presnted with the right 'Data Number' (DN) formatting (equivalent flux levels) so that SCASim can build up realistic observation ramps based on the number of frames, integration and exposures required for the simulation.

- Poisson (shot) noise
- Dead and hot pixels
- Detector persistency and latency
- Zero-point drift
- Detector non-linearity
- Dark current

- Quantum efficiency as a function of wavelength
- Read noise as a function of temperature
- Reference pixels
- Amplifier bias and electronic drift
- Amplifier gain and non-linearity
- MULTIACCUM detector readout modes
- MIRI Subarray modes
- Cosmic ray hits on the detector

Using libraries and models derived specifically for MIRI, or from MIRI testing. The work flow through SCASim can be seen in Figure 7.8.1.

The efficiency with which the detector converts photons into electrons is assumed to be described by the Quantum Efficiency (QE), which can vary between 0.0 and 1.0 and is wavelength dependent. The QE function is taken from measurements made during flight model testing. Some detector pixels might not be functional or might have an abnormally low QE. These pixels may be identified using a bad pixel map containing a 0 to indicate good pixels and 1 to indicate bad ones. `SCASim` simulates these bad pixels by giving them a QS of zero which makes them insensitive to light.

The detector dark current is added ot the flux illuminating the detectors. If the photon flux is zero, the dark current will result in the build up of a small number of electrons with time. Similarly, `SCASim` keeps a tab of the expected electron count as it builds up during an integration, and adds Poisson noise to the read out (by sampling from a poisson distribution with an expectation value equal to the expected electron count).

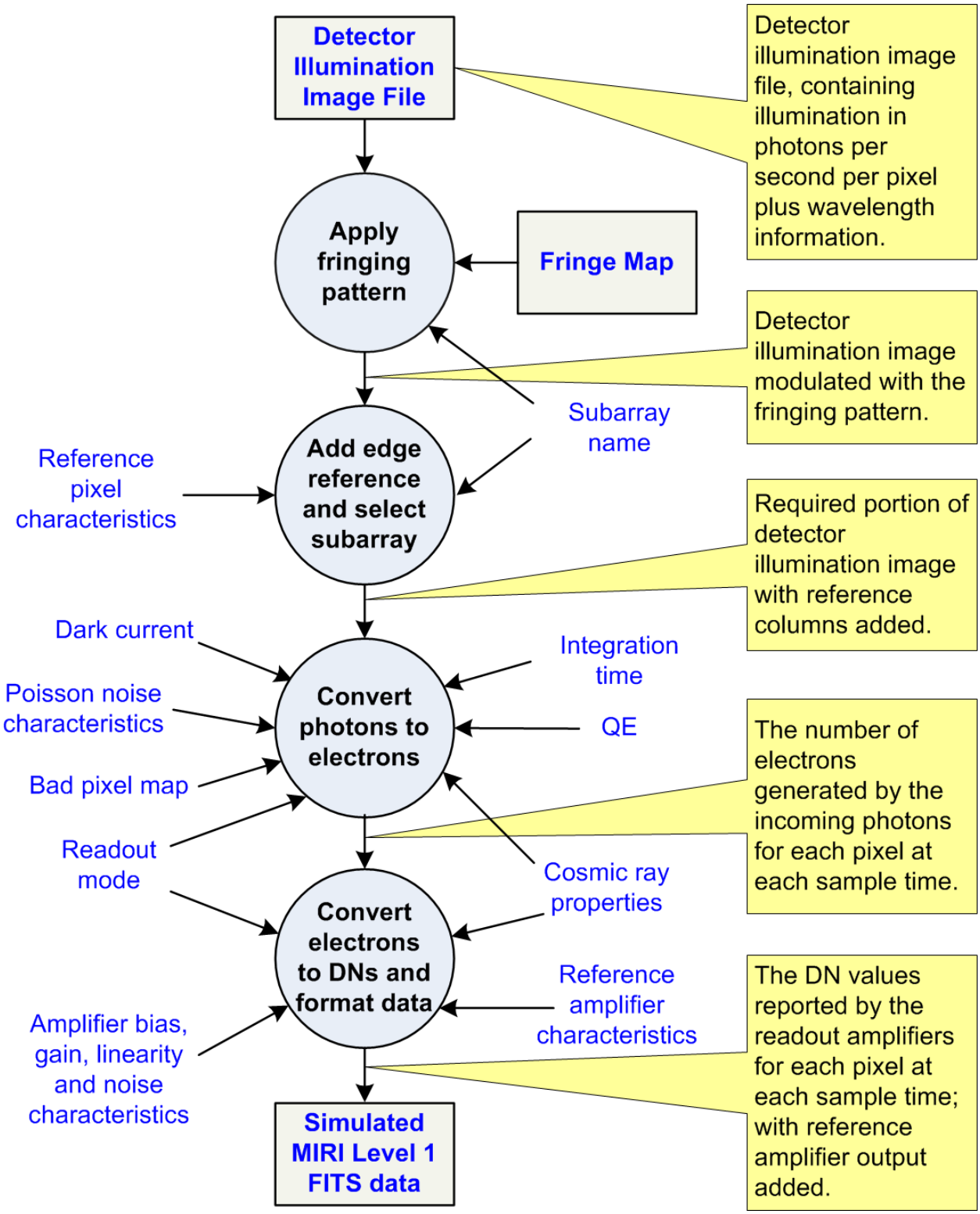
The sensitivity of each amplifier, i.e. how quickly the output data number changes when the input electron count changes, is determined by the amplifier gain. The conversion function between electron count and data number also deviates from linearity at high counts.

The simulation of detector latency and drift uses a charge trapping and release model derived from ground-based measurements made in FAST mode. The model is more accurate at a timescale of around 300s, but at much longer or much shorter timescales our model is an approximation. Latent and drift effects are turned off in SLOW mode, as we don't yet have reliable measurements in that mode. The latent and drift effects seen in flight will depend on the complete history of the illumination incident on the MIRI detectors, whereas the MIRI simulator begins with a "zero trapped charge" initial condition. The latent decay during wait periods between observations and dither maneuverer is currently not simulated correctly, as the detector is assumed not to be illuminated, and the wait time interval is not the same as in FAST mode.

Detector nonlinearity is simulated by applying an inverse of the function defined in the nonlinearity CDP. This effect should cancel out when the same CDP is used by the JWST Pipeline to correct the nonlinearity. Our tests show the two algorithms mostly cancel out, but there are two residual effects we are still investigating:

- In MRS simulations, there is a slight difference remaining in 3B, 4A, 4B and 4C aperture spectra but these are within 5% up to 4A and less than 10% in 4B and half-way into 4C.
- The continuum level in these MRS channels is higher than expected by about 10%. This may be related to the other flux issues mentioned above.

The detector bias level (i.e. minimum DN) simulated by SCASim is arbitrary. Turning a simulated effect on or off can change the bias level (since many effects are applied by adding them). The bias level may appear to be different for FAST and SLOW mode, but this is only because zero-point drift is simulated in FAST mode only. This difference does not affect the final result, since the relative change in DN will be the same.



7.8.1: Flow of information through SCASim

CHAPTER

EIGHT

GLOSSARY

'Sky' Coordinates RA, DEC, [wavelength]. The RA and DEC we refer here to are just orthogonal coordinates in degrees, they do not mean actual RA and DEC (this may change, if needed, but most probably there won't be such a need).

Channel There are four channels in the MIRI MRS (1,2,3 and 4), each of which is separated into three sub-channels (short, medium and long, or A,B and C) corresponding to coverages of different wavelength ranges. The wavelength coverages of the channel and sub-channel combinations is given in Table 3 of Reike et al. (2015, MIRI PASP Paper 1).

Flux Radiation energy per unit time (power) and unit area (Units: W m^{-2} $arcsec^{-2}$).

IFU Integral Field Unit. This kind of spectrograph is able to capture both spectral as well as positional information about the target, resulting in a spectral cube of data with two position and one wavelength axis.

JWST James Webb Space Telescope

JWST-FOV coordinates V2, V3, [wavelength]

LOSVD Line-of-sight Velocity Distribution. Distribution function of velocities along the line of sight. It has the effect of broadening along wavelength features in the spectrum of a source.

MIRI Mid-Infrared Instrument

MRS Mid-Resolution Spectrograph

Principle Optical Path The path along which light travels through the instrument. In the case of simultaneous observations, there could also be a *secondary* optical path, which would depend on the pointing of the principle optical path, and the rotation of the telescope/instrument.

SED Spectral Energy Distribution.

Specific Intensity Radiation power per unit area and unit solid angle (Units: W m^{-2} $arcsec^{-2}$).

Velocity Map Spatial distribution of velocities along the line of sight of an extended source.

CHAPTER

NINE

APPENDIX: BUILDING SCENES WITHIN MIRISIM

There are two fundamental ways of specifying the target *scene' to simulate with |MIRISim|: build up a `patch of sky* scene from the building blocks provided in SkySim (point and extended objects which can be placed in the MIRI FOV), or from pre-defined FITS data cubes (3D data are required).

Here, we give a brief overview of both options, noting that they are not mutually exclusive: Skysim targets can be superimposed on a FITS file defined scene.

9.1 Using a FITS Cube or Image

Often, the astronomical scene to be simulated will be pre-determined. For this case `MIRISim` allows a properly formatted FITS file to be read in. Input scene `FITS` files require a well formatted header structure with `crval`, `crpix`, `cdelt`, `ctype`, and `cunit`, etc. values populated for each of the axes (for 2D or 3D images/cubes). The `FITS` images/cubes are read in by `astropy`, and therefore must comply with their formatting constraints. `MIRISim` works in a coordinate system relative to the centre of the detector being used. This means that SkySim interprets the absolute positions in the header (e.g. RA and DEC in the `crval` keywords), and converts them to the header into a relative coordinate system based on the centre of the `FITS` file. In other words, internally, SkySim re-grids the input FITS file to be centred at (0,0). This allows `FITS` file inputs to be combined with additional sources in the field of view through specifying them in a scene.

An example 3D FITS file is provided on the website, as is a Jupyter notebook which shows how to convert a 2D image into a 3D cube for ingest into `MIRISim`. 3D data are required even for imager simulations because SkySim needs to know the wavelength binning of the data in order to get the flux conservations right across the filter passband.

To import a `FITS` cube using a `scene.ini` file, the portion of the input file relevant for importing the `FITS` file consists of:

```
[Input FITS]
    Type = SkyCube
    cubefits = 'scene.fits'                  # name of input fits file
    fits_ext = 1                                 # optional keyword to specify fits␣
↪extension (default = 0)
```

With the assumption that the `FITS` file is in the current directory. The optional `fits_ext` keyword specifies which `FITS` extension to look in for the data. The default is extension 0.

To import a `FITS` cube within python:

```
from mirisim.skysim import externalsources

externalsources.Skycube(FITSCUBE)
```

or, alternatively:

```
from mirisim.skysim import Skycube

s = Skycube(FITSCUBE)
```

There are a few different interpolation methods built into MIRISim for reading in the input Skycubes. They trade off between computational efficiency and better flux conservation. The three key methods are:

9.1.1: Skycube interpolation methods

Method Number	Interpolation Method
0	'Primitive' interpolation (no rebinning)
1	Rebinning the full cube one dimension at a time
2	Rebinning per 'column' in the data cube

The default interpolation method is method=1. Any other interpolation method must be specified when instantiating the skycube (i.e. s = Skycube(FITSCUBE,method=0) as a modification to the command above).

9.2 Building a scene from component targets

9.2.1 Types of Targets: Background Specification

There are a number of contributors to the background noise seen by MIRI, with significant contributions coming from the thermal emission of JWST itself, Zodiacal light, etc. These levels vary depending on the angle of the telescope, and so therefore can be set in an astronomical scene. The overall level can be chosen between LOW and HIGH, with background sources constrained from the MIRI sensitivity model (see [MIRI9]).

In a scene file, the background is specified as:

```
[Background]
    level      = LOW          # set overall background level
    gradient   = 5.           #% over 1 arcmin
    pa         = 15.          #position angle of gradient
    centreFOV  = 0. 0.        #centre of FOV
```

With an allowed gradient across the field of view (centred at centerFOV), and a specified position angle.

In python, the background is specified as:

```
from mirisim.skysim import Background

bg = Background(level = 'low',gradient = 5., pa = 15.)
```

9.2.2 Types of Targets: Spatial Specifications

The spatial specifications of MIRISim targets come in two varieties: point and extended objects. For extended objects, the geometries are limited to elliptical (e.g. disk like) morphologies with differing ways of specifying the flux fall off with radius (e.g. Sersic or Gaussian profiles). Disk truncation radii can be set for these extended objects, but spiral arms cannot. For these, and other such complex structures, we suggest the user specifies their disk in another package of their choice (e.g. the outputs of a radiative transfer simulation), and import the resulting FITS file directly into MIRISim.

In a scene file, point sources are initialised with a location:

```
[Point_Source_name]
    Type        = Point        #Type of target
    Cen         = 0.1 0.5      #Where to place the target (arcsec)
```

In python:

```
from mirisim.skysim import Point
pt_src = Point(Cen = (0.1,0.5))
```

While extended sources have the added parameterisation of their spatial extents. An example of how to specify an elliptical galaxy:

```
[Galaxy]
    Type        = Galaxy    #Type of target
    Cen         = 0. 0.     #RA,DEC (offset, specified in arcseconds)
    n           = 2.        #Sersic index of the Galaxy
    re          = 0.5       #Effective radius (arcsec)
    q           = 0.5       #Axial ratio
    pa          = 35.       #position angle (deg)
```

Or, alternatively in python:

```
from mirisim.skysim import Galaxy
ext_src = Galaxy(Cen = (0.,0.), n = 2., re = 0.5, q = 0.5, pa = 35.)
```

A protostellar disk (with an inner hole) can be specified as:

```
[Protostellar Disk]
        Type    = Sersic Disk    # Type of target
        Cen     =  0. 0.  # RA,DEC (offset, specified in  arcsec)
        n         = 1  # exponential disk
        pa       = 90 # position angle of the disk
        q         = 0.7 # axial ratio of the disk
re = 18,           # scale length of disk (in arcsec for exp. decay)
rtrunc_in = 1.,     # where to truncate the inner rad. (in arcsec)
rtrunc_out = 3.25 # where to truncate the outer rad. (in arcsec)
```

or, alternatively in python

> from mirisim.skysim import SersicDisk PD = SersicDisk(Cen = (0.,0.), pa = 90, q = 0.7, n = 1, re = 18, rtrunc_in = 1., rtrunc_out = 3.25)

9.2.3 Types of Targets: Spectral Specifications

Once the spatial extent and spectral component of a target have both been set, a velocity map can be attributed to the target, which will modify its spectral specifications across the object. The velocity fields across the spatially extended object can be modified according to rotation curves. An example of this, specified in a scene file after the SED for a target, is shown below:

```
[[velomap]]
          Type        = FlatDisk     #Type of Velocity map to initialise
          Cen         = 0. 0.        #Specify the centre of the disk
          vrot        = 200.         #Rotational Velocity (km/s)
          pa          = 35.          #Position angle of the velocity map (deg)
          q           = 0.5          #Axial ratio of major and minor axes
          c           = 0.           #measure of diskiness/boxiness
```

The spectral components of targets can be specified in a variety of ways, as described below, highlighting the general categories of smooth SEDs, spectral lines, and external formats, either from pysynphot, or a user supplied SED file listing two columns of wavelength and flux.

Spectral Energy Distribution

The key types of manually constructed SEDs in `SkySim` are blackbodies, and power law spectra. These simple continuous SEDs take reference wavelengths and fluxes as inputs as well as reference temperatures and power-law indicies (respectively) as described below. For the specification required for the other (more complicated) SEDs, download and run the jupyter notebook linked above to see the required inputs.

For black body sources, `MIRISim` uses an analytical Planck black-body spectrum. It takes 3 parameters:

- Temp: black body temperature [K]
- flux: reference flux [μJy] at the reference wavelength
- wref: Reference wavelength [micron] where the flux is equal to the reference flux

Plus, optional velocity (v, in km/s) or redshift (z) parameters.

The output spectrum is in μJy. In a scene file, this is specified as:

```
[[SED]]
    Type = BB      # Blackbody SED
    Temp = 1000  # Temperature of source
     wref = 10        # reference wavelength (micron)
     flux = 1e3      # reference flux at wref (microJy)
     v = 20           # Velocity shift (in km/s)  [optional]
     z = 0.4          # Redshift [optional]
```

In python, this is minimally specified as:

```
from mirisim.skysim.sed import BBSed
BB = BBSed(Temp = 1000., wref = 10., flux = 1e3)
```

For a power-law SED with an analytical spectrum defined as $F_nu(\lambda) = f_0(\lambda/\lambda_{ref})^{\alpha}$, the power-law object takes a minimum of 3 parameters:

- alpha: power-law index (alpha = 0 flat spectrum as a function of wavelength)
- flux: reference flux [μJy] at the reference wavelength
- wref: Reference wavelength [micron] where the flux is equal to the reference flux

With output spectrum in μJy, and the v and z parameters can also be optionally set to modify the spectrum. In the scene file, this is specified as:

```
[[SED]]
    Type = PL      #Powerlaw SED
    alpha = 1.      # power law index
    flux = 1e3     # reference flux at wref (microJy)
    wref = 10.     # reference wavelength (micron)
```

Or, in python:

```
from mirisim.skysim.sed import PLSed
PL = PLSed(alpha = 1., flux = 1e3, wref = 10.)
```

Because these types of SEDs are manually constructed within SkySim, they do not, by default have the SED.Lam and SED.fLam properties that user supplied spectra do. This is because no wavelength sampling was given as an input. However, the SED can be plotted by calling the SED class. An example is provided below for a blackbody SED (called SED) attached to a point source called star. The SED is being sampled by 500 points over the wavelength range between 5 and 25 micron:

```
WAV = np.linspace(5,25,500)
FLUX = star.SED(WAV)
```

In this example, the newly created WAV and FLUX variables stand in for the SED.Lam and SED.fLam properties available in other types of SEDs.

Spectral Lines

Users can also define a set of spectral lines. In this case, as SkySim expects the user to input integrated intensities for each of the lines:

```
[[SED]]
    Type = Lines
    wavels      = 5.7 6.6 8.7 10.1 13.4 15.5 20.6 24.0
    fluxes      = 1.E4 1.E4 1.E4 1.E4 1.E4 1.E4 1.E4 1.E4  #Integrated fluxes [10^-20␣
↪W/m^2] of each line
    fwhms       = 1.E-3 1.E-3 1.E-3 1.E-3 1.E-3 1.E-3 1.E-3 1.E-3  #FWHM
```

or, in python:

```
from mirisim.skysim.sed import LinesSed
lines_sed = LinesSed(wavels = [5.6,6.6], fluxes = [1.e4,1.e4], fwhms = [1e-3,1e-3])
```

When specifying spectral lines as above, or using external inputs to generated a spectral energy distribution (below), SkySim uses the input wavelengths as the edge of its bin. Therefore there are instances where lines will appear consistently shifted in wavelength by half a bin. This can be minimised by better spectral sampling of lines.

External SEDs

A third option for input spectral energy distributions, is for SkySim to read the contents of an external SED file which is a text file consisting of two columns: the first has wavelengths (in microns), the second, flux densiti1es (in microJy). This table must be compatible with astropy.io.ascii table formatting:

```
[[SED]]
    Type       = External    # use external SED file
    sedfile    = sedext.txt # name of input file
```

or, in python:

```
from mirisim.skysim import ExternalSed
ext_sed = ExternalSed(sedfile = 'sedext.txt')
```

Using PySynPhot

The ability to use pysynphot has been implemented in MIRISim for those familiar with it. Please see the documentation on the pysynphot website for a listing of the available models, the wavelength ranges covered by those models (most of which do not extend to the MIRI wavelengths), and expected input units for each model.

```
[[sed]]
    Type      = pysynphot
    family    = # part of the pysynphot description of a template
    subfamily = # part of the pysynphot description of a template
    sedname   = # part of the pysynphot description of a template
    wref      = # reference wavelength for flux calibration
    flux      = # reference flux at wref for flux calibration
```

or, in python:

```
from mirisim.skysim import PYSPSed
spe = PYSPSed(family='bc95', sedname='bc95_b_10E6', wref=10., flux=600.)
```

Associating an SED with a target

In the configuration file, an SED is associated (or attached to) a target by being specified directly after the spatial specification of the target, and indented one level from it.

In python, an SED needs to be explicitly attached to a spatially defined object. In the case of attaching the blackbody SED described above (as BB) to the point source defined as pt_src:

```
pt_src.set_SED(BB)
```

CHAPTER

TEN

APPENDIX: CREATING A SCENE IN PYTHON

A pre-defined sky scene can be injested into MIRISim, although the flexibility exists for the user to create their own custom scene. This notebook shows the steps required to create a viable (albeit simple) scene for use within mirisim. The equivalent code placed in a .ini file is given in parallel with the python definitions.

This example creates a scene with two objects: a star, and an extended galaxy. Both have imported spectral energy distributions (SEDs) and a few additional spectral lines overlaid.

[1]:
```
## skysim is the module within MIRISim that creates scenes.

from mirisim.skysim import Background, sed, kinetics,Galaxy, Point, Skycube
from mirisim.skysim import wrap_pysynphot as wS
```

10.1 setup the scene

There is a short preamble required for a .ini file to be executable. There is no python equivalent, so it is mentioned here briefly before delving into the python aspects of creating a scene

```
[sky]
    name = scene1
    loglevel = 1
```

10.1.1 Creating a Galaxy

The first target to be created in this simple sky scene is a galaxy. The steps to creating a complete galaxy include:

1. setting up the spatial extent of the galaxy
2. creating an SED to apply to the extended emission
3. Add the SED to the Galaxy
4. add a line of sight velocity distribution to the SED

Step 1: Setup the spatial extent of the galaxy

The galaxy created within python below has the same properties as this parameter grouping expressed in a .ini file:

```
::
```

[Galaxy] Type = Galaxy # Type of target Cen = -2. 2. # RA,DEC (offset, specified in arcseconds) n = 1. # Sersic index of the Galaxy re = 0.1 # Effective radius (arcsec) q = 0.5 # Axial ratio pa = 15. # position angle (deg)

[2]:
```
galaxy = Galaxy(Cen=(-2.,2.),n=1.,re=0.1,q=0.5,pa=15.)
```

```
2020-08-04 14:54:20,864 - INFO - Initializing Galaxy
2020-08-04 14:54:20,864 - INFO - Initializing Galaxy
2020-08-04 14:54:20,865 - INFO - Initializing Galaxy
```

Step 2: Create a spectral energy distribution for the Galaxy

Here we show an example of importing a pysynphot SED. The parameters required as input to a pysynphot SED can be found on the pysynphot website. note that the `sedname` must be consistent with one provided in the pysynphot directories, the location of which was set when mirisim was installed. To find it:

```
echo $PYSYN_CDBS
```

The .ini file equivalent of creating this pysynphot SED in python is as follows. Note that when placed in a .ini file, this set of properties can be indented for clarity.

```
[[sed]]
    Type        = pysynphot             # specify SED from pySynPhot librarires
    family      = bkmodels              # SED Family from which to draw.
    sedname     = bk_b0005              # Template within calatlogue to be used
    wref        = 10.                   # Reference wavelength for scaling (in micron)
    flux        = 1E+3                  # Reference flux for scaling (in microJy)
```

The 'bkmodels' are the Buser-Kurucz stellar atmosphere models, more details of which can be found at this link. This model were chosen for this example because the SED extends to 20 micron. Most of the SEDs in the pysynphot repository do not reach MIRI wavelengths.

[3]:
```
# specify which of the pysynphot models to use (in a dictionary)
PYSPsedDict = {'family':'bkmodels','sedname':'bk_b0004','flux':1E+3,'wref':10.}
# read that dictionary into the pysynphot interpreter
sedE = wS.PYSPSed(**PYSPsedDict)
```

Step 3: Add the SED to the galaxy object

The SED component can now be added to the properties of the galaxy created in Step 1. This step has no equivalent in the .ini file, as this is done automatically by placing the SEDs underneath the instance of `[Galaxy]`

[4]:
```
# link the SED to the galaxy object
galaxy.set_SED(sedE)
```

Step 4: add a velocity map to the Galaxy

The template galaxy created here is expected to be rotating, which will shift the SED as a function of position within the galaxy. This is accounted for by applying a velocity map to the galaxy object. The .ini file equivalent of the python code below is:

```
[[velomap]]
      Type        = FlatDisk          # Type of Velocity map to initialise
      Cen         = -2. 2.            # Specify the centre of the disk
      vrot        = 200.              # Rotational Velocity (km/s)
      pa          = 15.               # Position angle of the velocity map (deg)
      q           = 0.5               # Axial ratio of major and minor axes
      c           = 0.                # measure of diskiness/boxiness
```

Here, we've created a flattened galatic disk. Note that many of the parameters set here are identical to those set when initialising the galaxy at the beginning.

[5]:
```
# Create a dictionary of parameters
VMAPpars = {'vrot':200.,'Cen':(-2.,2.),'pa':15.,'q':0.5,'c':0}
# create an instance of a flattened disk
velomap = kinetics.FlatDisk(**VMAPpars)


# apply the velocity mapping to the galaxy

galaxy.set_velomap(velomap)
```

```
2020-08-04 14:54:20,934 - INFO - Initializing FlatDisk
```

10.1.2 Creating a "Star"

Because a star is a point source, the inputs are a little simpler. An SED can (and should) still be specified (for this example a star is a simple blackbody), however there is no point in specifying a velocity distribution since all of the flux is coming from a single point.

The steps involved in creating a star here include:

1. initialising a point source
2. adding a blackbody spectrum

The equivalent in a .ini file would entail:

```
[Star]
    Type = Point
    Cen = 0. 0.
    vel = 0.0

    [[sed]]
        Type = BB
        Temp = 1e4
        flux = 100.
        wref = 10.
```

[6]:
```
## 1. initialise the point source spatial position
star = Point(Cen=(0.,0.))
## 2a. create a dictionary with the required parameters for a blackbody
```

(continues on next page)

(continued from previous page)

```
BBparams = {'Temp':1e4,'wref':10.,'flux':100.}
## 2b. create the blackbody spectrum
BlackBody = sed.BBSed(**BBparams)
## 2c. add that spectrum to the 'star'
star.set_SED(BlackBody)
```

```
2020-08-04 14:54:20,940 - INFO - Initializing Point
```

10.1.3 Creating a Scene from a star and a Galaxy

The two components of the scene can now simply be added together to form a scene. To this two component scene, we will also (below) add some background radiation. The equivalent .ini file syntax for adding the background is:

```
[Background]
    level = low
    gradient = 5.
    pa = 45.
    centreFOV = 0 0
```

After completing the scene, a text description can be output to the screen using the print function

[7]:
```
# create a background object
bg = Background(level='low',gradient=5.,pa=45.)

# sum the background,galaxy and star
scene = bg + galaxy + star

## print the text version to the screen
#print(scene)
```

```
2020-08-04 14:54:20,946 - INFO - Initializing Background
```

10.1.4 Exporting the scene to a FITS file

With the scene created, there's now the oportunity to write it out to a FITS file. To do this requires additional specifications, including:

```
* Field of View (FOV)
* Spatial Sampling (spatsampling)
* Output Wavelength Range (wrange)
* Wavelength Sampling (wsampling)
* Time (time)
```

Note that the last parameter, time, needs to be specified for time dependent backgrounds only (which are not yet implemented).

[8]:
```
# The field of view and its sampling are specified in arcsecondsy
FOV = np.array([[-3,3],[-4,4]])  # 6x8" field of view
spatsampling = 0.1               # 0.1" spatial sampling (resolution)

# The wavelength range is specified in microns
wrange = [5.,15.]           # the output only covers 5 to 15 microns
wsampling = 0.002           # 0.002 micron wavelength resolution
```

(continues on next page)

(continued from previous page)

```
# Because time dependent backgrounds are not yet implemented,
# 'time' does not need to be set
```

With the additional parameters pertaining to the properties of the output FITS file set, the scene created above can be output to a FITS file

NOTE: The option overwrite = False ensures that any FITS file in the current directory with the suggested output file name exists, it will not be overwritten. To change this, set overwrite = True

[9]:
```
scene.writecube(cubefits = 'scene.FITS',
                FOV = FOV, spatsampling = spatsampling,
                wrange = wrange, wsampling = wsampling,
                overwrite = True, time = 0.0)
```

[ ]:
```
```

CHAPTER

ELEVEN

APPENDIX: CREATING AN IMAGER SCENE

Using the components of Skysim, this notebook generates the scene which is used in the user guide as an example. Please note that the target fluxes used here are exagerated so that they will appear in the single frame (un-calibrated) data produced by the example imager simulation

[1]:
```
# import the types of objects to be placed in the scene

# spatial
from mirisim.skysim import Background, Point, SersicDisk

# spectral
from mirisim.skysim import sed

#import the scene configuration parser
from mirisim.config_parser import SceneConfig

#other imports related to plotting, etc
import numpy as np
import matplotlib.pyplot as plt


scene = []
```

11.1 Setup two 'protoplanetary disks':

Each disk will have the same inner and outer truncation radii, and the same blackbody function defined. The only difference between them will be their positions

[2]:
```
# setup a function to define the disks symmetrically
# (with the position as the only difference)
def setup_PD(cen):

    # setup the spatial part of the disk
    PD = SersicDisk(Cen = cen,         # where to centre the disk
              pa = 90,           # position angle of the min/maj. axes
              q = 0.7,           # axis ratios
              n = 1,             # sersic index (1 = expontential disk)
              re = 18,           # scale length of disk
              rtrunc_in = 1.,     # where to truncate the inner rad.
              rtrunc_out = 3.25 # where to truncate the outer rad.)
              )
```

(continues on next page)

(continued from previous page)

```
    # setup the spectral part of the disk
    BB = sed.BBSed(Temp=300.,       # Reference temperature for the BB
                   flux=1.4e5,      # flux [muJy] at the reference wavelength
                   wref=10.)    # ref wave. for flux scaling

    # attach the SED to the spatial spec. of the disk
    PD.set_SED(BB)

    return PD


PD_one = setup_PD((-8,8))
PD_two = setup_PD((8,8))


scene.append(PD_one)
scene.append(PD_two)
```

```
2020-08-04 14:59:42,644 - INFO - Initializing SersicDisk
2020-08-04 14:59:42,645 - INFO - Initializing SersicDisk
2020-08-04 14:59:42,646 - INFO - Initializing SersicDisk
2020-08-04 14:59:42,646 - INFO - Initializing SersicDisk
```

11.1.1 Setup point sources:

Place a number of point sources in the field of view. To show a different kind of SED, these are set with a power law SED with a power law index of 1.

[3]:
```
def create_star(xpos,ypos):
    star = Point(Cen = (xpos,ypos))

    PLparams = {'alpha': 1,
                'wref': 10.,
                'flux': 120.}
    PowerLaw = sed.PLSed(**PLparams)
    star.set_SED(PowerLaw)

    return star


# create a semi-circle of point sources in the bottom half of the FOVE

Nstars = 35
radius = 15
for i in range(int(Nstars*0.5)+1,Nstars):
    xp = np.cos(2*np.pi/Nstars*i)*radius
    yp = np.sin(2*np.pi/Nstars*i)*radius
    star = create_star(xp,yp)
    scene.append(star)
```

```
2020-08-04 14:59:42,654 - INFO - Initializing Point
2020-08-04 14:59:42,655 - INFO - Initializing Point
2020-08-04 14:59:42,656 - INFO - Initializing Point
2020-08-04 14:59:42,656 - INFO - Initializing Point
```

(continues on next page)

(continued from previous page)

```
2020-08-04 14:59:42,657 - INFO - Initializing Point
2020-08-04 14:59:42,657 - INFO - Initializing Point
2020-08-04 14:59:42,658 - INFO - Initializing Point
2020-08-04 14:59:42,658 - INFO - Initializing Point
2020-08-04 14:59:42,659 - INFO - Initializing Point
2020-08-04 14:59:42,659 - INFO - Initializing Point
2020-08-04 14:59:42,660 - INFO - Initializing Point
2020-08-04 14:59:42,661 - INFO - Initializing Point
2020-08-04 14:59:42,661 - INFO - Initializing Point
2020-08-04 14:59:42,662 - INFO - Initializing Point
2020-08-04 14:59:42,662 - INFO - Initializing Point
2020-08-04 14:59:42,663 - INFO - Initializing Point
2020-08-04 14:59:42,663 - INFO - Initializing Point
```

11.1.2 Write scene to scene file

So far, we have created an object list called 'scene'. This needs to be wrapped into a scene configuration, which is done by the 'SceneConfig' module.

Here we also add the telescope background (set to low) to the scene configuration.

It is this object (scene_config) that could be passed to a python instance of running MIRISim.

[4]:
```
# generate the scene using the scene configuration parser
# (adding in a telescope background)
scene_config = SceneConfig.makeScene(loglevel=0,
                                     background =  Background(level='low',gradient=0,
↪pa=0),
                                     targets = scene)

# name of output scene file
output_scene = 'Imager_scene.ini'

# remove previous version of output INI file (if it exists)
try:
    os.remove(output_scene)
except OSError:
    pass


# create output scene file

scene_config.write(output_scene)
```

```
2020-08-04 14:59:42,668 - INFO - Initializing Background
```

11.1.3 Optional write to FITS

If you want to output the created scene to a FITS file, this is how to create a (small) field of view image centered at 10 microns (wrange)

[5]:
```
output_to_fits = False

if output_to_fits == True:

    FOV = np.array([[-20,20],[-20,20]])  # 6x8" field of view
    spatsampling = 0.1                # 0.1" spatial sampling (resolution)

    # The wavelength range is specified in microns
    wrange = [9.5,10.5]           # the output only covers 5 to 15 microns
    wsampling = 0.2            # 0.002 micron wavelength resolution


    scene.writecube(cubefits = 'scene.FITS',
                    FOV = FOV, spatsampling = spatsampling,
                    wrange = wrange, wsampling = wsampling,
                    overwrite=True,time = 0.0)
```

CHAPTER

TWELVE

APENDIX: MRS SIMULATION SHOWING POINT AND EXTENDED OBJECTS

The simulation captured below is used to make the figures at the end of the MIRI-MAISIE section of the user guide. Here there is an extended object with a pure blackbody SED, and a point source with equally spaced (in wavelength) spectral lines of the same intensity. This will show how the two different kinds of objects in a scene are projected by MIRI-MAISIE onto the detector.

[1]:
```
# import the types of objects to be created
from mirisim.skysim import Point, SersicDisk, sed, Background

# import the simulation configuration parsers
from mirisim.config_parser import SimConfig,SceneConfig,SimulatorConfig

# import the miri simulator
from mirisim import MiriSimulation


#other things to be used
import numpy as np
import glob                  # glob is used to find the output directory
import os                    # for listing directory contents
from astropy.io import fits # for reading FITS file contents

import matplotlib.pyplot as plt    # to display images
from matplotlib import colors,cm
%matplotlib inline
```

12.1 Create the scene to be simulated

[2]:
```
# create an extended disk with a blackbody SED
ex_disk = SersicDisk(Cen=(-1,0),n = 1, re = 1.3,q = 0.7,
                     pa = 45.,rtrunc_out=2.3)
BB = sed.BBSed(Temp = 400., flux = 1e7, wref = 4.)

ex_disk.set_SED(BB)

# create a point source with a set of lines in MIRI ch 1a and 2a

pt = Point(Cen = (1,0.5))

Nlines = 15   # number of spectral lines to create per channel
```

(continues on next page)

(continued from previous page)

```
lines_1A = np.linspace(4.9, 5.7, Nlines)
#lines_2A = np.linspace(7.5, 8.9, Nlines)

# only put lines in channel 1, so they're distinguishable
# from the BB spectrum of the extended object (in ch 2)
LINEpars = {'wavels': np.asarray(lines_1A),
            'fluxes': np.full(Nlines, 1e6),
            'fwhms': np.full(Nlines, 1e-2)}

lines = sed.LinesSed(**LINEpars)


pt.set_SED(lines)



# generate the scene using the scene configuration parser
# (adding in a telescope background)
scene_config = SceneConfig.makeScene(loglevel=0,
                                     background =  Background(level='low',gradient=0,
↪pa=0),
                                     targets = [pt,ex_disk])
```

```
2020-08-04 15:26:07,854 - INFO - Initializing SersicDisk
2020-08-04 15:26:07,855 - INFO - Initializing SersicDisk
2020-08-04 15:26:07,856 - INFO - Initializing Point
2020-08-04 15:26:07,901 - INFO - Initializing Background
```

12.2 Set up the simulation

with a single exposure and integration, and 50 frames

[3]:
```
sim_config = SimConfig.makeSim(
    name = 'MRS_simulation',     # name given to simulation
    scene = 'MRS_scene.ini', # name of scene file to input
    rel_obsdate = 0.0,           # relative observation date (0 = launch, 1 = end of 5␣
↪yrs)
    POP = 'MRS',                 # Component on which to center (Imager or MRS)
    ConfigPath = 'MRS_1SHORT',  # Configure the Optical path (MRS sub-band)
    Dither = False,              # Don't Dither
    StartInd = 1,                # start index for dither pattern [NOT USED HERE]
    NDither = 2,                 # number of dither positions [NOT USED HERE]
    DitherPat = 'mrs_recommended_dither.dat', # dither pattern to use [NOT USED HERE]
    disperser = 'SHORT',         # [NOT USED HERE]
    detector = 'SW',             # [NOT USED HERE]
    mrs_mode = 'SLOW',           # [NOT USED HERE]
    mrs_exposures = 1,           # [NOT USED HERE]
    mrs_integrations = 1,        # [NOT USED HERE]
    mrs_frames = 50,              # [NOT USED HERE]
    ima_exposures = 1,           # number of exposures
    ima_integrations = 1,        # number of integrations
    ima_frames = 100,             # number of groups (for MIRI, # Groups = # Frames)
    ima_mode = 'FAST',           # Imager read mode (default is FAST ~ 2.3 s)
```

(continues on next page)

(continued from previous page)

```
    filter = 'F1000W',          # Imager Filter to use
    readDetect = 'FULL'         # Portion of detector to read out
)
```

12.3 Run the simulation

Import the default simulator configuration file, and run the simulation with the properties defined above

[4]:
```
# load in default simulator configuration file
simulator_config = SimulatorConfig.from_default()

# setup the sim with the above properties
mysim = MiriSimulation(sim_config,scene_config,simulator_config)
# run the simulation
mysim.run()
```

```
2020-08-04 15:26:07,915 - INFO - MIRISim version: 2.3.0b0
2020-08-04 15:26:07,915 - INFO - MIRI Simulation started.
2020-08-04 15:26:07,916 - INFO - Output will be saved to: 20200804_152607_mirisim
2020-08-04 15:26:07,917 - INFO - Storing configs in output directory.
2020-08-04 15:26:08,395 - INFO - Reading cosmic ray properties from parameter file /
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪miri/simulators/scasim/cosmic_ray_properties.py
2020-08-04 15:26:08,401 - INFO - Reading detector properties from parameter file /
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪miri/simulators/scasim/detector_properties.py
2020-08-04 15:26:08,437 - INFO - Storing dither pattern in output directory.
2020-08-04 15:26:08,439 - INFO - Using $CDP_DIR for location of CDP files: /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP
2020-08-04 15:26:08,439 - INFO - Setting up simulated Observation, with following␣
↪settings:
2020-08-04 15:26:08,440 - INFO - Configuration Path: MRS_1SHORT
2020-08-04 15:26:08,440 - INFO - Primary optical path: MRS
2020-08-04 15:26:08,441 - INFO - MRS detector: SW
2020-08-04 15:26:08,441 - INFO - MRS band: SHORT
2020-08-04 15:26:08,442 - INFO - MRS detector readout mode: SLOW
2020-08-04 15:26:08,442 - INFO - MRS detector # exposures: 1
2020-08-04 15:26:08,443 - INFO - MRS detector # integrations: 1
2020-08-04 15:26:08,444 - INFO - MRS detector # frames: 50
2020-08-04 15:26:08,444 - INFO - Parsing: Background
2020-08-04 15:26:08,445 - INFO - Initializing Background
2020-08-04 15:26:08,445 - INFO - Parsing: point_1
2020-08-04 15:26:08,446 - INFO - Initializing Point
2020-08-04 15:26:08,488 - INFO - Parsing: sersicdisk_1
2020-08-04 15:26:08,489 - INFO - Initializing SersicDisk
2020-08-04 15:26:08,490 - INFO - Initializing SersicDisk
2020-08-04 15:26:08,491 - INFO - Simulating a single pointing.
2020-08-04 15:26:10,927 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪gwcs/wcs.py:131: VisibleDeprecationWarning: Creating an ndarray from ragged nested␣
↪sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different␣
↪lengths or shapes) is deprecated. If you meant to do this, you must specify
↪'dtype=object' when creating the ndarray
```

(continues on next page)

(continued from previous page)

```
  transforms = np.array(self._pipeline[from_ind: to_ind])[:, 1].copy()
2020-08-04 15:26:12,122 - INFO - Creating pointing for position 1
2020-08-04 15:26:12,123 - INFO - Creating exposure event for position 1
2020-08-04 15:26:12,127 - INFO - Observation simulation started.
2020-08-04 15:26:12,128 - INFO - Simulating ExposureEvent for pointing 1
2020-08-04 15:26:12,128 - INFO - Simulating MRS exposures for pointing 1
2020-08-04 15:26:12,129 - INFO - Creating skycubes for MRS exposures for pointing 1
2020-08-04 15:26:12,131 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:26:12,626 - INFO - Reading 'PSF' model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_1SHORT_PSF_07.02.00.fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪gwcs/wcs.py:126: VisibleDeprecationWarning: Creating an ndarray from ragged nested␣
↪sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different␣
↪lengths or shapes) is deprecated. If you meant to do this, you must specify
↪'dtype=object' when creating the ndarray
  transforms = np.array(self._pipeline[to_ind: from_ind])[:, 1].tolist()
WARNING: Model is linear in parameters; consider using linear fitting methods.␣
↪[astropy.modeling.fitting]
WARNING:astropy:Model is linear in parameters; consider using linear fitting methods.
2020-08-04 15:26:25,320 - INFO - Wrote skycube: 20200804_152607_mirisim/skycubes/
↪skycube_seq1_1SHORT.fits
2020-08-04 15:26:25,322 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:26:25,603 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:26:26,021 - INFO - Reading 'PSF' model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_2SHORT_PSF_07.02.00.fits'
2020-08-04 15:26:42,473 - INFO - Wrote skycube: 20200804_152607_mirisim/skycubes/
↪skycube_seq1_2SHORT.fits
2020-08-04 15:26:42,474 - INFO - Simulating detector illumination for MRS exposures␣
↪for pointing 1
2020-08-04 15:26:42,482 - INFO - Starting pySpecSimMiri
2020-08-04 15:26:42,489 - INFO - Simulating setup file /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/pySpecSim/MIRI/
↪MiriSetupData.py
2020-08-04 15:26:42,490 - INFO - Instrument setup starting.
2020-08-04 15:26:42,490 - INFO - Detector setup starting.
2020-08-04 15:26:42,491 - INFO - Could not find value for numBlind Columns, assuming␣
↪4.
2020-08-04 15:26:42,493 - INFO - MRS light path simulation starting.
2020-08-04 15:26:42,493 - INFO - Simulating Instrument.
2020-08-04 15:26:42,504 - INFO - Simulating Detector.
2020-08-04 15:26:42,505 - INFO - Simulating Detector Mapping for channel ['1'] and␣
↪band SHORT
2020-08-04 15:26:42,505 - INFO - Getting Coefficients and Substitute values
2020-08-04 15:26:42,507 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:26:42,816 - INFO - Valid cache found.
2020-08-04 15:26:42,820 - INFO - Starting Coordinate Transformation
2020-08-04 15:26:52,395 - INFO - Starting pixel projection with cache.
2020-08-04 15:27:53,313 - INFO - Mapping completed
2020-08-04 15:27:53,313 - INFO - Detector Intensity Array completed.
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:27:53,824 - INFO - Reading 'FRINGE' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_FRINGE_07.02.05.fits'
2020-08-04 15:27:54,001 - INFO - Reading 'PHOTOM' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_PHOTOM_8B.04.00.fits'
2020-08-04 15:27:54,248 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_GAIN_04.00.00.fits'
2020-08-04 15:27:54,305 - INFO - Simulating Detector Mapping for channel ['2'] and␣
↪band SHORT
2020-08-04 15:27:54,306 - INFO - Getting Coefficients and Substitute values
2020-08-04 15:27:54,307 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:27:54,943 - INFO - Valid cache found.
2020-08-04 15:27:54,947 - INFO - Starting Coordinate Transformation
2020-08-04 15:28:04,949 - INFO - Starting pixel projection with cache.
2020-08-04 15:29:04,364 - INFO - Mapping completed
2020-08-04 15:29:04,364 - INFO - Detector Intensity Array completed.
2020-08-04 15:29:04,781 - INFO - Reading 'FRINGE' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_FRINGE_07.02.05.fits'
2020-08-04 15:29:04,908 - INFO - Reading 'PHOTOM' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_PHOTOM_8B.04.00.fits'
2020-08-04 15:29:05,008 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_GAIN_04.00.00.fits'
2020-08-04 15:29:05,045 - INFO - MRS light path simulation finished.
2020-08-04 15:29:05,050 - INFO - Added 1 Cube to output list
2020-08-04 15:29:05,051 - INFO - Added 2 Cube to output list
2020-08-04 15:29:05,295 - INFO - Wrote illumination model: 20200804_152607_mirisim/
↪illum_models/illum_model_seq1_MIRIFUSHORT_12SHORT.fits
2020-08-04 15:29:05,296 - INFO - Simulating integrated detector images for MRS␣
↪exposures for pointing 1
2020-08-04 15:29:05,297 - INFO - Simulating MRS exposure 1
2020-08-04 15:29:05,302 - INFO - Running SCASim
2020-08-04 15:29:05,303 - INFO - Simulating  detector readout for MIRIFUSHORT from␣
↪illumination data model of shape (1024, 1024).
2020-08-04 15:29:05,321 - INFO - Results will be returned in an exposure data model.
2020-08-04 15:29:05,334 - INFO -   Detector readout mode is SLOW (samplesum=8,␣
↪sampleskip=1, nframe=1, groupgap=0)
2020-08-04 15:29:05,335 - INFO -   with 1 integrations and ngroups=50 defined␣
↪explicitly.
2020-08-04 15:29:05,335 - INFO -   Detector subarray mode is FULL.
2020-08-04 15:29:05,336 - INFO - Detector temperature = 6.70 K (which affects dark␣
↪current and read noise).
2020-08-04 15:29:05,337 - INFO - Cosmic ray environment is SOLAR_MIN.
2020-08-04 15:29:05,337 - INFO - Reading cosmic ray library file: '/Users/
↪pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/
↪simulators/data/cosmic_rays/CRs_SiAs_470_SUNMIN_08.fits'
2020-08-04 15:29:05,391 - INFO - Simulation control flags:
2020-08-04 15:29:05,392 - INFO -        Quantum efficiency simulation turned OFF.
2020-08-04 15:29:05,395 - INFO -        Poisson noise simulation turned ON.
2020-08-04 15:29:05,395 - INFO -        Read noise simulation turned ON.
2020-08-04 15:29:05,396 - INFO -        Reference pixels simulation turned ON.
2020-08-04 15:29:05,396 - INFO -        Bad pixels simulation turned ON.
2020-08-04 15:29:05,397 - INFO -        Dark current simulation turned ON.
2020-08-04 15:29:05,398 - INFO -        Flat-field simulation turned ON.
2020-08-04 15:29:05,398 - INFO -        Amplifier bias and gain turned ON.
2020-08-04 15:29:05,399 - INFO -        Detector non-linearity effects turned ON.
2020-08-04 15:29:05,399 - INFO -        Detector drift and latency effects not␣
↪simulated for SLOW readout mode.
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:29:05,400 - INFO - No output subarray mode. Input subarray mode assumed␣
↪FULL.
2020-08-04 15:29:05,401 - INFO - Input subarray mode assumed FULL
2020-08-04 15:29:05,403 - INFO - Creating a new detector object for 1024 rows x 1024␣
↪columns.
2020-08-04 15:29:05,407 - INFO - Reading 'MASK' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_MASK_07.02.01.fits'
2020-08-04 15:29:05,479 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_GAIN_04.00.00.fits'
2020-08-04 15:29:09,956 - INFO - Reading DARK model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_DARK_06.01.00.fits'
2020-08-04 15:29:10,522 - INFO - Reading 'PIXELFLAT' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_FLAT_8B.01.02.fits'
2020-08-04 15:29:10,590 - WARNING - Could not find exact match for pixel flat-field␣
↪for detector MIRIFUSHORT with SLOW mode with band='SHORT'. An alternative is being␣
↪used.
2020-08-04 15:29:10,605 - INFO - Reading 'LINEARITY' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_LINEARITY_06.02.00.fits'
2020-08-04 15:29:10,866 - INFO - Reading 'READNOISE' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_READNOISE_07.01.00.
↪fits'
2020-08-04 15:29:10,922 - INFO - Creating exposure_data with 1280 rows x 1032 columns␣
↪plus 50 groups and 1 ints.
2020-08-04 15:29:11,404 - INFO - Simulating 1 integration.
2020-08-04 15:29:11,411 - WARNING - ***Input flux array contains 1 negative values.
2020-08-04 15:29:11,415 - WARNING - ***Input flux array contains values that could␣
↪saturate at least 562 pixels.
2020-08-04 15:29:11,416 - WARNING -    Maximum flux of 20825.7 photons/s/pixel is␣
↪greater than first frame saturation level of 14993.4 photons/s/pixel.
2020-08-04 15:29:11,464 - INFO - Simulating 50 groups for integration 1.
2020-08-04 15:29:26,497 - INFO - Adding the DARK calibration from /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_DARK_06.01.00.fits
2020-08-04 15:29:26,499 - ERROR - DARK data has insufficient groups.: DARK addition␣
↪skipped.
2020-08-04 15:29:26,500 - INFO - Correcting nonlinearity from MIRI_FM_MIRIFUSHORT_12_
↪LINEARITY_06.02.00.fits
2020-08-04 15:29:27,903 - INFO - Output subarray undefined or FULL. SUBSTRT=(1,1),␣
↪SUBSIZE=(1032,1024)
2020-08-04 15:29:27,903 - INFO - WCS keywords defined as CRPIX1=0, CRPIX2=0
2020-08-04 15:29:28,646 - INFO - Exposure time 1194.50s (duration 1195.55s)
2020-08-04 15:29:31,686 - INFO - Wrote detector image: 20200804_152607_mirisim/det_
↪images/det_image_seq1_MIRIFUSHORT_12SHORTexp1.fits
2020-08-04 15:29:31,687 - INFO - MIRI Simulation finished. Results have been saved to:
↪ 20200804_152607_mirisim
```

[5]:
```
# save the name of the output directory
outputdir = sorted(glob.glob('*_*_mirisim'),key=os.path.getctime)[-1]


# the routine here doesn't do anything itself,
#but is used to plot the illumination models below

def show_outputs(MIRISim_outputdir,output_type):
    '''
    plot the specified channel of the MIRISim outputs
```

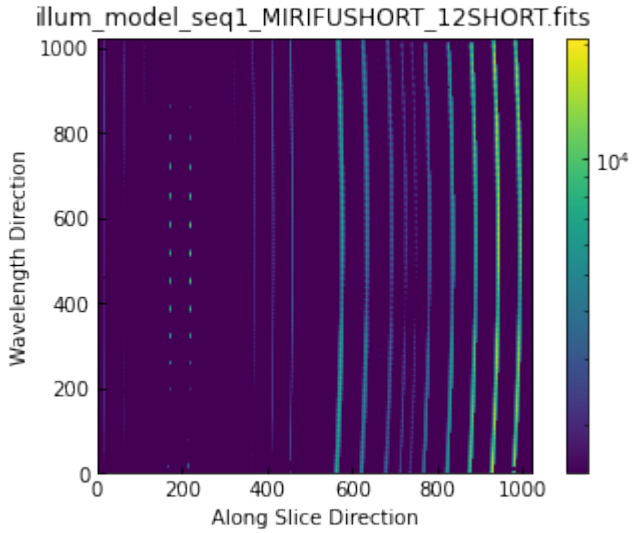
(continues on next page)

(continued from previous page)

```
    :param MIRISim_outputdir:
        name of the date-labelled dir. holding the MIRISIM outputs
    :param output_type:
        type of output to process
        (e.g. illum_models, det_images or skycubes)
    '''

    infits = glob.glob('{}/{}/*.fits'.format(MIRISim_outputdir,output_type))[0]

    hdulist = fits.open(infits)

    if output_type.lower() == 'skycubes':
        hdu_index = 0
        datashape = hdulist[hdu_index].data.shape
        central_chan = datashape[0]//2

        plt.imshow(hdulist[hdu_index].data[central_chan,:,:],
           origin = 'lower', interpolation = 'nearest',cmap = cm.viridis)
        plt.title('channel {} of {}'.format(central_chan,infits.split('/')[-1]))
        plt.xlabel(hdulist[hdu_index].header['ctype1'])
        plt.ylabel(hdulist[hdu_index].header['ctype2'])
    else:
        hdu_index = 1
        if len(hdulist[hdu_index].data.shape) > 2:
            integ,frames,nx,ny = hdulist[hdu_index].data.shape
            image = hdulist[hdu_index].data[integ-1,frames-1,:,:]
        else:
            image = hdulist[hdu_index].data

        # normalise the colourscale to just above the
        # background (mean) value, and use a LogNorm scaling
        norm = colors.LogNorm(image.mean() + 0.5 * image.std(), image.max(),␣
↪clip=True)
        #norm = colors.LogNorm(image.mean() - 0.5*image.std(), image.max(), clip=True)
        plt.imshow(image,origin = 'lower', cmap = cm.viridis,interpolation = 'nearest
↪',norm = norm)
        plt.title('{}'.format(infits.split('/')[-1]))
        plt.xlabel('Along Slice Direction')
        plt.ylabel('Wavelength Direction')


    plt.colorbar()
    plt.savefig('{}.pdf'.format(output_type))
```

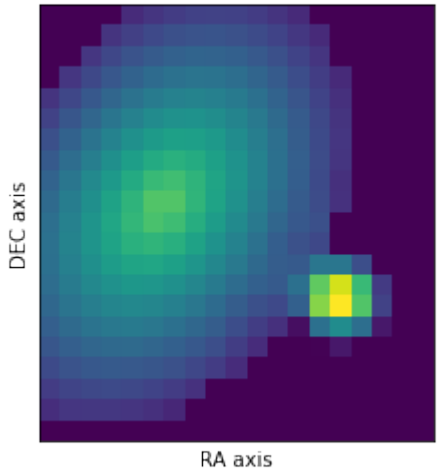
12.4 show the results of the simulation

[6]:
```
show_outputs(outputdir,'illum_models')
```



12.4.1 show a channel of the sky cube (that has an emission line from the point source)

[7]:
```
hdulist = fits.open('{}/skycubes/skycube_seq1_1SHORT.fits'.format(outputdir))

data = hdulist[0].data
image = data[54,:,:]

norm = colors.LogNorm(
        image.mean() - 0.5*image.std(), image.max(),clip=True)
plt.imshow(image,origin = 'lower',
           cmap = cm.viridis,
           interpolation = 'nearest',norm = norm)

plt.xticks([])
plt.yticks([])
plt.xlabel('RA axis')
plt.ylabel('DEC axis')


plt.savefig('skycube_slice.pdf')
```


DEC axis
RA axis

[ ]:

[ ]:

CHAPTER

THIRTEEN

APPENDIX: WALK THROUGH OF MIRISIM (MRS)

In this notebook, we create a scene with a two point sources and a galaxy (that all fit within the MRS field of view), and run through a MIRISim simulation showing the products created at each stage.

NOTE this notebook must be started within the mirisim anaconda environment. This is a pre-requist for using MIRISim, and installation instructions can be found here.

When launching the notebook, make sure you've run the appropriate version of

conda activate mirisim

in the terminal before starting the notebook.

13.1 Steps in this notebook:

1. Create a Scene
2. Initialise the simulation parameters
3. run the simulation
4. examine some of the outputs.

[1]:
```
# import the configuration file parsers so they can be written to file
from mirisim.config_parser import SimConfig, SimulatorConfig, SceneConfig

# import scene component generators
from mirisim.skysim import Background, sed, Point, Galaxy, kinetics
from mirisim.skysim import wrap_pysynphot as wS

from mirisim import MiriSimulation

#other things to be used
import numpy as np
import glob                 # glob is used to find the output directory
import os                   # for listing directory contents
from astropy.io import fits # for reading FITS file contents

import matplotlib.pyplot as plt    # to display images
from matplotlib import colors,cm
%matplotlib inline
```

13.1.1 Create a Scene

The scene created below consists of two point sources and an edge-on Galaxy, all three of which fit within the MRS field of view. The first point source has a blackbody SED, the second a pysynphot SED. The galaxy has a powerlaw SED, which is modified by a Keplerian velocity field.

Create the First Point Source

[2]:
```
# initialise the point source with a position
point1 = Point(Cen = (-0.5,0.5))

# set properties of the SED
Blackbody = sed.BBSed(Temp = 800., wref = 10., flux = 1e5)

# add the SED to the point source
point1.set_SED(Blackbody)
```

```
2020-08-04 15:22:04,014 - INFO - Initializing Point
```

Create the Second Point Source

[3]:
```
# initialise the point source with a position
point2 = Point(Cen = (0.5,0.5))

# set the properties of the SED
PYSPsedDict = {'family':'bkmodels','sedname':'bk_b0005','flux':1.e5, 'wref':10.}
sedP2 = wS.PYSPSed(**PYSPsedDict)

# add the SED to the point source
point2.set_SED(sedP2)
```

```
2020-08-04 15:22:04,021 - INFO - Initializing Point
```

Create the Galaxy

[4]:
```

# initialise the galaxy with a position
galaxy = Galaxy(Cen = (0.,-0.5),n=1.,re=1.,q=0.1,pa=90)

# set the properties of the SED
PowerLaw = sed.PLSed(alpha=1.0,flux=5e5,wref=10.)
# add the SED to the galaxy
galaxy.set_SED(PowerLaw)

# create a velocity mapping for the SED
VMAPpars = {'vrot': 200., 'Cen': (0., -0.5), 'pa': 90., 'q': 0.1, 'c': 0}
VelocityMap = kinetics.FlatDisk(**VMAPpars)
# add the velocity map to the galaxy
galaxy.set_velomap(VelocityMap)

# add a line of sight velocity distribution to the galaxy
losVeloDist = kinetics.Losvd(sigma=200.,h3=0.,h4=0.)
galaxy.set_LOSVD(losVeloDist)
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:22:04,087 - INFO - Initializing Galaxy
2020-08-04 15:22:04,088 - INFO - Initializing Galaxy
2020-08-04 15:22:04,088 - INFO - Initializing Galaxy
2020-08-04 15:22:04,089 - INFO - Initializing FlatDisk
2020-08-04 15:22:04,090 - INFO - Initializing Losvd
```

Create a low level background for the scene

[5]:
```
### create a low level background for the scene
bg = Background(level='low',gradient=5., pa=45.)
```

```
2020-08-04 15:22:04,144 - INFO - Initializing Background
```

13.2 From the components, create a scene

create a scene is as simple as adding together the targets.

[6]:
```
scene = bg + point1 + point2 + galaxy
```

13.3 Export the scene to an ini file and FITS

The scene can also be exported to an ini file (for future use), or a FITS file to be visualised.

Exporting to a FITS file requires specifing a number of aditional parametrs such as the Field of view, required spectral sampling, etc. They're described in more detail below. For large fields of view, and/or small spectral channels, which create large FITS files, writing to a FITS file can take a while (few minutes)

[7]:
```
## export to ini file

scene_config = SceneConfig.makeScene(loglevel=0,
                                     background=bg,
                                     targets = [point1,point2,galaxy])

#print [point1,point2,galaxy]

os.system('rm MRS_example_scene.ini')
scene_config.write('MRS_example_scene.ini')
'''
## export to FITS file
FOV = np.array([[-4.,4.],[-4.,4.]])   # field of view [xmin,xmax],[ymin,ymax] (in␣
↪arcsec)
SpatialSampling = 0.1                 # spatial sampling (in arcsec)
WavelengthRange = [5,15]              # wavelength range to process (in microns)
WavelengthSampling = 0.05             # channel width (in microns)
```

(continues on next page)

(continued from previous page)

```
# overwrite = True enables overwriting of any previous version of the fits file
# with the same name as that given in the writecube command
scene.writecube(cubefits = 'MRS_example_scene.fits',
                FOV = FOV, time = 0.0,
                spatsampling = SpatialSampling,
                wrange = WavelengthRange,
                wsampling = WavelengthSampling,
                overwrite = True)

# note: the galaxy is in the output FITS file, it's just very faint
'''
```

[7]:
```
"\n## export to FITS file\nFOV = np.array([[-4.,4.],[-4.,4.]])   # field of view␣
↪[xmin,xmax],[ymin,ymax] (in arcsec)\nSpatialSampling = 0.1               # spatial␣
↪sampling (in arcsec)\nWavelengthRange = [5,15]            # wavelength range to␣
↪process (in microns)\nWavelengthSampling = 0.05           # channel width (in␣
↪microns)\n\n\n# overwrite = True enables overwriting of any previous version of the␣
↪fits file\n# with the same name as that given in the writecube command\nscene.
↪writecube(cubefits = 'MRS_example_scene.fits',\n               FOV = FOV, time = 0.
↪0,\n               spatsampling = SpatialSampling,\n               wrange =␣
↪WavelengthRange,\n               wsampling = WavelengthSampling,\n              ␣
↪overwrite = True)  \n\n# note: the galaxy is in the output FITS file, it's just␣
↪very faint\n"
```

13.3.1 Initialise the Simulation Parameters

This is where the parameters for the MRS simulation get set. Note that for internal consistency in MIRISim, all settings (including those not used in the MRS simulation here) must be set. Those not being used in this simulation are labelled with NOT USED HERE in the comments of each line

[8]:
```
sim_config = SimConfig.makeSim(
    name = 'mrs_simulation',     # name given to simulation
    scene = 'MRS_example_scene.ini', # name of scene file to input
    rel_obsdate = 0.0,           # relative observation date (0 = launch, 1 = end of 5␣
↪yrs)
    POP = 'MRS',                 # Component on which to center (Imager or MRS)
    ConfigPath = 'MRS_1SHORT',   # Configure the Optical path (MRS sub-band)
    Dither = False,              # Don't Dither
    StartInd = 1,                # start index for dither pattern [NOT USED HERE]
    NDither = 2,                 # number of dither positions [NOT USED HERE]
    DitherPat = 'mrs_recommended_dither.dat', # dither pattern to use [NOT USED HERE]
    disperser = 'SHORT',         # Which disperser to use (SHORT/MEDIUM/LONG)
    detector = 'SW',             # Specify Channel (SW = channels 1,2, LW= channels 3,
↪4)
    mrs_mode = 'SLOW',           # MRS read mode (default is SLOW. ~ 24s)
    mrs_exposures = 2,           # number of exposures
    mrs_integrations = 3,        # number of integrations
    mrs_frames = 5,              # number of groups (for MIRI, # Groups = # Frames)
    ima_exposures = 0,           # [NOT USED HERE]
    ima_integrations = 0,        # [NOT USED HERE]
    ima_frames = 0,              # [NOT USED HERE]
    ima_mode = 'FAST',           # [NOT USED HERE]
    filter = 'F1130W',           # [NOT USED HERE]
```

(continues on next page)

(continued from previous page)

```
    readDetect = 'FULL'          # [NOT USED HERE]
)
```

13.4 Export the simulation setup to a file

[9]:
```
os.system('rm MRS_simulation.ini')
sim_config.write('MRS_simulation.ini')
```

13.4.1 Run the simulation

Now that the scene and the setup of the simulation have been set, we can run the simulation.

the last step is to setup the defaults for internal things like CDPs.

[10]:
```
simulator_config = SimulatorConfig.from_default()


mysim = MiriSimulation(sim_config,scene_config,simulator_config)
mysim.run()
```

```
2020-08-04 15:22:04,195 - INFO - MIRISim version: 2.3.0b0
2020-08-04 15:22:04,196 - INFO - MIRI Simulation started.
2020-08-04 15:22:04,197 - INFO - Output will be saved to: 20200804_152204_mirisim
2020-08-04 15:22:04,198 - INFO - Storing configs in output directory.
2020-08-04 15:22:04,664 - INFO - Reading cosmic ray properties from parameter file /
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪miri/simulators/scasim/cosmic_ray_properties.py
2020-08-04 15:22:04,669 - INFO - Reading detector properties from parameter file /
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪miri/simulators/scasim/detector_properties.py
2020-08-04 15:22:04,704 - INFO - Storing dither pattern in output directory.
2020-08-04 15:22:04,711 - INFO - Using $CDP_DIR for location of CDP files: /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP
2020-08-04 15:22:04,712 - INFO - Setting up simulated Observation, with following␣
↪settings:
2020-08-04 15:22:04,713 - INFO - Configuration Path: MRS_1SHORT
2020-08-04 15:22:04,713 - INFO - Primary optical path: MRS
2020-08-04 15:22:04,714 - INFO - MRS detector: SW
2020-08-04 15:22:04,714 - INFO - MRS band: SHORT
2020-08-04 15:22:04,715 - INFO - MRS detector readout mode: SLOW
2020-08-04 15:22:04,715 - INFO - MRS detector # exposures: 2
2020-08-04 15:22:04,715 - INFO - MRS detector # integrations: 3
2020-08-04 15:22:04,716 - INFO - MRS detector # frames: 5
2020-08-04 15:22:04,717 - INFO - Parsing: Background
2020-08-04 15:22:04,717 - INFO - Initializing Background
2020-08-04 15:22:04,718 - INFO - Parsing: point_1
2020-08-04 15:22:04,718 - INFO - Initializing Point
2020-08-04 15:22:04,719 - INFO - Parsing: point_2
2020-08-04 15:22:04,719 - INFO - Initializing Point
2020-08-04 15:22:04,765 - INFO - Parsing: galaxy_1
2020-08-04 15:22:04,766 - INFO - Initializing Galaxy
2020-08-04 15:22:04,772 - INFO - Initializing Galaxy
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:22:04,773 - INFO - Initializing Galaxy
2020-08-04 15:22:04,774 - INFO - Initializing FlatDisk
2020-08-04 15:22:04,774 - INFO - Initializing Losvd
2020-08-04 15:22:04,822 - INFO - Simulating a single pointing.
2020-08-04 15:22:08,817 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪gwcs/wcs.py:131: VisibleDeprecationWarning: Creating an ndarray from ragged nested␣
↪sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different␣
↪lengths or shapes) is deprecated. If you meant to do this, you must specify
↪'dtype=object' when creating the ndarray
  transforms = np.array(self._pipeline[from_ind: to_ind])[:, 1].copy()
2020-08-04 15:22:10,044 - INFO - Creating pointing for position 1
2020-08-04 15:22:10,045 - INFO - Creating exposure event for position 1
2020-08-04 15:22:10,051 - INFO - Observation simulation started.
2020-08-04 15:22:10,052 - INFO - Simulating ExposureEvent for pointing 1
2020-08-04 15:22:10,053 - INFO - Simulating MRS exposures for pointing 1
2020-08-04 15:22:10,056 - INFO - Creating skycubes for MRS exposures for pointing 1
2020-08-04 15:22:10,058 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:22:10,697 - INFO - Reading 'PSF' model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_1SHORT_PSF_07.02.00.fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪gwcs/wcs.py:126: VisibleDeprecationWarning: Creating an ndarray from ragged nested␣
↪sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different␣
↪lengths or shapes) is deprecated. If you meant to do this, you must specify
↪'dtype=object' when creating the ndarray
  transforms = np.array(self._pipeline[to_ind: from_ind])[:, 1].tolist()
WARNING: Model is linear in parameters; consider using linear fitting methods.␣
↪[astropy.modeling.fitting]
WARNING:astropy:Model is linear in parameters; consider using linear fitting methods.
2020-08-04 15:22:29,916 - INFO - Wrote skycube: 20200804_152204_mirisim/skycubes/
↪skycube_seq1_1SHORT.fits
2020-08-04 15:22:29,918 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:22:30,253 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:22:30,839 - INFO - Reading 'PSF' model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_2SHORT_PSF_07.02.00.fits'
2020-08-04 15:22:55,544 - INFO - Wrote skycube: 20200804_152204_mirisim/skycubes/
↪skycube_seq1_2SHORT.fits
2020-08-04 15:22:55,546 - INFO - Simulating detector illumination for MRS exposures␣
↪for pointing 1
2020-08-04 15:22:55,552 - INFO - Starting pySpecSimMiri
2020-08-04 15:22:55,581 - INFO - Simulating setup file /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/pySpecSim/MIRI/
↪MiriSetupData.py
2020-08-04 15:22:55,582 - INFO - Instrument setup starting.
2020-08-04 15:22:55,583 - INFO - Detector setup starting.
2020-08-04 15:22:55,584 - INFO - Could not find value for numBlind Columns, assuming␣
↪4.
2020-08-04 15:22:55,586 - INFO - MRS light path simulation starting.
2020-08-04 15:22:55,587 - INFO - Simulating Instrument.
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:22:55,596 - INFO - Simulating Detector.
2020-08-04 15:22:55,596 - INFO - Simulating Detector Mapping for channel ['1'] and␣
↪band SHORT
2020-08-04 15:22:55,597 - INFO - Getting Coefficients and Substitute values
2020-08-04 15:22:55,599 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:22:55,957 - INFO - Valid cache found.
2020-08-04 15:22:55,958 - INFO - Starting Coordinate Transformation
2020-08-04 15:23:06,070 - INFO - Starting pixel projection with cache.
2020-08-04 15:24:06,746 - INFO - Mapping completed
2020-08-04 15:24:06,747 - INFO - Detector Intensity Array completed.
2020-08-04 15:24:07,239 - INFO - Reading 'FRINGE' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_FRINGE_07.02.05.fits'
2020-08-04 15:24:07,423 - INFO - Reading 'PHOTOM' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_PHOTOM_8B.04.00.fits'
2020-08-04 15:24:07,675 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_GAIN_04.00.00.fits'
2020-08-04 15:24:07,727 - INFO - Simulating Detector Mapping for channel ['2'] and␣
↪band SHORT
2020-08-04 15:24:07,727 - INFO - Getting Coefficients and Substitute values
2020-08-04 15:24:07,729 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_DISTORTION_8B.05.01.
↪fits'
2020-08-04 15:24:08,328 - INFO - Valid cache found.
2020-08-04 15:24:08,329 - INFO - Starting Coordinate Transformation
2020-08-04 15:24:17,903 - INFO - Starting pixel projection with cache.
2020-08-04 15:25:14,678 - INFO - Mapping completed
2020-08-04 15:25:14,679 - INFO - Detector Intensity Array completed.
2020-08-04 15:25:15,058 - INFO - Reading 'FRINGE' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_FRINGE_07.02.05.fits'
2020-08-04 15:25:15,169 - INFO - Reading 'PHOTOM' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_PHOTOM_8B.04.00.fits'
2020-08-04 15:25:15,268 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_GAIN_04.00.00.fits'
2020-08-04 15:25:15,303 - INFO - MRS light path simulation finished.
2020-08-04 15:25:15,323 - INFO - Added 1 Cube to output list
2020-08-04 15:25:15,323 - INFO - Added 2 Cube to output list
2020-08-04 15:25:15,523 - INFO - Wrote illumination model: 20200804_152204_mirisim/
↪illum_models/illum_model_seq1_MIRIFUSHORT_12SHORT.fits
2020-08-04 15:25:15,523 - INFO - Simulating integrated detector images for MRS␣
↪exposures for pointing 1
2020-08-04 15:25:15,524 - INFO - Simulating MRS exposure 1
2020-08-04 15:25:15,528 - INFO - Running SCASim
2020-08-04 15:25:15,529 - INFO - Simulating  detector readout for MIRIFUSHORT from␣
↪illumination data model of shape (1024, 1024).
2020-08-04 15:25:15,539 - INFO - Results will be returned in an exposure data model.
2020-08-04 15:25:15,549 - INFO -   Detector readout mode is SLOW (samplesum=8,␣
↪sampleskip=1, nframe=1, groupgap=0)
2020-08-04 15:25:15,550 - INFO -   with 3 integrations and ngroups=5 defined␣
↪explicitly.
2020-08-04 15:25:15,551 - INFO -   Detector subarray mode is FULL.
2020-08-04 15:25:15,552 - INFO - Detector temperature = 6.70 K (which affects dark␣
↪current and read noise).
2020-08-04 15:25:15,553 - INFO - Cosmic ray environment is SOLAR_MIN.
2020-08-04 15:25:15,554 - INFO - Reading cosmic ray library file: '/Users/
↪pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/
↪simulators/data/cosmic_rays/CRs_SiAs_470_SUNMIN_06.fits'
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:25:15,599 - INFO - Simulation control flags:
2020-08-04 15:25:15,600 - INFO -        Quantum efficiency simulation turned OFF.
2020-08-04 15:25:15,601 - INFO -        Poisson noise simulation turned ON.
2020-08-04 15:25:15,601 - INFO -        Read noise simulation turned ON.
2020-08-04 15:25:15,602 - INFO -        Reference pixels simulation turned ON.
2020-08-04 15:25:15,603 - INFO -        Bad pixels simulation turned ON.
2020-08-04 15:25:15,603 - INFO -        Dark current simulation turned ON.
2020-08-04 15:25:15,604 - INFO -        Flat-field simulation turned ON.
2020-08-04 15:25:15,604 - INFO -        Amplifier bias and gain turned ON.
2020-08-04 15:25:15,605 - INFO -        Detector non-linearity effects turned ON.
2020-08-04 15:25:15,605 - INFO -        Detector drift and latency effects not␣
↪simulated for SLOW readout mode.
2020-08-04 15:25:15,606 - INFO - No output subarray mode. Input subarray mode assumed␣
↪FULL.
2020-08-04 15:25:15,606 - INFO - Input subarray mode assumed FULL
2020-08-04 15:25:15,608 - INFO - Creating a new detector object for 1024 rows x 1024␣
↪columns.
2020-08-04 15:25:15,616 - INFO - Reading 'MASK' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_MASK_07.02.01.fits'
2020-08-04 15:25:15,687 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_GAIN_04.00.00.fits'
2020-08-04 15:25:19,241 - INFO - Reading DARK model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_DARK_06.01.00.fits'
2020-08-04 15:25:19,789 - INFO - Reading 'PIXELFLAT' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_FLAT_8B.01.02.fits'
2020-08-04 15:25:19,849 - WARNING - Could not find exact match for pixel flat-field␣
↪for detector MIRIFUSHORT with SLOW mode with band='SHORT'. An alternative is being␣
↪used.
2020-08-04 15:25:19,864 - INFO - Reading 'LINEARITY' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_LINEARITY_06.02.00.fits'
2020-08-04 15:25:20,115 - INFO - Reading 'READNOISE' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_READNOISE_07.01.00.
↪fits'
2020-08-04 15:25:20,166 - INFO - Creating exposure_data with 1280 rows x 1032 columns␣
↪plus 5 groups and 3 ints.
2020-08-04 15:25:20,357 - INFO - Simulating 3 integrations.
2020-08-04 15:25:20,363 - WARNING - ***Input flux array contains 1 negative values.
2020-08-04 15:25:20,406 - INFO - Simulating 5 groups for integration 1.
2020-08-04 15:25:22,083 - INFO - Simulating 5 groups for integration 2.
2020-08-04 15:25:23,735 - INFO - Simulating 5 groups for integration 3.
2020-08-04 15:25:25,343 - INFO - Adding the DARK calibration from /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_DARK_06.01.00.fits
2020-08-04 15:25:25,501 - INFO - Correcting nonlinearity from MIRI_FM_MIRIFUSHORT_12_
↪LINEARITY_06.02.00.fits
2020-08-04 15:25:25,882 - INFO - Output subarray undefined or FULL. SUBSTRT=(1,1),␣
↪SUBSIZE=(1032,1024)
2020-08-04 15:25:25,883 - INFO - WCS keywords defined as CRPIX1=0, CRPIX2=0
2020-08-04 15:25:26,540 - INFO - Exposure time 358.35s (duration 359.41s)
2020-08-04 15:25:27,575 - INFO - Wrote detector image: 20200804_152204_mirisim/det_
↪images/det_image_seq1_MIRIFUSHORT_12SHORTexp1.fits
2020-08-04 15:25:27,576 - INFO - Simulating MRS exposure 2
2020-08-04 15:25:27,592 - INFO - Running SCASim
2020-08-04 15:25:27,593 - INFO - Simulating  detector readout for MIRIFUSHORT from␣
↪illumination data model of shape (1024, 1024).
2020-08-04 15:25:27,606 - INFO - Results will be returned in an exposure data model.
2020-08-04 15:25:27,619 - INFO -   Detector readout mode is SLOW (samplesum=8,␣
↪sampleskip=1, nframe=1, groupgap=0)
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:25:27,620 - INFO -   with 3 integrations and ngroups=5 defined␣
↪explicitly.
2020-08-04 15:25:27,620 - INFO -   Detector subarray mode is FULL.
2020-08-04 15:25:27,621 - INFO - Detector temperature = 6.70 K (which affects dark␣
↪current and read noise).
2020-08-04 15:25:27,622 - INFO - Cosmic ray environment is SOLAR_MIN. (No change.)
2020-08-04 15:25:27,623 - INFO - Simulation control flags:
2020-08-04 15:25:27,623 - INFO -       Quantum efficiency simulation turned OFF.
2020-08-04 15:25:27,624 - INFO -       Poisson noise simulation turned ON.
2020-08-04 15:25:27,624 - INFO -       Read noise simulation turned ON.
2020-08-04 15:25:27,625 - INFO -       Reference pixels simulation turned ON.
2020-08-04 15:25:27,625 - INFO -       Bad pixels simulation turned ON.
2020-08-04 15:25:27,626 - INFO -       Dark current simulation turned ON.
2020-08-04 15:25:27,626 - INFO -       Flat-field simulation turned ON.
2020-08-04 15:25:27,627 - INFO -       Amplifier bias and gain turned ON.
2020-08-04 15:25:27,627 - INFO -       Detector non-linearity effects turned ON.
2020-08-04 15:25:27,628 - INFO -       Detector drift and latency effects not␣
↪simulated for SLOW readout mode.
2020-08-04 15:25:27,628 - INFO - No output subarray mode. Input subarray mode assumed␣
↪FULL.
2020-08-04 15:25:27,629 - INFO - Input subarray mode assumed FULL
2020-08-04 15:25:27,630 - INFO - Reading 'MASK' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_MASK_07.02.01.fits'
2020-08-04 15:25:27,677 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_GAIN_04.00.00.fits'
2020-08-04 15:25:27,711 - INFO - Reading DARK model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_DARK_06.01.00.fits'
2020-08-04 15:25:28,098 - INFO - Reading 'PIXELFLAT' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12SHORT_FLAT_8B.01.02.fits'
2020-08-04 15:25:28,146 - WARNING - Could not find exact match for pixel flat-field␣
↪for detector MIRIFUSHORT with SLOW mode with band='SHORT'. An alternative is being␣
↪used.
2020-08-04 15:25:28,158 - INFO - Reading 'LINEARITY' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_12_LINEARITY_06.02.00.fits'
2020-08-04 15:25:28,335 - INFO - Reading 'READNOISE' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_READNOISE_07.01.00.
↪fits'
2020-08-04 15:25:28,368 - INFO - Creating exposure_data with 1280 rows x 1032 columns␣
↪plus 5 groups and 3 ints.
2020-08-04 15:25:28,481 - INFO - Simulating 3 integrations.
2020-08-04 15:25:28,488 - WARNING - ***Input flux array contains 1 negative values.
2020-08-04 15:25:28,511 - INFO - Simulating 5 groups for integration 1.
2020-08-04 15:25:30,072 - INFO - Simulating 5 groups for integration 2.
2020-08-04 15:25:31,622 - INFO - Simulating 5 groups for integration 3.
2020-08-04 15:25:33,143 - INFO - Adding the DARK calibration from /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIFUSHORT_SLOW_12_DARK_06.01.00.fits
2020-08-04 15:25:33,286 - INFO - Correcting nonlinearity from MIRI_FM_MIRIFUSHORT_12_
↪LINEARITY_06.02.00.fits
2020-08-04 15:25:33,667 - INFO - Output subarray undefined or FULL. SUBSTRT=(1,1),␣
↪SUBSIZE=(1032,1024)
2020-08-04 15:25:33,668 - INFO - WCS keywords defined as CRPIX1=0, CRPIX2=0
2020-08-04 15:25:34,379 - INFO - Exposure time 358.35s (duration 359.41s)
2020-08-04 15:25:35,437 - INFO - Wrote detector image: 20200804_152204_mirisim/det_
↪images/det_image_seq1_MIRIFUSHORT_12SHORTexp2.fits
2020-08-04 15:25:35,438 - INFO - MIRI Simulation finished. Results have been saved to:
↪ 20200804_152204_mirisim
```

13.4.2 Examine some of the results

Now that the MIRISim simulation has completed, lets examine the results.

The first thing to note is that the outputs are placed in a date-labelled directory taking the form YYYYMMDD_hhmmss_mirisim. The name of the output directory is given in the last line of the MIRISim log above. Because the output directory is date-labelled, we can quantify which was the most recent run of MIRISim, and find its output directory using glob.glob

[11]:
```
outputdir = sorted(glob.glob('*_*_mirisim'),key=os.path.getmtime)[-1]     #[-1] takes␣
↪the last entry found

outputDirContents = os.listdir(outputdir)

directories = [name for name in outputDirContents if os.path.isdir(os.path.
↪join(outputdir,name))]
files = [name for name in outputDirContents if not os.path.isdir(os.path.
↪join(outputdir,name))]

print('The subdirectories in the outputdirectory are:\n{}'.format(directories))
print('The files in the outputdirectory are:\n{}'.format(files))
```

```
The subdirectories in the outputdirectory are:
['det_images', 'skycubes', 'illum_models']
The files in the outputdirectory are:
['simulator.ini', 'scene.ini', 'mrs_recommended_dither.dat', 'simulation.ini',
↪'mirisim.log']
```

The files contain the log which was also output to the terminal (mirisim.log) and copies of the .ini files used (or created from python inputs) to create the simulation. These versions of the .ini files can be used to re-create the run of the simulation

The directories contain various outputs of MIRISim:

- ** skycubes ** houses a 3D representation of the input scene to the MRS simulation (skycubes are not generated for imager or LRS simulations). This cube has not been processed by MIRISim, it is simply a gridded (spatially and spectrally) version of the input scene.
- ** illum_models ** houses FITS images of the illuminations sent to the detector (sent to SCAsim - the simulator of the Sensor Chip Assembly). This should have the same format as the detector image, but without all of the detector effects and noise. There are FITS files produced for each exposure and dither position
- ** det_images ** houses FITS images of the final outputs of MIRISim. These detector images have all of the detector effects and noise incorporated. The number of detector images should be the same as the number of illumination models.

The headers of the detector images are formatted for ingest into the JWST pipeline.

Below is a small code snippet used to draw the images. it needs to be run, but doesn't produce any output directly (it's called later to show the output images)

[12]:
```
def show_outputs(MIRISim_outputdir,output_type):
    '''
    plot the specified channel of the MIRISim outputs
    :param MIRISim_outputdir:
        name of the date-labelled dir. holding the MIRISIM outputs
    :param output_type:
        type of output to process
        (e.g. illum_models, det_images or skycubes)
    '''
```

(continues on next page)

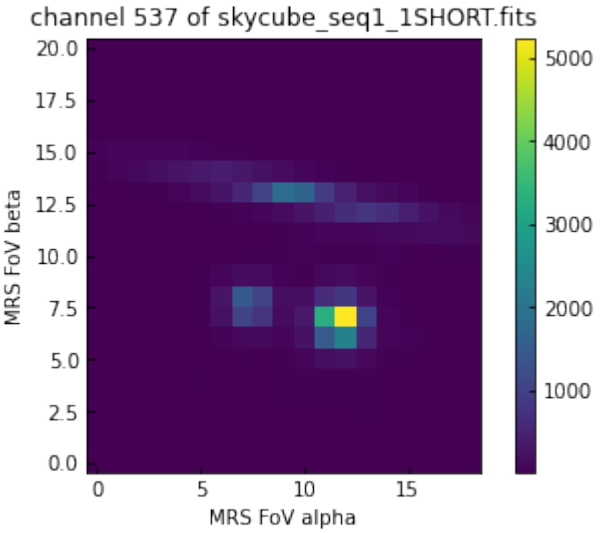
(continued from previous page)

```
    infits = glob.glob('{}/{}/*.fits'.format(MIRISim_outputdir,output_type))[0]

    hdulist = fits.open(infits)

    if output_type.lower() == 'skycubes':
        hdu_index = 0
        datashape = hdulist[hdu_index].data.shape
        central_chan = datashape[0]//2

        plt.imshow(hdulist[hdu_index].data[central_chan,:,:],
           origin = 'lower', interpolation = 'nearest',cmap = cm.viridis)
        plt.title('channel {} of {}'.format(central_chan,infits.split('/')[-1]))
        plt.xlabel(hdulist[hdu_index].header['ctype1'])
        plt.ylabel(hdulist[hdu_index].header['ctype2'])
    else:
        hdu_index = 1
        if len(hdulist[hdu_index].data.shape) > 2:
            integ,frames,nx,ny = hdulist[hdu_index].data.shape
            image = hdulist[hdu_index].data[integ-1,frames-1,:,:]
        else:
            image = hdulist[hdu_index].data


        plt.imshow(image, origin = 'lower',cmap = cm.viridis,interpolation='nearest')
        plt.title('{}'.format(infits.split('/')[-1]))
        plt.xlabel('Along Slice Direction')
        plt.ylabel('Wavelength Direction')


    plt.colorbar()
    plt.savefig('MRS_{}.png'.format(output_type))
```

13.5 Viewing a slice of the skycube

Below, we show an image of the central channel of one of the skycubes generated by MIRISim

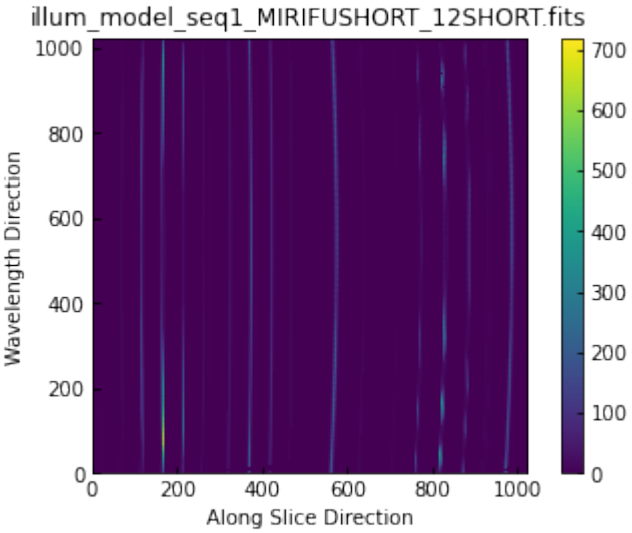
[13]:
```
show_outputs(outputdir,'skycubes')
```



13.6 Viewing an illumination model

Below shows an example of an output illumination model

[14]:
```
show_outputs(outputdir,'illum_models')
```



13.6.1 Viewing the final detector image

The detector images are the final output of MIRISim, and have data structures and formatting consistent with what will come from MIRI itself. The data format is JWST pipeline ready. Below an example image of the last frame of the last integration is shown. The units are DN.

Additionally, below the FITS header information for the SCIENCE extension is listed

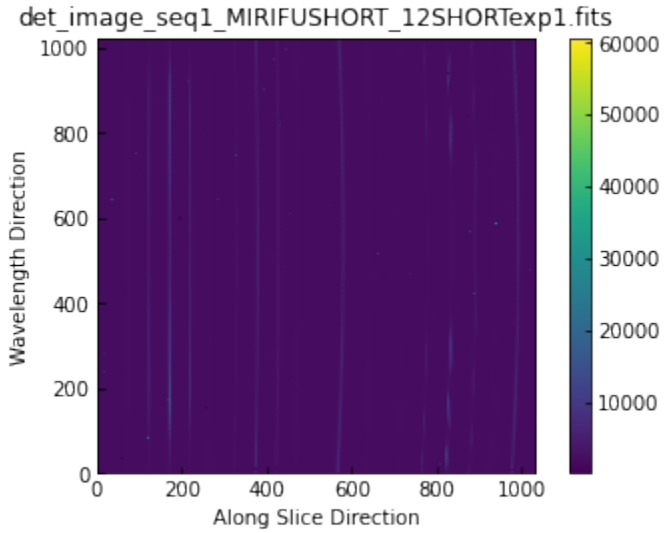
[15]:
```
show_outputs(outputdir,'det_images')

infits = glob.glob('{}/det_images/*.fits'.format(outputdir))[0]
hdulist = fits.open(infits)
hdulist[1].header
```

[15]:
```
XTENSION= 'IMAGE   '           / Image extension
BITPIX  =                  -32 / array data type
NAXIS   =                    4 / number of array dimensions
NAXIS1  =                 1032
NAXIS2  =                 1024
NAXIS3  =                    5
NAXIS4  =                    3
PCOUNT  =                    0 / number of parameters
GCOUNT  =                    1 / number of groups
EXTNAME = 'SCI     '           / extension name

        Information about the coordinates in the file

RADESYS = 'ICRS    '           / Name of the coordinate reference frame

        Information about the data array

BUNIT   = 'DN      '           / Units of the data array

        Spacecraft pointing information

RA_V1   =   0.1399040200401982 / [deg] RA of telescope V1 axis
DEC_V1  =  0.08853956963814555 / [deg] Dec of telescope V1 axis
PA_V3   =                -0.0 / [deg] Position angle of telescope V3 axis

        WCS parameters

WCSAXES =                    3 / number of World Coordinate System axes
CRPIX1  =                    0 / axis 1 coordinate of the reference pixel
CRPIX2  =                    0 / axis 2 coordinate of the reference pixel
CRPIX3  =                    0 / axis 3 coordinate of the reference pixel
CRVAL1  =                  0.0 / first axis value at the reference pixel
CRVAL2  =                  0.0 / second axis value at the reference pixel
CRVAL3  =                  0.0 / third axis value at the reference pixel
CTYPE1  = '' / first axis coordinate type
CTYPE2  = '' / second axis coordinate type
CTYPE3  = '' / third axis coordinate type
CUNIT1  = '' / first axis units
CUNIT2  = '' / second axis units
CUNIT3  = '' / third axis units
CDELT1  =                  1.0 / first axis increment per pixel
CDELT2  =                  1.0 / second axis increment per pixel
CDELT3  =                  1.0 / third axis increment per pixel
V2_REF  =   -503.6544721447135 / [arcsec] Telescope v2 coordinate of the referen
```

(continues on next page)

(continued from previous page)

```
V3_REF  =    -318.742450697324 / [arcsec] Telescope v3 coordinate of the referen
VPARITY =                   -1 / Relative sense of rotation between Ideal xy and
V3I_YANG=                  0.0 / [deg] Angle from V3 axis to Ideal y axis
RA_REF  =                  0.0 / [deg] Right ascension of the reference point
DEC_REF =                  0.0 / [deg] Declination of the reference point
ROLL_REF=                 -0.0 / [deg] V3 roll angle at the ref point (N over E)
EXTVER  =                    1 / extension value
```



[ ]:

CHAPTER

FOURTEEN

APPENDIX: WALK THROUGH OF MIRISIM (IMAGER)

In this notebook, we create a scene with a circle of point sources around a central galaxy (that all fit within the Imager field of view), and run through a MIRISim simulation showing the products created at each stage.

NOTE this notebook must be started within the mirisim anaconda environment. This is a pre-requist for using MIRISim, and installation instructions can be found here.

When launching the notebook, make sure you've run the appropriate version of

conda activate mirisim

in the terminal before starting the notebook.

14.1 Steps in this notebook:

1. Create a Scene
2. Initialise the simulation parameters
3. run the simulation
4. examine some of the outputs.

[1]:
```
# import the configuration file parsers so they can be written to file
from mirisim.config_parser import SimConfig, SimulatorConfig, SceneConfig

# import scene component generators
from mirisim.skysim import Background, sed, Point, Galaxy, kinetics
from mirisim.skysim import wrap_pysynphot as wS

from mirisim import MiriSimulation

#other things to be used
import numpy as np
import glob                 # glob is used to find the output directory
import os                   # for listing directory contents
from astropy.io import fits # for reading FITS file contents

import matplotlib.pyplot as plt    # to display images
from matplotlib import colors,cm
%matplotlib inline
```

14.1.1 Create a Scene

The scene created below consists of a circle of point sources surrounding a Galaxy,and a low level background, all of which fit within the main imager field of view. The point sources all have the same (blackbody) SEDs to make things simple. The galaxy as a pysynphot SED which is modified by a Keplerian velocity field.

to be able to simply add the point sources ('stars') to a scene, it first needs to be initialised with a background

Create the background emission

[2]:
```
bg = Background(level = 'low', gradient = 5., pa = 45.)
```

```
2020-08-04 15:02:13,216 - INFO - Initializing Background
```

Create a circle of point sources

These points will be put into a 'compound target list' for simplicity later. This list needs to be initialised, so the first thing will be to create a single point source, which is placed to fall into the LRS slit.

[3]:
```
# create point source objects across the Imager Field of View
def create_star(xpos,ypos):
    star = Point(Cen = (xpos,ypos))
    BBparams = {'Temp': 1e4,
                'wref': 10.,
                'flux': 7e5}
    Blackbody = sed.BBSed(**BBparams)
    star.set_SED(Blackbody)

    return star


# create a single point source at the position of the LRS slit,
# and another at the LRS slitless position. The positions of the slit
# and slitless targets (in pixels) come from figure 2 of
# Kendrew et al. (2015)  (MIRI PASP series #4)

center_pixel = (692,512)    # center of the array (in pixel coordinates)
slit_pixel = (321,512)      # centering pixel for the slit
slitless_pixel = (1,529)    # centering pixel for slitless observations
pixel_scale = 0.11          #arcsec/pixel


slit_star = create_star(
    (center_pixel[0]-slit_pixel[0])*pixel_scale,
    (center_pixel[1]-slit_pixel[1])*pixel_scale)
slitless_star = create_star(
    (center_pixel[0]-slitless_pixel[0])*pixel_scale,
    (center_pixel[1]-slitless_pixel[1])*pixel_scale)

# add these two stars to the compound target list called 'stars'
stars = slit_star + slitless_star
# for writing to scene.ini file
targetlist = [slit_star,slitless_star]
```

(continues on next page)

(continued from previous page)

```
# create a circle of stars, and add them to the scene
IMA_radius = 73/2.  # radius of Imager (in arcsec)
Nstars = 20         # number of stars to put in the circle
for i in range(0,Nstars):
    xp = np.cos(2*np.pi/Nstars*i)*IMA_radius
    yp = np.sin(2*np.pi/Nstars*i)*IMA_radius
    star = create_star(xp,yp)
    stars += star              # for creating a scene
    targetlist.append(star)    # for writing to file
```

```
2020-08-04 15:02:13,227 - INFO - Initializing Point
2020-08-04 15:02:13,228 - INFO - Initializing Point
2020-08-04 15:02:13,229 - INFO - Initializing Point
2020-08-04 15:02:13,230 - INFO - Initializing Point
2020-08-04 15:02:13,230 - INFO - Initializing Point
2020-08-04 15:02:13,231 - INFO - Initializing Point
2020-08-04 15:02:13,231 - INFO - Initializing Point
2020-08-04 15:02:13,232 - INFO - Initializing Point
2020-08-04 15:02:13,232 - INFO - Initializing Point
2020-08-04 15:02:13,233 - INFO - Initializing Point
2020-08-04 15:02:13,233 - INFO - Initializing Point
2020-08-04 15:02:13,234 - INFO - Initializing Point
2020-08-04 15:02:13,234 - INFO - Initializing Point
2020-08-04 15:02:13,235 - INFO - Initializing Point
2020-08-04 15:02:13,236 - INFO - Initializing Point
2020-08-04 15:02:13,236 - INFO - Initializing Point
2020-08-04 15:02:13,237 - INFO - Initializing Point
2020-08-04 15:02:13,237 - INFO - Initializing Point
2020-08-04 15:02:13,238 - INFO - Initializing Point
2020-08-04 15:02:13,238 - INFO - Initializing Point
2020-08-04 15:02:13,239 - INFO - Initializing Point
2020-08-04 15:02:13,239 - INFO - Initializing Point
```

Create the Galaxy

[4]:

```
# initialise the galaxy with a position
galaxy = Galaxy(Cen = (0.,0.),n=1.,re=1.,q=0.4,pa=0)

# set the properties of the SED
PYSPsedDict = {'family': 'bkmodels', 'sedname': 'bk_b0005', 'flux': 1E+5, 'wref': 10.}
# read that dictionary into the pysynphot interpreter
sedE = wS.PYSPSed(**PYSPsedDict)
# add the SED to the galaxy
galaxy.set_SED(sedE)

# create a velocity mapping for the SED
VMAPpars = {'vrot': 200., 'Cen': (0., 0.), 'pa': 0., 'q': 0.4, 'c': 0}
VelocityMap = kinetics.FlatDisk(**VMAPpars)
# add the velocity map to the galaxy
galaxy.set_velomap(VelocityMap)

# add a line of sight velocity distribution to the galaxy
```

(continues on next page)

(continued from previous page)

```
losVeloDist = kinetics.Losvd(sigma=200.,h3=0.,h4=0.)
galaxy.set_LOSVD(losVeloDist)
```

```
2020-08-04 15:02:13,246 - INFO - Initializing Galaxy
2020-08-04 15:02:13,247 - INFO - Initializing Galaxy
2020-08-04 15:02:13,248 - INFO - Initializing Galaxy
2020-08-04 15:02:13,295 - INFO - Initializing FlatDisk
2020-08-04 15:02:13,296 - INFO - Initializing Losvd
```

14.2 From the components, create a scene

create a scene is as simple as adding together the targets.

[5]:
```
scene = bg + stars + galaxy

# for writing to scene.ini file
targetlist.append(galaxy)
```

14.3 Export the scene to an ini file and FITS

The scene can also be exported to an ini file (for future use), or a FITS file to be visualised.

Exporting to a FITS file requires specifing a number of aditional parametrs such as the Field of view, required spectral sampling, etc. They're described in more detail below. For large fields of view, and/or small spectral channels, which create large FITS files, writing to a FITS file can take a while (few minutes)

NOTE: export to fits currently (8/Feb/2017) doesn't work properly

[6]:
```
## export to ini file

scene_config = SceneConfig.makeScene(loglevel=0,
                                     background=bg,
                                     targets = targetlist)

os.system('rm IMA_example_scene.ini')
scene_config.write('IMA_example_scene.ini')

## export to FITS file
FOV = np.array([[-57.,57.],[-57.,57.]])   # field of view [xmin,xmax],[ymin,ymax] (in␣
↪arcsec)
SpatialSampling = 0.1                # spatial sampling (in arcsec)
WavelengthRange = [5,15]             # wavelength range to process (in microns)
WavelengthSampling = 0.5            # channel width (in microns)


# overwrite = True enables overwriting of any previous version of the fits file
# with the same name as that given in the writecube command
scene.writecube(cubefits = 'IMA_example_scene.fits',
```

(continues on next page)

(continued from previous page)

```
                FOV = FOV, time = 0.0,
                spatsampling = SpatialSampling,
                wrange = WavelengthRange,
                wsampling = WavelengthSampling,
                overwrite = True)

#It's not clear why the point sources are not being written to the FITS file.
```

14.3.1 Initialise the Simulation Parameters

This is where the parameters for the MRS simulation get set. Note that for internal consistency in MIRISim, all settings (including those not used in the MRS simulation here) must be set. Those not being used in this simulation are labelled with NOT USED HERE in the comments of each line

[7]:
```
sim_config = SimConfig.makeSim(
    name = 'ima_simulation',     # name given to simulation
    scene = 'IMA_example_scene.ini', # name of scene file to input
    rel_obsdate = 0.0,           # relative observation date (0 = launch, 1 = end of 5␣
↪yrs)
    POP = 'IMA',                 # Component on which to center (Imager or MRS)
    ConfigPath = 'IMA_FULL',  # Configure the Optical path (MRS sub-band)
    Dither = False,              # Don't Dither
    StartInd = 1,                # start index for dither pattern [NOT USED HERE]
    NDither = 2,                 # number of dither positions [NOT USED HERE]
    DitherPat = 'ima_recommended_dither.dat', # dither pattern to use [NOT USED HERE]
    disperser = 'SHORT',         # [NOT USED HERE]
    detector = 'SW',             # [NOT USED HERE]
    mrs_mode = 'SLOW',           # [NOT USED HERE]
    mrs_exposures = 2,           # [NOT USED HERE]
    mrs_integrations = 3,        # [NOT USED HERE]
    mrs_frames = 5,              # [NOT USED HERE]
    ima_exposures = 2,           # number of exposures
    ima_integrations = 3,        # number of integrations
    ima_frames = 5,              # number of groups (for MIRI, # Groups = # Frames)
    ima_mode = 'FAST',           # Imager read mode (default is FAST ~ 2.3 s)
    filter = 'F1130W',           # Imager Filter to use
    readDetect = 'FULL'          # Portion of detector to read out
)
```

14.4 Export the simulation setup to a file

[8]:
```
os.system('rm IMA_simulation.ini')
sim_config.write('IMA_simulation.ini')
```

14.4.1 Run the simulation

Now that the scene and the setup of the simulation have been set, we can run the simulation.

the last step is to setup the defaults for internal things like CDPs.

[9]:
```
simulator_config = SimulatorConfig.from_default()


mysim = MiriSimulation(sim_config,scene_config,simulator_config)
mysim.run()
```

```
2020-08-04 15:03:31,979 - INFO - MIRISim version: 2.3.0b0
2020-08-04 15:03:31,980 - INFO - MIRI Simulation started.
2020-08-04 15:03:31,981 - INFO - Output will be saved to: 20200804_150331_mirisim
2020-08-04 15:03:31,981 - INFO - Storing configs in output directory.
2020-08-04 15:03:33,918 - INFO - Reading cosmic ray properties from parameter file /
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪miri/simulators/scasim/cosmic_ray_properties.py
2020-08-04 15:03:33,940 - INFO - Reading detector properties from parameter file /
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪miri/simulators/scasim/detector_properties.py
2020-08-04 15:03:34,119 - INFO - Storing dither pattern in output directory.
2020-08-04 15:03:34,125 - INFO - Using $CDP_DIR for location of CDP files: /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP
2020-08-04 15:03:34,126 - INFO - Setting up simulated Observation, with following␣
↪settings:
2020-08-04 15:03:34,126 - INFO - Configuration Path: IMA_FULL
2020-08-04 15:03:34,127 - INFO - Primary optical path: IMA
2020-08-04 15:03:34,128 - INFO - IMA Filter: F1130W
2020-08-04 15:03:34,128 - INFO - IMA Subarray: FULL
2020-08-04 15:03:34,129 - INFO - IMA detector readout mode: FAST
2020-08-04 15:03:34,130 - INFO - IMA detector # exposures: 2
2020-08-04 15:03:34,130 - INFO - IMA detector # integrations: 3
2020-08-04 15:03:34,131 - INFO - IMA detector # frames: 5
2020-08-04 15:03:34,132 - INFO - Parsing: Background
2020-08-04 15:03:34,133 - INFO - Initializing Background
2020-08-04 15:03:34,133 - INFO - Parsing: point_1
2020-08-04 15:03:34,134 - INFO - Initializing Point
2020-08-04 15:03:34,135 - INFO - Parsing: point_2
2020-08-04 15:03:34,135 - INFO - Initializing Point
2020-08-04 15:03:34,136 - INFO - Parsing: point_3
2020-08-04 15:03:34,137 - INFO - Initializing Point
2020-08-04 15:03:34,137 - INFO - Parsing: point_4
2020-08-04 15:03:34,138 - INFO - Initializing Point
2020-08-04 15:03:34,139 - INFO - Parsing: point_5
2020-08-04 15:03:34,139 - INFO - Initializing Point
2020-08-04 15:03:34,140 - INFO - Parsing: point_6
2020-08-04 15:03:34,141 - INFO - Initializing Point
2020-08-04 15:03:34,141 - INFO - Parsing: point_7
2020-08-04 15:03:34,142 - INFO - Initializing Point
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:03:34,143 - INFO - Parsing: point_8
2020-08-04 15:03:34,143 - INFO - Initializing Point
2020-08-04 15:03:34,144 - INFO - Parsing: point_9
2020-08-04 15:03:34,144 - INFO - Initializing Point
2020-08-04 15:03:34,145 - INFO - Parsing: point_10
2020-08-04 15:03:34,146 - INFO - Initializing Point
2020-08-04 15:03:34,146 - INFO - Parsing: point_11
2020-08-04 15:03:34,147 - INFO - Initializing Point
2020-08-04 15:03:34,147 - INFO - Parsing: point_12
2020-08-04 15:03:34,148 - INFO - Initializing Point
2020-08-04 15:03:34,149 - INFO - Parsing: point_13
2020-08-04 15:03:34,149 - INFO - Initializing Point
2020-08-04 15:03:34,150 - INFO - Parsing: point_14
2020-08-04 15:03:34,150 - INFO - Initializing Point
2020-08-04 15:03:34,151 - INFO - Parsing: point_15
2020-08-04 15:03:34,152 - INFO - Initializing Point
2020-08-04 15:03:34,152 - INFO - Parsing: point_16
2020-08-04 15:03:34,153 - INFO - Initializing Point
2020-08-04 15:03:34,153 - INFO - Parsing: point_17
2020-08-04 15:03:34,154 - INFO - Initializing Point
2020-08-04 15:03:34,155 - INFO - Parsing: point_18
2020-08-04 15:03:34,155 - INFO - Initializing Point
2020-08-04 15:03:34,156 - INFO - Parsing: point_19
2020-08-04 15:03:34,157 - INFO - Initializing Point
2020-08-04 15:03:34,157 - INFO - Parsing: point_20
2020-08-04 15:03:34,158 - INFO - Initializing Point
2020-08-04 15:03:34,158 - INFO - Parsing: point_21
2020-08-04 15:03:34,159 - INFO - Initializing Point
2020-08-04 15:03:34,160 - INFO - Parsing: point_22
2020-08-04 15:03:34,160 - INFO - Initializing Point
2020-08-04 15:03:34,161 - INFO - Parsing: galaxy_1
2020-08-04 15:03:34,161 - INFO - Initializing Galaxy
2020-08-04 15:03:34,162 - INFO - Initializing Galaxy
2020-08-04 15:03:34,162 - INFO - Initializing Galaxy
2020-08-04 15:03:34,215 - INFO - Initializing FlatDisk
2020-08-04 15:03:34,216 - INFO - Initializing Losvd
2020-08-04 15:03:34,224 - INFO - Simulating a single pointing.
2020-08-04 15:03:37,871 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-08-04 15:03:39,090 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪gwcs/wcs.py:131: VisibleDeprecationWarning: Creating an ndarray from ragged nested␣
↪sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different␣
↪lengths or shapes) is deprecated. If you meant to do this, you must specify
↪'dtype=object' when creating the ndarray
  transforms = np.array(self._pipeline[from_ind: to_ind])[:, 1].copy()
2020-08-04 15:03:39,209 - INFO - Creating pointing for position 1
2020-08-04 15:03:39,210 - INFO - Creating exposure event for position 1
2020-08-04 15:03:39,212 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-08-04 15:03:39,332 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-08-04 15:03:39,443 - INFO - Observation simulation started.
2020-08-04 15:03:39,444 - INFO - Simulating ExposureEvent for pointing 1
2020-08-04 15:03:39,445 - INFO - Simulating Imager exposures for pointing 1
2020-08-04 15:03:39,446 - INFO - Simulating detector illumination for Imager␣
↪exposures for pointing 1
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:03:39,447 - INFO - Running ImSim.
2020-08-04 15:03:39,448 - INFO - Running MirimImager (version=36) for subarray=FULL
2020-08-04 15:03:39,451 - INFO - Reading 'AREA' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_AREA_07.00.00.fits'
2020-08-04 15:03:39,522 - INFO - Retrieving PCE CDP for filter : F1130W
2020-08-04 15:03:39,525 - INFO - Reading 'PCE' model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_F1130W_PCE_07.00.00.fits'
2020-08-04 15:03:39,609 - INFO - Reading 'PSF' model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_F1130W_PSF_07.02.00.fits'
2020-08-04 15:03:39,892 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-08-04 15:03:40,003 - INFO - Processing point sources for IMAGER
/Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪gwcs/wcs.py:126: VisibleDeprecationWarning: Creating an ndarray from ragged nested␣
↪sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different␣
↪lengths or shapes) is deprecated. If you meant to do this, you must specify
↪'dtype=object' when creating the ndarray
  transforms = np.array(self._pipeline[to_ind: from_ind])[:, 1].tolist()
2020-08-04 15:04:22,435 - INFO - No SkyCube
2020-08-04 15:04:22,441 - INFO - Reading 'SKYFLAT' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_F1130W_SKYFLAT_07.00.00.fits'
2020-08-04 15:04:22,679 - INFO - Apply Footprint
2020-08-04 15:04:22,680 - INFO - Retrieving Foot Print from PIXELFLAT
2020-08-04 15:04:22,682 - INFO - Reading 'PIXELFLAT' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_SLOW_F1000W_PIXELFLAT_07.01.01.
↪fits'
2020-08-04 15:04:22,901 - INFO - Wrote illumination model: 20200804_150331_mirisim/
↪illum_models/illum_model_seq1_MIRIMAGE_F1130W.fits
2020-08-04 15:04:22,901 - INFO - Simulating integrated detector images for Imager␣
↪exposures for pointing 1
2020-08-04 15:04:22,903 - INFO - Simulating Imager exposure 1
2020-08-04 15:04:22,907 - INFO - Running SCASim
2020-08-04 15:04:22,908 - INFO - Simulating  detector readout for MIRIMAGE from␣
↪illumination data model of shape (1, 1024, 1032).
2020-08-04 15:04:22,920 - INFO - Results will be returned in an exposure data model.
2020-08-04 15:04:22,932 - INFO -   Detector readout mode is FAST (samplesum=1,␣
↪sampleskip=0, nframe=1, groupgap=0)
2020-08-04 15:04:22,933 - INFO -   with 3 integrations and ngroups=5 defined␣
↪explicitly.
2020-08-04 15:04:22,933 - INFO -   Detector subarray mode is FULL.
2020-08-04 15:04:22,934 - INFO - Detector temperature = 6.70 K (which affects dark␣
↪current and read noise).
2020-08-04 15:04:22,935 - INFO - Cosmic ray environment is SOLAR_MIN.
2020-08-04 15:04:22,935 - INFO - Reading cosmic ray library file: '/Users/
↪pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/
↪simulators/data/cosmic_rays/CRs_SiAs_470_SUNMIN_01.fits'
2020-08-04 15:04:22,968 - INFO - Simulation control flags:
2020-08-04 15:04:22,969 - INFO -        Quantum efficiency simulation turned OFF.
2020-08-04 15:04:22,969 - INFO -        Poisson noise simulation turned ON.
2020-08-04 15:04:22,970 - INFO -        Read noise simulation turned ON.
2020-08-04 15:04:22,971 - INFO -        Reference pixels simulation turned ON.
2020-08-04 15:04:22,972 - INFO -        Bad pixels simulation turned ON.
2020-08-04 15:04:22,973 - INFO -        Dark current simulation turned ON.
2020-08-04 15:04:22,973 - INFO -        Flat-field simulation turned ON.
2020-08-04 15:04:22,974 - INFO -        Amplifier bias and gain turned ON.
2020-08-04 15:04:22,975 - INFO -        Detector non-linearity effects turned ON.
2020-08-04 15:04:22,975 - INFO -        Detector drift effects turned ON.
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:04:22,976 - INFO -        Detector latency effects turned ON.
2020-08-04 15:04:22,977 - WARNING - ***Illumination map of size 1024 x 1032 is too␣
↪large! Truncating to detector size of 1024 x 1024 pixels.
2020-08-04 15:04:22,979 - INFO - Input subarray mode obtained from illumination map:␣
↪FULL
2020-08-04 15:04:22,980 - INFO - Detector properties translates input subarray FULL␣
↪into None
2020-08-04 15:04:22,983 - INFO - Creating a new detector object for 1024 rows x 1024␣
↪columns.
2020-08-04 15:04:22,989 - INFO - Reading 'MASK' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_MASK_07.02.01.fits'
2020-08-04 15:04:23,083 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits'
2020-08-04 15:04:25,556 - INFO - Reading DARK model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_DARK_06.01.00.fits'
2020-08-04 15:04:28,241 - INFO - Reading 'PIXELFLAT' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_F1130W_PIXELFLAT_07.01.01.
↪fits'
2020-08-04 15:04:28,354 - INFO - Reading 'LINEARITY' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_LINEARITY_06.02.00.fits'
2020-08-04 15:04:28,616 - INFO - Reading 'READNOISE' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_READNOISE_07.01.00.fits'
2020-08-04 15:04:28,676 - INFO - Creating exposure_data with 1280 rows x 1032 columns␣
↪plus 5 groups and 3 ints.
2020-08-04 15:04:29,163 - INFO - Simulating 3 integrations.
2020-08-04 15:04:29,171 - WARNING - ***Input flux array contains values that could␣
↪saturate at least 163 pixels.
2020-08-04 15:04:29,172 - WARNING -    Maximum flux of 232682 photons/s/pixel is␣
↪greater than first frame saturation level of 127825 photons/s/pixel.
2020-08-04 15:04:29,723 - INFO - Simulating 5 groups for integration 1.
2020-08-04 15:04:31,065 - INFO - Simulating 5 groups for integration 2.
2020-08-04 15:04:32,567 - INFO - Simulating 5 groups for integration 3.
2020-08-04 15:04:33,760 - INFO - Adding the DARK calibration from /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_DARK_06.01.00.fits
2020-08-04 15:04:33,971 - INFO - Correcting nonlinearity from MIRI_FM_MIRIMAGE_
↪LINEARITY_06.02.00.fits
2020-08-04 15:04:34,443 - INFO - Output subarray undefined or FULL. SUBSTRT=(1,1),␣
↪SUBSIZE=(1032,1024)
2020-08-04 15:04:34,444 - INFO - WCS keywords defined as CRPIX1=0, CRPIX2=0
2020-08-04 15:04:38,196 - INFO - Exposure time 41.63s (duration 42.68s)
2020-08-04 15:04:39,261 - INFO - Wrote detector image: 20200804_150331_mirisim/det_
↪images/det_image_seq1_MIRIMAGE_F1130Wexp1.fits
2020-08-04 15:04:39,261 - INFO - Simulating Imager exposure 2
2020-08-04 15:04:39,275 - INFO - Running SCASim
2020-08-04 15:04:39,276 - INFO - Simulating  detector readout for MIRIMAGE from␣
↪illumination data model of shape (1, 1024, 1024).
2020-08-04 15:04:39,289 - INFO - Results will be returned in an exposure data model.
2020-08-04 15:04:39,301 - INFO -   Detector readout mode is FAST (samplesum=1,␣
↪sampleskip=0, nframe=1, groupgap=0)
2020-08-04 15:04:39,301 - INFO -   with 3 integrations and ngroups=5 defined␣
↪explicitly.
2020-08-04 15:04:39,302 - INFO -   Detector subarray mode is FULL.
2020-08-04 15:04:39,302 - INFO - Detector temperature = 6.70 K (which affects dark␣
↪current and read noise).
2020-08-04 15:04:39,303 - INFO - Cosmic ray environment is SOLAR_MIN. (No change.)
2020-08-04 15:04:39,303 - INFO - Simulation control flags:
2020-08-04 15:04:39,304 - INFO -        Quantum efficiency simulation turned OFF.
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:04:39,304 - INFO -        Poisson noise simulation turned ON.
2020-08-04 15:04:39,305 - INFO -        Read noise simulation turned ON.
2020-08-04 15:04:39,305 - INFO -        Reference pixels simulation turned ON.
2020-08-04 15:04:39,306 - INFO -        Bad pixels simulation turned ON.
2020-08-04 15:04:39,306 - INFO -        Dark current simulation turned ON.
2020-08-04 15:04:39,307 - INFO -        Flat-field simulation turned ON.
2020-08-04 15:04:39,307 - INFO -        Amplifier bias and gain turned ON.
2020-08-04 15:04:39,308 - INFO -        Detector non-linearity effects turned ON.
2020-08-04 15:04:39,309 - INFO -        Detector drift effects turned ON.
2020-08-04 15:04:39,310 - INFO -        Detector latency effects turned ON.
2020-08-04 15:04:39,310 - INFO - Input subarray mode obtained from illumination map:␣
↪FULL
2020-08-04 15:04:39,311 - INFO - Detector properties translates input subarray FULL␣
↪into None
2020-08-04 15:04:39,313 - INFO - Reading 'MASK' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_MASK_07.02.01.fits'
2020-08-04 15:04:39,365 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits'
2020-08-04 15:04:39,405 - INFO - Reading DARK model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_DARK_06.01.00.fits'
2020-08-04 15:04:41,705 - INFO - Reading 'PIXELFLAT' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_F1130W_PIXELFLAT_07.01.01.
↪fits'
2020-08-04 15:04:41,792 - INFO - Reading 'LINEARITY' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_LINEARITY_06.02.00.fits'
2020-08-04 15:04:41,995 - INFO - Reading 'READNOISE' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_READNOISE_07.01.00.fits'
2020-08-04 15:04:42,036 - INFO - Creating exposure_data with 1280 rows x 1032 columns␣
↪plus 5 groups and 3 ints.
2020-08-04 15:04:42,207 - INFO - Simulating 3 integrations.
2020-08-04 15:04:42,217 - WARNING - ***Input flux array contains values that could␣
↪saturate at least 163 pixels.
2020-08-04 15:04:42,218 - WARNING -    Maximum flux of 232682 photons/s/pixel is␣
↪greater than first frame saturation level of 127825 photons/s/pixel.
2020-08-04 15:04:42,266 - INFO - Simulating 5 groups for integration 1.
2020-08-04 15:04:43,495 - INFO - Simulating 5 groups for integration 2.
2020-08-04 15:04:44,727 - INFO - Simulating 5 groups for integration 3.
2020-08-04 15:04:45,897 - INFO - Adding the DARK calibration from /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_DARK_06.01.00.fits
2020-08-04 15:04:46,058 - INFO - Correcting nonlinearity from MIRI_FM_MIRIMAGE_
↪LINEARITY_06.02.00.fits
2020-08-04 15:04:46,487 - INFO - Output subarray undefined or FULL. SUBSTRT=(1,1),␣
↪SUBSIZE=(1032,1024)
2020-08-04 15:04:46,488 - INFO - WCS keywords defined as CRPIX1=0, CRPIX2=0
2020-08-04 15:04:49,925 - INFO - Exposure time 41.63s (duration 42.68s)
2020-08-04 15:04:51,027 - INFO - Wrote detector image: 20200804_150331_mirisim/det_
↪images/det_image_seq1_MIRIMAGE_F1130Wexp2.fits
2020-08-04 15:04:51,028 - INFO - MIRI Simulation finished. Results have been saved to:
↪ 20200804_150331_mirisim
```

14.4.2 Examine some of the results

Now that the MIRISim simulation has completed, lets examine the results.

The first thing to note is that the outputs are placed in a date-labelled directory taking the form YYYYMMDD_hhmmss_mirisim. The name of the output directory is given in the last line of the MIRISim log above. Because the output directory is date-labelled, we can quantify which was the most recent run of MIRISim, and find its output directory using glob.glob

[10]:
```
outputdir = sorted(glob.glob('*_*_mirisim'),key=os.path.getmtime)[-1]    #[-1] takes␣
↪the last entry found

outputDirContents = os.listdir(outputdir)

directories = [name for name in outputDirContents if os.path.isdir(os.path.
↪join(outputdir,name))]
files = [name for name in outputDirContents if not os.path.isdir(os.path.
↪join(outputdir,name))]

print('The subdirectories in the outputdirectory are:\n{}'.format(directories))
print('The files in the outputdirectory are:\n{}'.format(files))
```

```
The subdirectories in the outputdirectory are:
['det_images', 'illum_models']
The files in the outputdirectory are:
['simulator.ini', 'ima_recommended_dither.dat', 'scene.ini', 'simulation.ini',
↪'mirisim.log']
```

The files contain the log which was also output to the terminal (mirisim.log) and copies of the .ini files used (or created from python inputs) to create the simulation. These versions of the .ini files can be used to re-create the run of the simulation

The directories contain various outputs of MIRISim:

- ** illum_models ** houses FITS images of the illuminations sent to the detector (sent to SCAsim - the simulator of the Sensor Chip Assembly). This should have the same format as the detector image, but without all of the detector effects and noise. There are FITS files produced for each exposure and dither position
- ** det_images ** houses FITS images of the final outputs of MIRISim. These detector images have all of the detector effects and noise incorporated. The number of detector images should be the same as the number of illumination models.

The headers of the detector images are formatted for ingest into the JWST pipeline.

For those on the MIRI Test Team, the following link takes you to Patrick Kavanagh's ipython notebook walkthrough of the current build of the JWST pipeline

(http://miri.ster.kuleuven.be/bin/view/Internal/TutorialsPipeline)

Which is also linked from the MIRISim twiki page.

Below is a small code snippet used to draw the images. it needs to be run, but doesn't produce any output directly (it's called later to show the output images)

[11]:
```
def show_outputs(MIRISim_outputdir,output_type):
    '''
    plot the specified channel of the MIRISim outputs
    :param MIRISim_outputdir:
        name of the date-labelled dir. holding the MIRISIM outputs
    :param output_type:
```

(continues on next page)

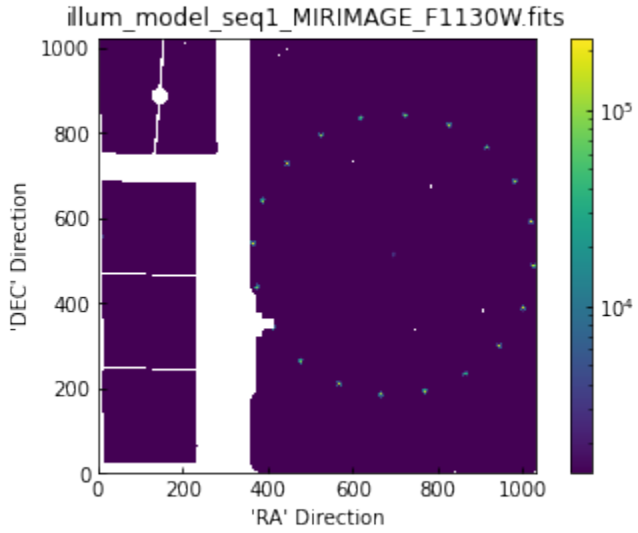
(continued from previous page)

```
        type of output to process
        (e.g. illum_models, det_images or skycubes)
    '''

    infits = glob.glob('{}/{}/*.fits'.format(MIRISim_outputdir,output_type))[0]

    hdulist = fits.open(infits)


    hdu_index = 1
    if len(hdulist[hdu_index].data.shape) > 3:
        integ,frames,nx,ny = hdulist[hdu_index].data.shape
        image = hdulist[hdu_index].data[integ-1,frames-1,:,:]


    else:
        image = hdulist[hdu_index].data[0,:,:]

    norm = colors.LogNorm(image.mean() + 0.5 * image.std(), image.max(), clip=True)
    plt.imshow(image,origin = 'lower', cmap = cm.viridis,interpolation = 'nearest',
↪norm = norm)
    plt.title('{}'.format(infits.split('/')[-1]))
    plt.xlabel("'RA' Direction")
    plt.ylabel("'DEC' Direction")


    plt.colorbar()
```

14.5 Viewing an illumination model

Below shows an example of an output illumination model

[12]:
```
show_outputs(outputdir,'illum_models')
```



14.5.1 Viewing the final detector image

The detector images are the final output of MIRISim, and have data structures and formatting consistent with what will come from MIRI itself. The data format is JWST pipeline ready. Below an example image of the last frame of the last integration is shown. The units are DN.

Additionally, below the FITS header information for the SCIENCE extension is listed

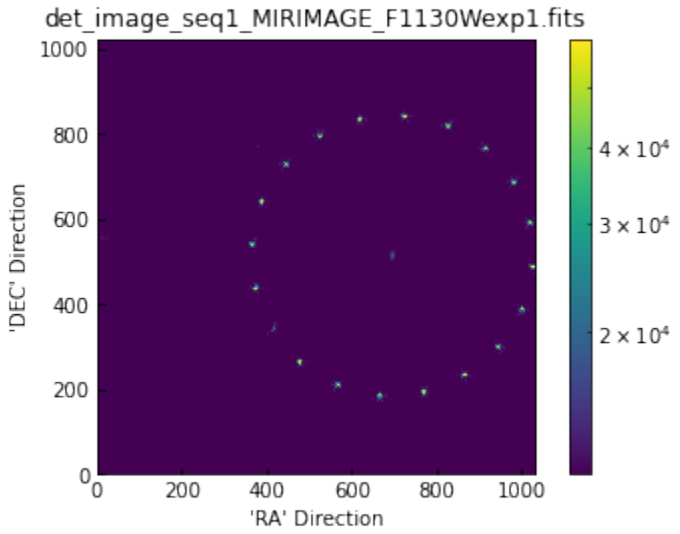
[13]:
```
show_outputs(outputdir,'det_images')

infits = glob.glob('{}/det_images/*.fits'.format(outputdir))[0]
hdulist = fits.open(infits)
hdulist[1].header
```

[13]:
```
XTENSION= 'IMAGE   '           / Image extension
BITPIX  =                  -32 / array data type
NAXIS   =                    4 / number of array dimensions
NAXIS1  =                 1032
NAXIS2  =                 1024
NAXIS3  =                    5
NAXIS4  =                    3
PCOUNT  =                    0 / number of parameters
GCOUNT  =                    1 / number of groups
EXTNAME = 'SCI     '           / extension name

        Information about the coordinates in the file

RADESYS = 'ICRS    '           / Name of the coordinate reference frame

        Information about the data array

BUNIT   = 'DN      '           / Units of the data array

        Spacecraft pointing information

RA_V1   =   0.1259948156684385 / [deg] RA of telescope V1 axis
```

(continues on next page)

(continued from previous page)

```
DEC_V1  =   0.1038551438193729 / [deg] Dec of telescope V1 axis
PA_V3   =                 -0.0 / [deg] Position angle of telescope V3 axis

        WCS parameters

WCSAXES =                    2 / number of World Coordinate System axes
CRPIX1  =                    0 / axis 1 coordinate of the reference pixel
CRPIX2  =                    0 / axis 2 coordinate of the reference pixel
CRVAL1  =                  0.0 / first axis value at the reference pixel
CRVAL2  =                  0.0 / second axis value at the reference pixel
CTYPE1  = 'RA---TAN'           / first axis coordinate type
CTYPE2  = 'DEC--TAN'           / second axis coordinate type
CUNIT1  = 'deg     '           / first axis units
CUNIT2  = 'deg     '           / second axis units
CDELT1  = 3.06830555555555E-05 / first axis increment per pixel
CDELT2  = 3.09669444444444E-05 / second axis increment per pixel
PC1_1   =                 -1.0 / linear transformation matrix element
PC1_2   =                  0.0 / linear transformation matrix element
PC2_1   =                  0.0 / linear transformation matrix element
PC2_2   =                  1.0 / linear transformation matrix element
V2_REF  =   -453.5591160661637 / [arcsec] Telescope v2 coordinate of the referen
V3_REF  =   -373.8144468729251 / [arcsec] Telescope v3 coordinate of the referen
VPARITY =                   -1 / Relative sense of rotation between Ideal xy and
V3I_YANG=               5.0152 / [deg] Angle from V3 axis to Ideal y axis
RA_REF  = 6.17231672631785E-06 / [deg] Right ascension of the reference point
DEC_REF = 1.77974657825706E-05 / [deg] Declination of the reference point
ROLL_REF=                 -0.0 / [deg] V3 roll angle at the ref point (N over E)
EXTVER  =                    1 / extension value
```



[ ]:

CHAPTER

FIFTEEN

APPENDIX: WALK THROUGH OF MIRISIM (LRS-SLIT)

In this notebook, we create a scene with a circle of point sources around a central galaxy (that all fit within the Imager field of view), and run through a MIRISim simulation showing the products created at each stage.

NOTE this notebook must be started within the mirisim anaconda environment. This is a pre-requist for using MIRISim, and installation instructions can be found here.

When launching the notebook, make sure you've run the appropriate version of

conda activate mirisim

in the terminal before starting the notebook.

15.1 Steps in this notebook:

1. Create a Scene
2. Initialise the simulation parameters
3. run the simulation
4. examine some of the outputs.

[1]:
```
# import the configuration file parsers so they can be written to file
from mirisim.config_parser import SimConfig, SimulatorConfig, SceneConfig

# import scene component generators
from mirisim.skysim import Background, sed, Point, Galaxy, kinetics
from mirisim.skysim import wrap_pysynphot as wS

from mirisim import MiriSimulation

#other things to be used
import numpy as np
import glob                 # glob is used to find the output directory
import os                   # for listing directory contents
from astropy.io import fits # for reading FITS file contents

import matplotlib.pyplot as plt    # to display images
from matplotlib import colors,cm
%matplotlib inline
```

15.1.1 Create a Scene

The scene created below consists of a circle of point sources surrounding a Galaxy,and a low level background, all of which fit within the main imager field of view. The point sources all have the same (blackbody) SEDs to make things simple. The galaxy as a pysynphot SED which is modified by a Keplerian velocity field.

to be able to simply add the point sources ('stars') to a scene, it first needs to be initialised with a background

Create the background emission

[2]:
```
bg = Background(level = 'low', gradient = 5., pa = 45.)
```

```
2020-08-04 15:05:19,381 - INFO - Initializing Background
```

Create a circle of point sources

These points will be put into a 'compound target list' for simplicity later. This list needs to be initialised, so the first thing will be to create a single point source, which is placed to fall into the LRS slit.

[3]:
```
# create point source objects across the Imager Field of View
def create_star(xpos,ypos):
    star = Point(Cen = (xpos,ypos))
    BBparams = {'Temp': 1e4,
                'wref': 10.,
                'flux': 7e8}
    Blackbody = sed.BBSed(**BBparams)
    star.set_SED(Blackbody)

    return star


# create a single point source at the position of the LRS slit,
# and another at the LRS slitless position. The positions of the slit
# and slitless targets (in pixels) come from figure 2 of
# Kendrew et al. (2015)  (MIRI PASP series #4)

center_pixel = (692,512)    # center of the array (in pixel coordinates)
slit_pixel = (321,512)      # centering pixel for the slit
slitless_pixel = (1,529)    # centering pixel for slitless observations
pixel_scale = 0.11          #arcsec/pixel


slit_star = create_star(
    (center_pixel[0]-slit_pixel[0])*pixel_scale,
    (center_pixel[1]-slit_pixel[1])*pixel_scale)
slitless_star = create_star(
    (center_pixel[0]-slitless_pixel[0])*pixel_scale,
    (center_pixel[1]-slitless_pixel[1])*pixel_scale)

# add these two stars to the compound target list called 'stars'
stars = slit_star + slitless_star
# for writing to scene.ini file
targetlist = [slit_star,slitless_star]
```

```
2020-08-04 15:05:19,390 - INFO - Initializing Point
2020-08-04 15:05:19,391 - INFO - Initializing Point
```

Create the Galaxy

[4]:
```
# initialise the galaxy with a position
galaxy = Galaxy(Cen = (0.,0.),n=1.,re=1.,q=0.4,pa=0)

# set the properties of the SED
PYSPsedDict = {'family': 'bkmodels', 'sedname': 'bk_b0005', 'flux': 1E+5, 'wref': 10.}
# read that dictionary into the pysynphot interpreter
sedE = wS.PYSPSed(**PYSPsedDict)
# add the SED to the galaxy
galaxy.set_SED(sedE)

# create a velocity mapping for the SED
VMAPpars = {'vrot': 200., 'Cen': (0., 0.), 'pa': 0., 'q': 0.4, 'c': 0}
VelocityMap = kinetics.FlatDisk(**VMAPpars)
# add the velocity map to the galaxy
galaxy.set_velomap(VelocityMap)

# add a line of sight velocity distribution to the galaxy
losVeloDist = kinetics.Losvd(sigma=200.,h3=0.,h4=0.)
galaxy.set_LOSVD(losVeloDist)
```

```
2020-08-04 15:05:19,399 - INFO - Initializing Galaxy
2020-08-04 15:05:19,400 - INFO - Initializing Galaxy
2020-08-04 15:05:19,400 - INFO - Initializing Galaxy
2020-08-04 15:05:19,451 - INFO - Initializing FlatDisk
2020-08-04 15:05:19,452 - INFO - Initializing Losvd
```

15.2 From the components, create a scene

create a scene is as simple as adding together the targets.

[5]:
```
scene = bg + stars + galaxy

# for writing to scene.ini file
targetlist.append(galaxy)
```

15.3 Export the scene to an ini file and FITS

The scene can also be exported to an ini file (for future use), or a FITS file to be visualised.

Exporting to a FITS file requires specifing a number of additional parametrs such as the Field of View, required spectral sampling, etc. They're described in more detail below. For large fields of view, and/or small spectral channels, which create large FITS files, writing to a FITS file can take a while (few minutes)

[6]:
```
## export to ini file

scene_config = SceneConfig.makeScene(loglevel=0,
                                     background=bg,
                                     targets = targetlist)

os.system('rm LRS_example_scene.ini')
scene_config.write('LRS_example_scene.ini')

## export to FITS file
FOV = np.array([[-57.,57.],[-57.,57.]])   # field of view [xmin,xmax],[ymin,ymax] (in␣
↪arcsec)
SpatialSampling = 0.1                # spatial sampling (in arcsec)
WavelengthRange = [5,10]             # wavelength range to process (in microns)
WavelengthSampling = 0.2            # channel width (in microns)


# overwrite = True enables overwriting of any previous version of the fits file
# with the same name as that given in the writecube command
scene.writecube(cubefits = 'LRS_example_scene.fits',
                FOV = FOV, time = 0.0,
                spatsampling = SpatialSampling,
                wrange = WavelengthRange,
                wsampling = WavelengthSampling,
                overwrite = True)
```

15.3.1 Initialise the Simulation Parameters

This is where the parameters for the MRS simulation get set. Note that for internal consistency in MIRISim, all settings (including those not used in the MRS simulation here) must be set. Those not being used in this simulation are labelled with NOT USED HERE in the comments of each line

[7]:
```
sim_config = SimConfig.makeSim(
    name = 'ima_simulation',     # name given to simulation
    scene = 'LRS_example_scene.ini', # name of scene file to input
    rel_obsdate = 0.0,           # relative observation date (0 = launch, 1 = end of 5␣
↪yrs)
    POP = 'IMA',                 # Component on which to center (Imager or MRS)
    ConfigPath = 'LRS_SLIT',  # Configure the Optical path (MRS sub-band)
    Dither = False,              # Don't Dither
    StartInd = 1,                # start index for dither pattern [NOT USED HERE]
    NDither = 2,                 # number of dither positions [NOT USED HERE]
    DitherPat = 'ima_recommended_dither.dat', # dither pattern to use [NOT USED HERE]
    disperser = 'SHORT',         # [NOT USED HERE]
```

(continues on next page)

(continued from previous page)

```
    detector = 'SW',              # [NOT USED HERE]
    mrs_mode = 'SLOW',            # [NOT USED HERE]
    mrs_exposures = 2,            # [NOT USED HERE]
    mrs_integrations = 3,         # [NOT USED HERE]
    mrs_frames = 5,               # [NOT USED HERE]
    ima_exposures = 1,            # number of exposures
    ima_integrations = 1,         # number of integrations
    ima_frames = 5,               # number of groups (for MIRI, # Groups = # Frames)
    ima_mode = 'FAST',            # Imager read mode (default is FAST ~ 2.3 s)
    filter = 'P750L',            # Imager Filter to use
    readDetect = 'FULL'           # Portion of detector to read out
)
```

15.4 Export the simulation setup to a file

[8]:
```
os.system('rm LRS_simulation.ini')
sim_config.write('LRS_simulation.ini')
```

15.4.1 Run the simulation

Now that the scene and the setup of the simulation have been set, we can run the simulation.

the last step is to setup the defaults for internal things like CDPs.

[9]:
```
simulator_config = SimulatorConfig.from_default()


mysim = MiriSimulation(sim_config,scene_config,simulator_config,loglevel='DEBUG')
mysim.run()
```

```
2020-08-04 15:06:51,457 - INFO - MIRISim version: 2.3.0b0
2020-08-04 15:06:51,459 - INFO - MIRI Simulation started.
2020-08-04 15:06:51,459 - INFO - Output will be saved to: 20200804_150651_mirisim
2020-08-04 15:06:51,460 - INFO - Storing configs in output directory.
2020-08-04 15:06:52,124 - miri.simulators - INFO - Reading cosmic ray properties from␣
↪parameter file /Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/
↪site-packages/miri/simulators/scasim/cosmic_ray_properties.py
2020-08-04 15:06:52,129 - miri.simulators - INFO - Reading detector properties from␣
↪parameter file /Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/
↪site-packages/miri/simulators/scasim/detector_properties.py
2020-08-04 15:06:52,134 - miri.simulators - DEBUG - Found simulator file: dark_
↪currentIM.fits
2020-08-04 15:06:52,135 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/dark_currentIM.fits
2020-08-04 15:06:52,136 - miri.simulators - DEBUG - Found simulator file: qe_
↪measurementIM.fits
2020-08-04 15:06:52,137 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/qe_measurementIM.fits
2020-08-04 15:06:52,139 - miri.simulators - DEBUG - Found simulator file: dark_
↪currentLW.fits
2020-08-04 15:06:52,140 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/dark_currentLW.fits
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:06:52,143 - miri.simulators - DEBUG - Found simulator file: qe_
↪measurementLW.fits
2020-08-04 15:06:52,143 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/qe_measurementLW.fits
2020-08-04 15:06:52,145 - miri.simulators - DEBUG - Found simulator file: dark_
↪currentSW.fits
2020-08-04 15:06:52,146 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/dark_currentSW.fits
2020-08-04 15:06:52,149 - miri.simulators - DEBUG - Found simulator file: qe_
↪measurementSW.fits
2020-08-04 15:06:52,149 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/qe_measurementSW.fits
2020-08-04 15:06:52,154 - miri.simulators - DEBUG - Found simulator file: dark_
↪currentIM.fits
2020-08-04 15:06:52,155 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/dark_currentIM.fits
2020-08-04 15:06:52,157 - miri.simulators - DEBUG - Found simulator file: qe_
↪measurementIM.fits
2020-08-04 15:06:52,157 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/qe_measurementIM.fits
2020-08-04 15:06:52,160 - miri.simulators - DEBUG - Found simulator file: dark_
↪currentLW.fits
2020-08-04 15:06:52,161 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/dark_currentLW.fits
2020-08-04 15:06:52,163 - miri.simulators - DEBUG - Found simulator file: qe_
↪measurementLW.fits
2020-08-04 15:06:52,164 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/qe_measurementLW.fits
2020-08-04 15:06:52,166 - miri.simulators - DEBUG - Found simulator file: dark_
↪currentSW.fits
2020-08-04 15:06:52,167 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/dark_currentSW.fits
2020-08-04 15:06:52,170 - miri.simulators - DEBUG - Found simulator file: qe_
↪measurementSW.fits
2020-08-04 15:06:52,171 - miri.simulators - DEBUG -     at /Users/pamelaklaassen/opt/
↪anaconda3/envs/miricle.test/lib/python3.7/site-packages/miri/simulators/data/
↪detector/qe_measurementSW.fits
2020-08-04 15:06:52,202 - INFO - Storing dither pattern in output directory.
2020-08-04 15:06:52,203 - INFO - Using $CDP_DIR for location of CDP files: /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP
2020-08-04 15:06:52,204 - mirisim.obssim.obssim - DEBUG - Initializing␣
↪ObservationSimulation
2020-08-04 15:06:52,205 - mirisim.obssim.obssim - INFO - Setting up simulated␣
↪Observation, with following settings:
2020-08-04 15:06:52,205 - mirisim.obssim.obssim - INFO - Configuration Path: LRS_SLIT
2020-08-04 15:06:52,206 - mirisim.obssim.obssim - INFO - Primary optical path: IMA
2020-08-04 15:06:52,207 - mirisim.obssim.obssim - INFO - IMA Filter: P750L
2020-08-04 15:06:52,208 - mirisim.obssim.obssim - INFO - IMA Subarray: FULL
2020-08-04 15:06:52,209 - mirisim.obssim.obssim - INFO - IMA detector readout mode:␣
↪FAST
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:06:52,209 - mirisim.obssim.obssim - INFO - IMA detector # exposures: 1
2020-08-04 15:06:52,210 - mirisim.obssim.obssim - INFO - IMA detector # integrations:␣
↪1
2020-08-04 15:06:52,211 - mirisim.obssim.obssim - INFO - IMA detector # frames: 5
2020-08-04 15:06:52,212 - INFO - Parsing: Background
2020-08-04 15:06:52,213 - INFO - Initializing Background
2020-08-04 15:06:52,213 - INFO - Parsing: point_1
2020-08-04 15:06:52,214 - INFO - Initializing Point
2020-08-04 15:06:52,215 - INFO - Parsing: point_2
2020-08-04 15:06:52,216 - INFO - Initializing Point
2020-08-04 15:06:52,217 - INFO - Parsing: galaxy_1
2020-08-04 15:06:52,219 - INFO - Initializing Galaxy
2020-08-04 15:06:52,220 - INFO - Initializing Galaxy
2020-08-04 15:06:52,220 - INFO - Initializing Galaxy
2020-08-04 15:06:52,421 - INFO - Initializing FlatDisk
2020-08-04 15:06:52,422 - INFO - Initializing Losvd
2020-08-04 15:06:52,430 - mirisim.obssim.clock - DEBUG - Initializing Clock
2020-08-04 15:06:52,432 - mirisim.obssim.obssim - INFO - Simulating a single pointing.
2020-08-04 15:06:56,975 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.
↪02.00.fits'
2020-08-04 15:06:57,732 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.
↪fits'
2020-08-04 15:06:57,881 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.
↪fits'
2020-08-04 15:06:57,998 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.
↪02.00.fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪gwcs/wcs.py:131: VisibleDeprecationWarning: Creating an ndarray from ragged nested␣
↪sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different␣
↪lengths or shapes) is deprecated. If you meant to do this, you must specify
↪'dtype=object' when creating the ndarray
  transforms = np.array(self._pipeline[from_ind: to_ind])[:, 1].copy()
2020-08-04 15:06:58,078 - mirisim.obssim.obssim - INFO - Creating pointing for␣
↪position 1
2020-08-04 15:06:58,079 - mirisim.obssim.obssim - INFO - Creating exposure event for␣
↪position 1
2020-08-04 15:06:58,080 - mirisim.obssim.event - DEBUG - Initializing ExposureEvent
2020-08-04 15:06:58,081 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.
↪02.00.fits'
2020-08-04 15:06:58,147 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.
↪fits'
2020-08-04 15:06:58,270 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.
↪02.00.fits'
2020-08-04 15:06:58,324 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.
↪fits'
2020-08-04 15:06:58,436 - mirisim.obssim.exposure - DEBUG - Initializing LrsExposure
2020-08-04 15:06:58,437 - mirisim.obssim.obssim - INFO - Observation simulation␣
↪started.
2020-08-04 15:06:58,438 - mirisim.obssim.event - INFO - Simulating ExposureEvent for␣
↪pointing 1
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:06:58,439 - mirisim.obssim.event - DEBUG - Start time of ExposureEvent:␣
↪2019-01-01 17:00:00
2020-08-04 15:06:58,440 - mirisim.obssim.exposure - INFO - Simulating Lrs exposures␣
↪for pointing 1
2020-08-04 15:06:58,440 - mirisim.obssim.exposure - INFO - Simulating detector␣
↪illumination for LRS exposures for pointing 1
2020-08-04 15:06:58,442 - mirisim.lrssim.lrs - INFO - lrs simulator processing ␣
↪cfgpath=LRS_SLIT
2020-08-04 15:06:58,443 - mirisim.lrssim.lrs - INFO - lrssim takes CDP PSF
2020-08-04 15:06:58,444 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.
↪02.00.fits'
2020-08-04 15:06:58,570 - mirisim.lrssim.lrs_extend - INFO - use PHOTOM (SRF) CDP to␣
↪convert flux
2020-08-04 15:06:58,573 - mirisim.cdp.get_cdp - INFO - Reading 'PHOTOM' model from '/
↪USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_PHOTOM_8B.03.00.fits'
2020-08-04 15:06:58,666 - mirisim.cdp.get_cdp - INFO - Reading 'AREA' model from '/
↪USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_AREA_07.00.00.fits'
2020-08-04 15:06:58,706 - mirisim.lrssim.lrs - INFO - v2_ref= -415.069047, v3_ref=-
↪400.575920 , v2_off=0.000000, v3_off=0.000000, pa=0.000000
2020-08-04 15:06:58,707 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.
↪fits'
2020-08-04 15:06:58,814 - mirisim.cdp.get_cdp - INFO - Reading 'DISTORTION' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.
↪02.00.fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪gwcs/wcs.py:126: VisibleDeprecationWarning: Creating an ndarray from ragged nested␣
↪sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different␣
↪lengths or shapes) is deprecated. If you meant to do this, you must specify
↪'dtype=object' when creating the ndarray
  transforms = np.array(self._pipeline[to_ind: from_ind])[:, 1].tolist()
2020-08-04 15:06:58,972 - mirisim.cdp.get_cdp - INFO - Reading 'PSF' model from '/
↪USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_PSF_07.04.00.fits'
2020-08-04 15:06:59,128 - mirisim.lrssim.LrsArrayMask - INFO - Retrieving file for␣
↪Lrs Array Mask: FocalPlaneMask
2020-08-04 15:06:59,163 - mirisim.lrssim.lrs - INFO - 2 point sources in scene
2020-08-04 15:06:59,164 - mirisim.lrssim.lrs - INFO - point source no: 0
2020-08-04 15:06:59,165 - mirisim.lrssim.lrs - INFO - lrs simulator processing in get_
↪point_int in mirisim.lrssim.lrs
2020-08-04 15:06:59,180 - mirisim.lrssim.lrs - INFO - xcenter -42.87364336671749
2020-08-04 15:06:59,180 - mirisim.lrssim.lrs - INFO - ycenter 330.67007377422743
2020-08-04 15:06:59,181 - mirisim.lrssim.lrs - INFO - xdiff -368.0036433667175
2020-08-04 15:06:59,181 - mirisim.lrssim.lrs - INFO - ydiff 30.970073774227444
2020-08-04 15:06:59,182 - mirisim.lrssim.lrs - WARNING - point source could be␣
↪outside the slit in spectral direction
2020-08-04 15:06:59,230 - mirisim.lrssim.lrs - INFO - point source no: 1
2020-08-04 15:06:59,230 - mirisim.lrssim.lrs - INFO - lrs simulator processing in get_
↪point_int in mirisim.lrssim.lrs
2020-08-04 15:06:59,245 - mirisim.lrssim.lrs - INFO - xcenter -362.28429457577647
2020-08-04 15:06:59,246 - mirisim.lrssim.lrs - INFO - ycenter 340.52043426203187
2020-08-04 15:06:59,246 - mirisim.lrssim.lrs - INFO - xdiff -687.4142945757765
2020-08-04 15:06:59,247 - mirisim.lrssim.lrs - INFO - ydiff 40.82043426203188
2020-08-04 15:06:59,247 - mirisim.lrssim.lrs - WARNING - point source could be␣
↪outside the slit in spectral direction
2020-08-04 15:06:59,287 - mirisim.lrssim.lrs - INFO - lrs simulator processing in get_
↪extended_from_scene_lrs in mirisim.lrssim.lrs
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:06:59,287 - mirisim.lrssim.lrs - INFO - 1 extended sources available in␣
↪scene
2020-08-04 15:06:59,288 - mirisim.lrssim.lrs - INFO - extended source no: 0
2020-08-04 15:06:59,710 - mirisim.lrssim.lrs - INFO - lrs simulator processing in␣
↪extract_sky_cube in mirisim.lrssim.lrs
2020-08-04 15:06:59,710 - mirisim.lrssim.lrs - INFO - no scenecube sources in scene
2020-08-04 15:06:59,711 - mirisim.lrssim.lrs - INFO - no fits skyCube sources are␣
↪computed
2020-08-04 15:06:59,711 - mirisim.lrssim.lrs - INFO - lrs simulator processing in get_
↪background_from_scene_lrs in mirisim.lrssim.lrs
2020-08-04 15:06:59,712 - mirisim.lrssim.lrs - INFO - 1 backgrounds available in scene
2020-08-04 15:07:00,455 - mirisim.obssim.exposure - INFO - Wrote illumination model:␣
↪20200804_150651_mirisim/illum_models/illum_model_seq1_MIRIMAGE_P750L.fits
2020-08-04 15:07:00,456 - mirisim.obssim.exposure - INFO - Simulating integrated␣
↪detector images for LRS exposures for pointing 1
2020-08-04 15:07:00,457 - mirisim.obssim.exposure - INFO - Simulating LRS exposure 1
2020-08-04 15:07:00,461 - mirisim.obssim.scasim - INFO - Running SCASim
2020-08-04 15:07:00,461 - miri.simulators.scasim - INFO - Simulating  detector␣
↪readout for MIRIMAGE from illumination data model of shape (1024, 1032).
2020-08-04 15:07:00,474 - miri.simulators.scasim - DEBUG - Metadata
2020-08-04 15:07:00,474 - miri.simulators.scasim - DEBUG - Data model location
↪            FITS key   Value                  Comment
2020-08-04 15:07:00,475 - miri.simulators.scasim - DEBUG - ~~~~~~~~~~~~~~~~~~~
↪            ~~~~~~~~   ~~~~~                  ~~~~~~~
2020-08-04 15:07:00,476 - miri.simulators.scasim - DEBUG - meta.coordinates.reference_
↪frame     | RADESYS  = ICRS                / Name of the coordinate reference frame
2020-08-04 15:07:00,476 - miri.simulators.scasim - DEBUG - meta.date
↪          | DATE     = 2020-08-04T15:07:00.356 / Date this file was created (UTC)
2020-08-04 15:07:00,477 - miri.simulators.scasim - DEBUG - meta.exposure.nframes
↪          | NFRAMES  = 1                   / Number of frames coadded in a group
2020-08-04 15:07:00,477 - miri.simulators.scasim - DEBUG - meta.filename
↪          | FILENAME = illum_model_seq1_MIRIMAGE_P750L.fits / Name of the file
2020-08-04 15:07:00,478 - miri.simulators.scasim - DEBUG - meta.filetype
↪          | FILETYPE = ILLUMINATION        / Type of data in the file
2020-08-04 15:07:00,478 - miri.simulators.scasim - DEBUG - meta.instrument.ccc_state
↪          | CCCSTATE = OPEN                / Contamination control cover state
2020-08-04 15:07:00,479 - miri.simulators.scasim - DEBUG - meta.instrument.detector
↪          | DETECTOR = MIRIMAGE            / Name of detector used to acquire the␣
↪data
2020-08-04 15:07:00,480 - miri.simulators.scasim - DEBUG - meta.instrument.detector_
↪settings    | DETSETNG = N/A                 / Detector settings used
2020-08-04 15:07:00,481 - miri.simulators.scasim - DEBUG - meta.instrument.filter
↪          | FILTER   = P750L               / Filter used by the instrument␣
↪(imaging)
2020-08-04 15:07:00,481 - miri.simulators.scasim - DEBUG - meta.instrument.model
↪          | MODELNAM = FM                  / Instrument model name
2020-08-04 15:07:00,482 - miri.simulators.scasim - DEBUG - meta.instrument.name
↪          | INSTRUME = MIRI                / Instrument used to acquire the data
2020-08-04 15:07:00,482 - miri.simulators.scasim - DEBUG - meta.model_type
↪          | DATAMODL =                     / Type of data model
2020-08-04 15:07:00,483 - miri.simulators.scasim - DEBUG - meta.observation.date
↪          | DATE-OBS = 2020-08-04T15:07:00.193 / [yyyy-mm-dd] UTC date at start of␣
↪exposure
2020-08-04 15:07:00,484 - miri.simulators.scasim - DEBUG - meta.pointing.dec_v1
↪          | DEC_V1   = 0.11127108899959638  / Dec of telescope V1 axis [deg]
2020-08-04 15:07:00,485 - miri.simulators.scasim - DEBUG - meta.pointing.pa_v3
↪          | PA_V3    = -0.0                / Position angle of telescope V3 axis␣
↪[deg]
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:07:00,486 - miri.simulators.scasim - DEBUG - meta.pointing.ra_v1
↪          | RA_V1    = 0.1152969573922563   / RA of telescope V1 axis [deg]
2020-08-04 15:07:00,486 - miri.simulators.scasim - DEBUG - meta.subarray.fastaxis
↪          | FASTAXIS = 1                    / Fast readout axis direction
2020-08-04 15:07:00,487 - miri.simulators.scasim - DEBUG - meta.subarray.name
↪          | SUBARRAY = FULL                 / Subarray used
2020-08-04 15:07:00,488 - miri.simulators.scasim - DEBUG - meta.subarray.slowaxis
↪          | SLOWAXIS = 2                    / Slow readout axis direction
2020-08-04 15:07:00,489 - miri.simulators.scasim - DEBUG - meta.subarray.xsize
↪          | SUBSIZE1 = 1032                 / Number of pixels in axis 1 direction
2020-08-04 15:07:00,490 - miri.simulators.scasim - DEBUG - meta.subarray.xstart
↪          | SUBSTRT1 = 1                    / Starting pixel in axis 1 direction
2020-08-04 15:07:00,490 - miri.simulators.scasim - DEBUG - meta.subarray.ysize
↪          | SUBSIZE2 = 1024                 / Number of pixels in axis 2 direction
2020-08-04 15:07:00,491 - miri.simulators.scasim - DEBUG - meta.subarray.ystart
↪          | SUBSTRT2 = 1                    / Starting pixel in axis 2 direction
2020-08-04 15:07:00,492 - miri.simulators.scasim - DEBUG - meta.telescope
↪          | TELESCOP = JWST                 / Telescope used to acquire the data
2020-08-04 15:07:00,492 - miri.simulators.scasim - DEBUG - meta.wcsinfo.cdelt1
↪          | CDELT1   = 1.0                  / Axis 1 increment per pixel
2020-08-04 15:07:00,493 - miri.simulators.scasim - DEBUG - meta.wcsinfo.cdelt2
↪          | CDELT2   = 1.0                  / Axis 2 increment per pixel
2020-08-04 15:07:00,493 - miri.simulators.scasim - DEBUG - meta.wcsinfo.cdelt3
↪          | CDELT3   = 1.0                  / Axis 3 increment per pixel
2020-08-04 15:07:00,494 - miri.simulators.scasim - DEBUG - meta.wcsinfo.crpix1
↪          | CRPIX1   = 0                    / Axis 1 coordinate of the reference␣
↪pixel
2020-08-04 15:07:00,495 - miri.simulators.scasim - DEBUG - meta.wcsinfo.crpix2
↪          | CRPIX2   = 0                    / Axis 2 coordinate of the reference␣
↪pixel
2020-08-04 15:07:00,496 - miri.simulators.scasim - DEBUG - meta.wcsinfo.crpix3
↪          | CRPIX3   = 0                    / Axis 3 coordinate of the reference␣
↪pixel
2020-08-04 15:07:00,496 - miri.simulators.scasim - DEBUG - meta.wcsinfo.crval1
↪          | CRVAL1   = 0.0                  / Axis 1 value at the reference pixel
2020-08-04 15:07:00,497 - miri.simulators.scasim - DEBUG - meta.wcsinfo.crval2
↪          | CRVAL2   = 0.0                  / Axis 2 value at the reference pixel
2020-08-04 15:07:00,498 - miri.simulators.scasim - DEBUG - meta.wcsinfo.crval3
↪          | CRVAL3   = 0.0                  / Axis 3 value at the reference pixel
2020-08-04 15:07:00,498 - miri.simulators.scasim - DEBUG - meta.wcsinfo.ctype1
↪          | CTYPE1   =                      / Axis 1 coordinate type
2020-08-04 15:07:00,499 - miri.simulators.scasim - DEBUG - meta.wcsinfo.ctype2
↪          | CTYPE2   =                      / Axis 2 coordinate type
2020-08-04 15:07:00,500 - miri.simulators.scasim - DEBUG - meta.wcsinfo.ctype3
↪          | CTYPE3   =                      / Axis 3 coordinate type
2020-08-04 15:07:00,500 - miri.simulators.scasim - DEBUG - meta.wcsinfo.dec_ref
↪          | DEC_REF  = 0.0                  / Declination of the reference point␣
↪(deg)
2020-08-04 15:07:00,501 - miri.simulators.scasim - DEBUG - meta.wcsinfo.ra_ref
↪          | RA_REF   = 0.0                  / Right ascension of the reference␣
↪point (deg)
2020-08-04 15:07:00,502 - miri.simulators.scasim - DEBUG - meta.wcsinfo.roll_ref
↪          | ROLL_REF = -0.0                 / Telescope roll angle of V3 measured␣
↪from North over East at the ref. point (deg)
2020-08-04 15:07:00,502 - miri.simulators.scasim - DEBUG - meta.wcsinfo.v2_ref
↪          | V2_REF   = -415.0690466121227   / Telescope v2 coordinate of the␣
↪reference point (arcsec)
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:07:00,503 - miri.simulators.scasim - DEBUG - meta.wcsinfo.v3_ref
↪          | V3_REF   = -400.575920398547    / Telescope v3 coordinate of the␣
↪reference point (arcsec)
2020-08-04 15:07:00,504 - miri.simulators.scasim - DEBUG - meta.wcsinfo.v3yangle
↪          | V3I_YANG = 0.0                   / Angle from V3 axis to Ideal y axis␣
↪(deg)
2020-08-04 15:07:00,504 - miri.simulators.scasim - DEBUG - meta.wcsinfo.vparity
↪          | VPARITY  = -1                    / Relative sense of rotation between␣
↪Ideal xy and V2V3
2020-08-04 15:07:00,505 - miri.simulators.scasim - DEBUG - meta.wcsinfo.wcsaxes
↪          | WCSAXES  = 3                     / number of World Coordinate System axes
2020-08-04 15:07:00,505 - miri.simulators.scasim - DEBUG -
2020-08-04 15:07:00,506 - miri.simulators.scasim - INFO - Results will be returned in␣
↪an exposure data model.
2020-08-04 15:07:00,507 - miri.simulators.scasim - DEBUG - NOTE: The following␣
↪effects are switched off: 'Quantum efficiency'
2020-08-04 15:07:00,507 - miri.simulators.scasim - DEBUG - CDP FTP host: www.miricle.
↪org
2020-08-04 15:07:00,508 - miri.simulators.scasim - DEBUG - CDP search path folder: /
↪CDPSIM/2.3/:/CDPSIM/
2020-08-04 15:07:00,522 - miri.simulators.scasim - INFO -   Detector readout mode is␣
↪FAST (samplesum=1, sampleskip=0, nframe=1, groupgap=0)
2020-08-04 15:07:00,522 - miri.simulators.scasim - INFO -   with 1 integrations and␣
↪ngroups=5 defined explicitly.
2020-08-04 15:07:00,523 - miri.simulators.scasim - INFO -   Detector subarray mode is␣
↪FULL.
2020-08-04 15:07:00,524 - miri.simulators.scasim - INFO - Detector temperature = 6.70␣
↪K (which affects dark current and read noise).
2020-08-04 15:07:00,524 - miri.simulators.scasim - INFO - Cosmic ray environment is␣
↪SOLAR_MIN.
2020-08-04 15:07:00,525 - miri.simulators - INFO - Reading cosmic ray library file: '/
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle.test/lib/python3.7/site-packages/
↪miri/simulators/data/cosmic_rays/CRs_SiAs_470_SUNMIN_04.fits'
2020-08-04 15:07:00,567 - miri.simulators.scasim - INFO - Simulation control flags:
2020-08-04 15:07:00,568 - miri.simulators.scasim - INFO -       Quantum efficiency␣
↪simulation turned OFF.
2020-08-04 15:07:00,568 - miri.simulators.scasim - INFO -       Poisson noise␣
↪simulation turned ON.
2020-08-04 15:07:00,569 - miri.simulators.scasim - INFO -       Read noise simulation␣
↪turned ON.
2020-08-04 15:07:00,569 - miri.simulators.scasim - INFO -       Reference pixels␣
↪simulation turned ON.
2020-08-04 15:07:00,570 - miri.simulators.scasim - INFO -       Bad pixels simulation␣
↪turned ON.
2020-08-04 15:07:00,570 - miri.simulators.scasim - INFO -       Dark current␣
↪simulation turned ON.
2020-08-04 15:07:00,571 - miri.simulators.scasim - INFO -       Flat-field simulation␣
↪turned ON.
2020-08-04 15:07:00,571 - miri.simulators.scasim - INFO -       Amplifier bias and␣
↪gain turned ON.
2020-08-04 15:07:00,572 - miri.simulators.scasim - INFO -       Detector non-
↪linearity effects turned ON.
2020-08-04 15:07:00,572 - miri.simulators.scasim - INFO -       Detector drift␣
↪effects turned ON.
2020-08-04 15:07:00,573 - miri.simulators.scasim - INFO -       Detector latency␣
↪effects turned ON.
2020-08-04 15:07:00,573 - miri.simulators.scasim - WARNING - ***Illumination map of␣
↪size 1024 x 1032 is too large! Truncating to detector size of 1024 x 1024 pixels. ***
```

(continues on next page)

(continued from previous page)

```
2020-08-04 15:07:00,577 - miri.simulators.scasim - INFO - Input subarray mode␣
↪obtained from illumination map: FULL
2020-08-04 15:07:00,578 - miri.simulators.scasim - INFO - Detector properties␣
↪translates input subarray FULL into None
2020-08-04 15:07:00,578 - miri.simulators.scasim - INFO - Creating a new detector␣
↪object for 1024 rows x 1024 columns.
2020-08-04 15:07:00,581 - miri.simulators.get_cdp - INFO - Reading 'MASK' model from
↪'/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_MASK_07.02.01.fits'
2020-08-04 15:07:00,670 - miri.simulators.get_cdp - INFO - Reading 'GAIN' model from
↪'/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits'
2020-08-04 15:07:03,321 - miri.simulators.scasim.detector - INFO - Reading DARK model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_DARK_06.01.00.
↪fits'
2020-08-04 15:07:07,395 - miri.simulators.scasim.detector - DEBUG - Find a PIXELFLAT␣
↪for detector=MIRIMAGE, readpatt=FAST, subarray=FULL filter=P750L, band=None
2020-08-04 15:07:07,401 - miri.simulators.get_cdp - INFO - Reading 'PIXELFLAT' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_PIXELFLAT_07.
↪01.01.fits'
2020-08-04 15:07:07,494 - miri.simulators.scasim.detector - WARNING - Could not find␣
↪exact match for pixel flat-field for detector MIRIMAGE with FAST mode with filter=
↪'P750L'. An alternative is being used.
2020-08-04 15:07:07,562 - miri.simulators.scasim.detector - DEBUG - Find a LINEARITY␣
↪CDP for detector=MIRIMAGE filter=P750L, band=None
2020-08-04 15:07:07,565 - miri.simulators.get_cdp - INFO - Reading 'LINEARITY' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_LINEARITY_06.02.00.
↪fits'
2020-08-04 15:07:07,841 - miri.simulators.get_cdp - INFO - Reading 'READNOISE' model␣
↪from '/USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_READNOISE_07.
↪01.00.fits'
2020-08-04 15:07:07,906 - miri.simulators.scasim - INFO - Creating exposure_data with␣
↪1280 rows x 1032 columns plus 5 groups and 1 ints.
2020-08-04 15:07:08,179 - miri.simulators.scasim - INFO - Simulating 1 integration.
2020-08-04 15:07:08,190 - miri.simulators.scasim - WARNING - ***Input flux array␣
↪contains 12 negative values.
2020-08-04 15:07:08,329 - miri.simulators.scasim - INFO - Simulating 5 groups for␣
↪integration 1.
2020-08-04 15:07:09,342 - miri.simulators.scasim - INFO - Adding the DARK calibration␣
↪from /USERS/pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_DARK_06.01.00.
↪fits
2020-08-04 15:07:09,415 - miri.simulators.scasim - INFO - Correcting nonlinearity␣
↪from MIRI_FM_MIRIMAGE_LINEARITY_06.02.00.fits
2020-08-04 15:07:09,588 - miri.simulators.scasim - INFO - Output subarray undefined␣
↪or FULL. SUBSTRT=(1,1), SUBSIZE=(1032,1024)
2020-08-04 15:07:09,589 - miri.simulators.scasim - INFO - WCS keywords defined as␣
↪CRPIX1=0, CRPIX2=0
2020-08-04 15:07:14,684 - miri.simulators.scasim - INFO - Exposure time 13.88s␣
↪(duration 14.93s)
2020-08-04 15:07:15,135 - mirisim.obssim.exposure - INFO - Wrote detector image:␣
↪20200804_150651_mirisim/det_images/det_image_seq1_MIRIMAGE_P750Lexp1.fits
2020-08-04 15:07:15,135 - mirisim.obssim.event - DEBUG - Elapsed time (total, ima,␣
↪mrs): 0:00:14.931968, 0:00:14.931968, 0:00:00
2020-08-04 15:07:15,136 - INFO - MIRI Simulation finished. Results have been saved to:
↪ 20200804_150651_mirisim
```

15.4.2 Examine some of the results

Now that the MIRISim simulation has completed, lets examine the results.

The first thing to note is that the outputs are placed in a date-labelled directory taking the form YYYYMMDD_hhmmss_mirisim. The name of the output directory is given in the last line of the MIRISim log above. Because the output directory is date-labelled, we can quantify which was the most recent run of MIRISim, and find its output directory using glob.glob

[10]:
```
outputdir = sorted(glob.glob('*_*_mirisim'),key=os.path.getmtime)[-1]
#[-1] takes the last entry found

outputDirContents = os.listdir(outputdir)

directories = [name for name in outputDirContents if
               os.path.isdir(os.path.join(outputdir,name))]
files = [name for name in outputDirContents if not
               os.path.isdir(os.path.join(outputdir,name))]

print('The subdirectories in the outputdirectory are:\n{}'.format(directories))
print('The files in the outputdirectory are:\n{}'.format(files))
```

```
The subdirectories in the outputdirectory are:
['det_images', 'illum_models']
The files in the outputdirectory are:
['simulator.ini', 'ima_recommended_dither.dat', 'scene.ini', 'simulation.ini',
↪'mirisim.log']
```

The files contain the log which was also output to the terminal (mirisim.log) and copies of the .ini files used (or created from python inputs) to create the simulation. These versions of the .ini files can be used to re-create the run of the simulation

The directories contain various outputs of MIRISim:

- ** illum_models ** houses FITS images of the illuminations sent to the detector (sent to SCASim - the simulator of the Sensor Chip Assembly). This should have the same format as the detector image, but without all of the detector effects and noise. There are FITS files produced for each exposure and dither position
- ** det_images ** houses FITS images of the final outputs of MIRISim. These detector images have all of the detector effects and noise incorporated. The number of detector images should be the same as the number of illumination models.

The headers of the detector images are formatted for ingest into the JWST pipeline.

Below is a small code snippet used to draw the images. it needs to be run, but doesn't produce any output directly (it's called later to show the output images)

[11]:
```
def show_outputs(MIRISim_outputdir,output_type):
    '''
    plot the specified channel of the MIRISim outputs
    :param MIRISim_outputdir:
        name of the date-labelled dir. holding the MIRISIM outputs
    :param output_type:
        type of output to process
        (e.g. illum_models, det_images or skycubes)
    '''

    infits = glob.glob('{}/{}/*.fits'.format(MIRISim_outputdir,output_type))[0]
```

(continues on next page)

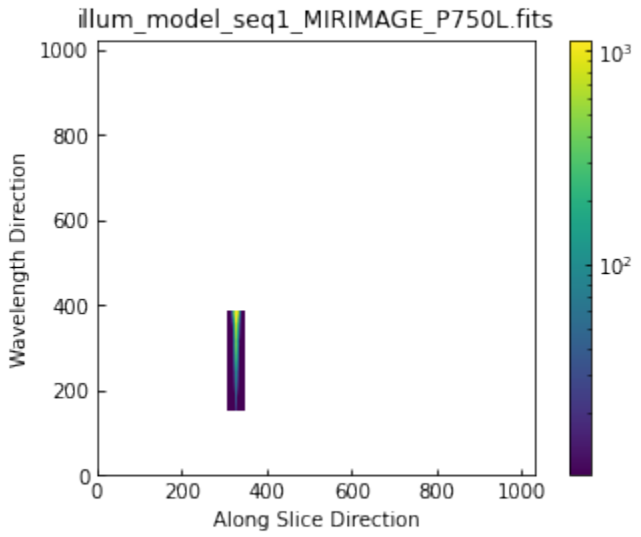
(continued from previous page)

```
    hdulist = fits.open(infits)


    hdu_index = 1
    if len(hdulist[hdu_index].data.shape) > 3:
        integ,frames,nx,ny = hdulist[hdu_index].data.shape
        image = hdulist[hdu_index].data[integ-1,frames-1,:,:]


    else:
        image = hdulist[hdu_index].data[:,:]

    norm = colors.LogNorm(image.mean() + 0.5 * image.std(), 0.5*image.max(),␣
↪clip=True)
    plt.imshow(image,origin = 'lower', cmap = cm.viridis,interpolation = 'nearest',
↪norm = norm)
    plt.title('{}'.format(infits.split('/')[-1]))
    plt.xlabel('Along Slice Direction')
    plt.ylabel('Wavelength Direction')


    plt.colorbar()
```

15.5 Viewing an illumination model

Below shows an example of an output illumination model

[12]:
```
show_outputs(outputdir,'illum_models')
```



15.5.1 Viewing the final detector image

The detector images are the final output of MIRISim, and have data structures and formatting consistent with what will come from MIRI itself. The data format is JWST pipeline ready. Below an example image of the last frame of the last integration is shown. The units are DN.

Additionally, below the FITS header information for the SCI extension is listed.

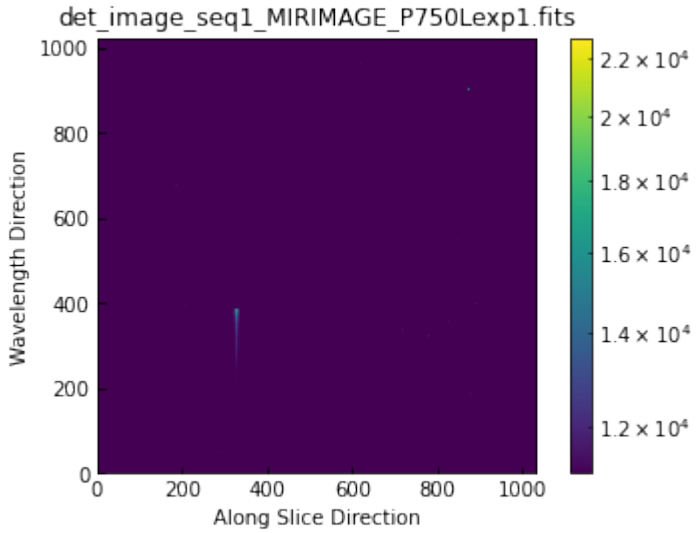
[13]:
```
show_outputs(outputdir,'det_images')

infits = glob.glob('{}/det_images/*.fits'.format(outputdir))[0]
hdulist = fits.open(infits)
hdulist[1].header
```

[13]:
```
XTENSION= 'IMAGE   '           / Image extension
BITPIX  =                  -32 / array data type
NAXIS   =                    4 / number of array dimensions
NAXIS1  =                 1032
NAXIS2  =                 1024
NAXIS3  =                    5
NAXIS4  =                    1
PCOUNT  =                    0 / number of parameters
GCOUNT  =                    1 / number of groups
EXTNAME = 'SCI     '           / extension name

        Information about the coordinates in the file

RADESYS = 'ICRS    '           / Name of the coordinate reference frame

        Information about the data array

BUNIT   = 'DN      '           / Units of the data array

        Spacecraft pointing information

RA_V1   =   0.1152969573922563 / [deg] RA of telescope V1 axis
DEC_V1  =   0.1112710889995964 / [deg] Dec of telescope V1 axis
PA_V3   =                -0.0 / [deg] Position angle of telescope V3 axis

        WCS parameters

WCSAXES =                    3 / number of World Coordinate System axes
CRPIX1  =                    0 / axis 1 coordinate of the reference pixel
CRPIX2  =                    0 / axis 2 coordinate of the reference pixel
CRPIX3  =                    0 / axis 3 coordinate of the reference pixel
CRVAL1  =                  0.0 / first axis value at the reference pixel
CRVAL2  =                  0.0 / second axis value at the reference pixel
CRVAL3  =                  0.0 / third axis value at the reference pixel
CTYPE1  = '' / first axis coordinate type
CTYPE2  = '' / second axis coordinate type
CTYPE3  = '' / third axis coordinate type
CUNIT1  = '' / first axis units
CUNIT2  = '' / second axis units
CUNIT3  = '' / third axis units
CDELT1  =                  1.0 / first axis increment per pixel
CDELT2  =                  1.0 / second axis increment per pixel
CDELT3  =                  1.0 / third axis increment per pixel
V2_REF  =   -415.0690466121227 / [arcsec] Telescope v2 coordinate of the referen
```

(continues on next page)

(continued from previous page)

```
V3_REF  =    -400.575920398547 / [arcsec] Telescope v3 coordinate of the referen
VPARITY =                   -1 / Relative sense of rotation between Ideal xy and
V3I_YANG=                  0.0 / [deg] Angle from V3 axis to Ideal y axis
RA_REF  =                  0.0 / [deg] Right ascension of the reference point
DEC_REF =                  0.0 / [deg] Declination of the reference point
ROLL_REF=                 -0.0 / [deg] V3 roll angle at the ref point (N over E)
EXTVER  =                    1 / extension value
```



[ ]:

CHAPTER

SIXTEEN

APPENDIX: WALK THROUGH OF MIRISIM (LRS-SLITLESS)

In this notebook, we create a scene with a circle of point sources around a central galaxy (that all fit within the Imager field of view), and run through a MIRISim simulation showing the products created at each stage.

NOTE this notebook must be started within the mirisim anaconda environment in which you have MIRISim installed. This is a pre-requist for using MIRISim, and installation instructions can be found here.

When launching the notebook, make sure you've run the appropriate version of

conda activate mirisim

or

conda activate miricle

in the terminal before starting the notebook.

16.1 Steps in this notebook:

1. Create a Scene
2. Initialise the simulation parameters
3. run the simulation
4. examine some of the outputs.

[1]:
```
# import the configuration file parsers so they can be written to file
from mirisim.config_parser import SimConfig, SimulatorConfig, SceneConfig

# import scene component generators
from mirisim.skysim import Background, sed, Point, Galaxy, kinetics
from mirisim.skysim import wrap_pysynphot as wS

from mirisim import MiriSimulation

#other things to be used
import numpy as np
import glob                 # glob is used to find the output directory
import os                   # for listing directory contents
from astropy.io import fits # for reading FITS file contents

import matplotlib.pyplot as plt    # to display images
from matplotlib import colors,cm
%matplotlib inline
```

16.1.1 Create a Scene

The scene created below consists of a circle of point sources surrounding a Galaxy,and a low level background, all of which fit within the main imager field of view. The point sources all have the same (blackbody) SEDs to make things simple. The galaxy as a pysynphot SED which is modified by a Keplerian velocity field.

to be able to simply add the point sources ('stars') to a scene, it first needs to be initialised with a background

Create the background emission

[2]:
```
bg = Background(level = 'low', gradient = 5., pa = 45.)
```

```
2020-09-11 14:32:55,545 - INFO - Initializing Background
```

Create a circle of point sources

These points will be put into a 'compound target list' for simplicity later. This list needs to be initialised, so the first thing will be to create a single point source, which is placed to fall into the LRS slit.

[3]:
```
# create point source objects across the Imager Field of View
def create_star(xpos,ypos):
    star = Point(Cen = (xpos,ypos))
    BBparams = {'Temp': 1e4,
                'wref': 10.,
                'flux': 7e5}
    Blackbody = sed.BBSed(**BBparams)
    star.set_SED(Blackbody)

    return star


# create a single point source at the position of the LRS slit,
# and another at the LRS slitless position. The positions of the slit
# and slitless targets (in pixels) come from figure 2 of
# Kendrew et al. (2015)  (MIRI PASP series #4)

full_array = (1024,1032)
#center_pixel = (692,512)    # center of the array (in pixel coordinates)
center_pixel = (512,512)
slit_pixel = (321,512)      # centering pixel for the slit
slitless_target = (512,512)
pixel_scale = 0.11          #arcsec/pixel


slit_star = create_star(
    (center_pixel[0]-slit_pixel[0])*pixel_scale,
    (center_pixel[1]-slit_pixel[1])*pixel_scale)

slitless_star = create_star(
    (center_pixel[0]-(full_array[0]-slitless_target[0]))*pixel_scale,
    (center_pixel[1]-(full_array[1]-slitless_target[1]))*pixel_scale)

# add these two stars to the compound target list called 'stars'
stars = slitless_star + slit_star
```

(continues on next page)

(continued from previous page)

```
# for writing to scene.ini file
targetlist = [slit_star,slitless_star]
```

```
2020-09-11 14:32:55,555 - INFO - Initializing Point
2020-09-11 14:32:55,556 - INFO - Initializing Point
```

Create the Galaxy

[4]:
```
# initialise the galaxy with a position
galaxy = Galaxy(Cen = (0.,0.),n=1.,re=1.,q=0.4,pa=0)

# set the properties of the SED
PYSPsedDict = {'family': 'bkmodels', 'sedname': 'bk_b0005', 'flux': 1E+5, 'wref': 10.}
# read that dictionary into the pysynphot interpreter
sedE = wS.PYSPSed(**PYSPsedDict)
# add the SED to the galaxy
galaxy.set_SED(sedE)

# create a velocity mapping for the SED
VMAPpars = {'vrot': 200., 'Cen': (0., 0.), 'pa': 0., 'q': 0.4, 'c': 0}
VelocityMap = kinetics.FlatDisk(**VMAPpars)
# add the velocity map to the galaxy
galaxy.set_velomap(VelocityMap)

# add a line of sight velocity distribution to the galaxy
losVeloDist = kinetics.Losvd(sigma=200.,h3=0.,h4=0.)
galaxy.set_LOSVD(losVeloDist)
```

```
2020-09-11 14:32:55,564 - INFO - Initializing Galaxy
2020-09-11 14:32:55,565 - INFO - Initializing Galaxy
2020-09-11 14:32:55,565 - INFO - Initializing Galaxy
2020-09-11 14:32:55,630 - INFO - Initializing FlatDisk
2020-09-11 14:32:55,630 - INFO - Initializing Losvd
```

16.2 From the components, create a scene

create a scene is as simple as adding together the targets.

[5]:
```
scene = bg + stars + galaxy

# for writing to scene.ini file
#targetlist.append(galaxy)
```

16.3 Export the scene to an ini file and FITS

The scene can also be exported to an ini file (for future use), or a FITS file to be visualised.

Exporting to a FITS file requires specifing a number of aditional parametrs such as the Field of view, required spectral sampling, etc. They're described in more detail below. For large fields of view, and/or small spectral channels, which create large FITS files, writing to a FITS file can take a while (few minutes)

NOTE: export to fits currently (8/Feb/2017) doesn't work properly

[6]:
```
## export to ini file

scene_config = SceneConfig.makeScene(loglevel=0,
                                     background=bg,
                                     targets = targetlist)

os.system('rm LRS_example_scene.ini')
scene_config.write('LRS_example_scene.ini')

## export to FITS file
FOV = np.array([[-57.,57.],[-57.,57.]])   # field of view [xmin,xmax],[ymin,ymax] (in␣
↪arcsec)
SpatialSampling = 0.1                # spatial sampling (in arcsec)
WavelengthRange = [5,10]             # wavelength range to process (in microns)
WavelengthSampling = 0.2            # channel width (in microns)


# overwrite = True enables overwriting of any previous version of the fits file
# with the same name as that given in the writecube command
scene.writecube(cubefits = 'LRS_example_scene.fits',
                FOV = FOV, time = 0.0,
                spatsampling = SpatialSampling,
                wrange = WavelengthRange,
                wsampling = WavelengthSampling,
                overwrite = True)
```

16.3.1 Initialise the Simulation Parameters

This is where the parameters for the MRS simulation get set. Note that for internal consistency in MIRISim, all settings (including those not used in the MRS simulation here) must be set. Those not being used in this simulation are labelled with NOT USED HERE in the comments of each line

[7]:
```
sim_config = SimConfig.makeSim(
    name = 'ima_simulation',     # name given to simulation
    scene = 'LRS_example_scene.ini', # name of scene file to input
    rel_obsdate = 0.0,           # relative observation date (0 = launch, 1 = end of 5␣
↪yrs)
    POP = 'IMA',                 # Component on which to center (Imager or MRS)
    ConfigPath = 'LRS_SLITLESS',  # Configure the Optical path (MRS sub-band)
    Dither = False,              # Don't Dither
    StartInd = 1,                # start index for dither pattern [NOT USED HERE]
    NDither = 2,                 # number of dither positions [NOT USED HERE]
    DitherPat = 'ima_recommended_dither.dat', # dither pattern to use [NOT USED HERE]
```

(continues on next page)

(continued from previous page)

```
    disperser = 'SHORT',          # [NOT USED HERE]
    detector = 'SW',              # [NOT USED HERE]
    mrs_mode = 'SLOW',            # [NOT USED HERE]
    mrs_exposures = 1,            # [NOT USED HERE]
    mrs_integrations = 1,         # [NOT USED HERE]
    mrs_frames = 1,               # [NOT USED HERE]
    ima_exposures = 1,            # number of exposures
    ima_integrations = 1,         # number of integrations
    ima_frames = 20,               # number of groups (for MIRI, # Groups = # Frames)
    ima_mode = 'FAST',            # Imager read mode (default is FAST ~ 2.3 s)
    filter = 'P750L',           # Imager Filter to use
    readDetect = 'SLITLESSPRISM'          # Portion of detector to read out
)
```

16.4 Export the simulation setup to a file

[8]:
```
os.system('rm LRS_simulation.ini')
sim_config.write('LRS_simulation.ini')
```

16.4.1 Run the simulation

Now that the scene and the setup of the simulation have been set, we can run the simulation.

the last step is to setup the defaults for internal things like CDPs.

[9]:
```
simulator_config = SimulatorConfig.from_default()


mysim = MiriSimulation(sim_config,scene_config,simulator_config)
mysim.run()
```

```
2020-09-11 14:34:29,039 - INFO - MIRISim version: 2.3.0
2020-09-11 14:34:29,040 - INFO - MIRI Simulation started.
2020-09-11 14:34:29,041 - INFO - Output will be saved to: 20200911_143429_mirisim
2020-09-11 14:34:29,041 - INFO - Storing configs in output directory.
2020-09-11 14:34:29,680 - INFO - Reading cosmic ray properties from parameter file /
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle/lib/python3.7/site-packages/miri/
↪simulators/scasim/cosmic_ray_properties.py
2020-09-11 14:34:29,687 - INFO - Reading detector properties from parameter file /
↪Users/pamelaklaassen/opt/anaconda3/envs/miricle/lib/python3.7/site-packages/miri/
↪simulators/scasim/detector_properties.py
2020-09-11 14:34:29,725 - INFO - Storing dither pattern in output directory.
2020-09-11 14:34:29,727 - INFO - Using $CDP_DIR for location of CDP files: /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP
2020-09-11 14:34:29,727 - INFO - Setting up simulated Observation, with following␣
↪settings:
2020-09-11 14:34:29,728 - INFO - Configuration Path: LRS_SLITLESS
2020-09-11 14:34:29,729 - INFO - Primary optical path: IMA
2020-09-11 14:34:29,729 - INFO - IMA Filter: P750L
2020-09-11 14:34:29,730 - INFO - IMA Subarray: SLITLESSPRISM
2020-09-11 14:34:29,731 - INFO - IMA detector readout mode: FAST
2020-09-11 14:34:29,731 - INFO - IMA detector # exposures: 1
2020-09-11 14:34:29,732 - INFO - IMA detector # integrations: 1
```

(continues on next page)

(continued from previous page)

```
2020-09-11 14:34:29,732 - INFO - IMA detector # frames: 20
2020-09-11 14:34:29,733 - INFO - Parsing: Background
2020-09-11 14:34:29,734 - INFO - Initializing Background
2020-09-11 14:34:29,734 - INFO - Parsing: point_1
2020-09-11 14:34:29,735 - INFO - Initializing Point
2020-09-11 14:34:29,735 - INFO - Parsing: point_2
2020-09-11 14:34:29,736 - INFO - Initializing Point
2020-09-11 14:34:29,737 - INFO - Simulating a single pointing.
2020-09-11 14:34:33,354 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.02.00.fits'
2020-09-11 14:34:34,144 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-09-11 14:34:34,299 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-09-11 14:34:34,408 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.02.00.fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle/lib/python3.7/site-packages/gwcs/wcs.
↪py:131: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences␣
↪(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or␣
↪shapes) is deprecated. If you meant to do this, you must specify 'dtype=object'␣
↪when creating the ndarray
  transforms = np.array(self._pipeline[from_ind: to_ind])[:, 1].copy()
2020-09-11 14:34:34,475 - INFO - Creating pointing for position 1
2020-09-11 14:34:34,476 - INFO - Creating exposure event for position 1
2020-09-11 14:34:34,477 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.02.00.fits'
2020-09-11 14:34:34,534 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-09-11 14:34:34,642 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.02.00.fits'
2020-09-11 14:34:34,694 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-09-11 14:34:34,807 - INFO - Observation simulation started.
2020-09-11 14:34:34,807 - INFO - Simulating ExposureEvent for pointing 1
2020-09-11 14:34:34,808 - INFO - Simulating Lrs exposures for pointing 1
2020-09-11 14:34:34,809 - INFO - Simulating detector illumination for LRS exposures␣
↪for pointing 1
2020-09-11 14:34:34,810 - INFO - lrs simulator processing  cfgpath=LRS_SLITLESS
2020-09-11 14:34:34,811 - INFO - lrssim takes CDP PSF
2020-09-11 14:34:34,813 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.02.00.fits'
2020-09-11 14:34:34,874 - INFO - use PHOTOM (SRF) CDP to convert flux
2020-09-11 14:34:34,876 - INFO - Reading 'PHOTOM' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_SLITLESSPRISM_PHOTOM_8B.03.00.fits'
2020-09-11 14:34:34,999 - INFO - Reading 'AREA' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_AREA_07.00.00.fits'
2020-09-11 14:34:35,054 - INFO - v2_ref= -378.832074, v3_ref=-344.944543 , v2_off=0.
↪000000, v3_off=0.000000, pa=0.000000
2020-09-11 14:34:35,055 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_DISTORTION_07.04.01.fits'
2020-09-11 14:34:35,174 - INFO - Reading 'DISTORTION' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_DISTORTION_07.02.00.fits'
/Users/pamelaklaassen/opt/anaconda3/envs/miricle/lib/python3.7/site-packages/gwcs/wcs.
↪py:126: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences␣
↪(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or␣
↪shapes) is deprecated. If you meant to do this, you must specify 'dtype=object'␣
↪when creating the ndarray
```

(continues on next page)

(continued from previous page)

```
  transforms = np.array(self._pipeline[to_ind: from_ind])[:, 1].tolist()
2020-09-11 14:34:35,350 - INFO - Reading 'PSF' model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_SLITLESSPRISM_PSF_07.04.00.fits'
2020-09-11 14:34:35,483 - INFO - Retrieving file for Lrs Array Mask: FocalPlaneMask
2020-09-11 14:34:35,508 - INFO - 2 point sources in scene
2020-09-11 14:34:35,509 - INFO - point source no: 0
2020-09-11 14:34:35,510 - INFO - lrs simulator processing in get_point_int in mirisim.
↪lrssim.lrs
2020-09-11 14:34:35,522 - INFO - xcenter -152.97374031791537
2020-09-11 14:34:35,523 - INFO - ycenter 845.5328569337906
2020-09-11 14:34:35,524 - INFO - xdiff -190.47374031791537
2020-09-11 14:34:35,525 - INFO - ydiff 17.53285693379064
2020-09-11 14:34:35,576 - INFO - point source no: 1
2020-09-11 14:34:35,577 - INFO - lrs simulator processing in get_point_int in mirisim.
↪lrssim.lrs
2020-09-11 14:34:35,591 - INFO - xcenter 36.87939441505
2020-09-11 14:34:35,592 - INFO - ycenter 820.0677420430213
2020-09-11 14:34:35,593 - INFO - xdiff -0.620605584949999
2020-09-11 14:34:35,594 - INFO - ydiff -7.932257956978674
2020-09-11 14:34:35,644 - INFO - no extended sources are computed
2020-09-11 14:34:35,645 - INFO - lrs simulator processing in extract_sky_cube in␣
↪mirisim.lrssim.lrs
2020-09-11 14:34:35,645 - INFO - no scenecube sources in scene
2020-09-11 14:34:35,646 - INFO - no fits skyCube sources are computed
2020-09-11 14:34:35,647 - INFO - calculate background for slitless mode
2020-09-11 14:34:35,664 - INFO - lrs simulator processing in get_background_from_
↪scene_lrs_extend in mirisim.lrssim.lrs_extend
2020-09-11 14:34:35,665 - INFO - 1 backgrounds available in scene
2020-09-11 14:37:27,941 - INFO - Wrote illumination model: 20200911_143429_mirisim/
↪illum_models/illum_model_seq1_MIRIMAGE_P750L.fits
2020-09-11 14:37:27,942 - INFO - Simulating integrated detector images for LRS␣
↪exposures for pointing 1
2020-09-11 14:37:27,943 - INFO - Simulating LRS exposure 1
2020-09-11 14:37:27,947 - INFO - Running SCASim
2020-09-11 14:37:27,948 - INFO - Simulating  detector readout for MIRIMAGE from␣
↪illumination data model of shape (416, 72).
2020-09-11 14:37:27,959 - INFO - Results will be returned in an exposure data model.
2020-09-11 14:37:27,972 - INFO -   Detector readout mode is FAST (samplesum=1,␣
↪sampleskip=0, nframe=1, groupgap=0)
2020-09-11 14:37:27,973 - INFO -   with 1 integrations and ngroups=20 defined␣
↪explicitly.
2020-09-11 14:37:27,973 - INFO -   Detector subarray mode is SLITLESSPRISM  [529, 1,␣
↪416, 72].
2020-09-11 14:37:27,974 - INFO - Detector temperature = 6.70 K (which affects dark␣
↪current and read noise).
2020-09-11 14:37:27,974 - INFO - Cosmic ray environment is SOLAR_MIN.
2020-09-11 14:37:27,975 - INFO - Reading cosmic ray library file: '/Users/
↪pamelaklaassen/opt/anaconda3/envs/miricle/lib/python3.7/site-packages/miri/
↪simulators/data/cosmic_rays/CRs_SiAs_470_SUNMIN_01.fits'
2020-09-11 14:37:28,073 - INFO - Simulation control flags:
2020-09-11 14:37:28,074 - INFO -        Quantum efficiency simulation turned OFF.
2020-09-11 14:37:28,074 - INFO -        Poisson noise simulation turned ON.
2020-09-11 14:37:28,075 - INFO -        Read noise simulation turned ON.
2020-09-11 14:37:28,075 - INFO -        Reference pixels simulation turned ON.
2020-09-11 14:37:28,076 - INFO -        Bad pixels simulation turned ON.
2020-09-11 14:37:28,077 - INFO -        Dark current simulation turned ON.
2020-09-11 14:37:28,078 - INFO -        Flat-field simulation turned ON.
```

(continues on next page)

(continued from previous page)

```
2020-09-11 14:37:28,078 - INFO -        Amplifier bias and gain turned ON.
2020-09-11 14:37:28,079 - INFO -        Detector non-linearity effects turned ON.
2020-09-11 14:37:28,080 - INFO -        Detector drift effects turned ON.
2020-09-11 14:37:28,080 - INFO -        Detector latency effects turned ON.
2020-09-11 14:37:28,081 - INFO - Input subarray mode obtained from illumination map:␣
↪SLITLESSPRISM
2020-09-11 14:37:28,082 - INFO - Detector properties translates input subarray␣
↪SLITLESSPRISM into [529, 1, 416, 72]
2020-09-11 14:37:28,083 - INFO - Creating a new detector object for 1024 rows x 1024␣
↪columns.
2020-09-11 14:37:28,086 - INFO - Reading 'MASK' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_MASK_07.02.01.fits'
2020-09-11 14:37:28,175 - INFO - Reading 'GAIN' model from '/USERS/pamelaklaassen/
↪WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_GAIN_04.00.00.fits'
2020-09-11 14:37:31,390 - INFO - Reading DARK model from '/USERS/pamelaklaassen/WORK/
↪MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_SLITLESSPRISM_DARK_06.01.00.fits'
2020-09-11 14:37:31,674 - INFO - Reading 'PIXELFLAT' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_P750L_SLITLESSPRISM_PIXELFLAT_07.
↪01.01.fits'
2020-09-11 14:37:31,738 - WARNING - Could not find exact match for pixel flat-field␣
↪for detector MIRIMAGE with FAST mode with filter='P750L'. An alternative is being␣
↪used.
2020-09-11 14:37:31,748 - INFO - Reading 'LINEARITY' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_LINEARITY_06.02.00.fits'
2020-09-11 14:37:31,957 - INFO - Reading 'READNOISE' model from '/USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_READNOISE_07.01.00.fits'
2020-09-11 14:37:31,999 - INFO - Creating exposure_data with 520 rows x 72 columns␣
↪plus 20 groups and 1 ints.
2020-09-11 14:37:32,178 - INFO - Simulating 1 integration.
2020-09-11 14:37:32,286 - INFO - Simulating 20 groups for integration 1.
2020-09-11 14:37:35,419 - INFO - Adding the DARK calibration from /USERS/
↪pamelaklaassen/WORK/MIRI/DHAS/CDP/MIRI_FM_MIRIMAGE_FAST_SLITLESSPRISM_DARK_06.01.00.
↪fits
2020-09-11 14:37:35,425 - INFO - Correcting nonlinearity from MIRI_FM_MIRIMAGE_
↪LINEARITY_06.02.00.fits
2020-09-11 14:37:35,463 - INFO - Output subarray defined. SUBSTRT=(1,529),␣
↪SUBSIZE=(72,416)
2020-09-11 14:37:35,464 - INFO - WCS keywords defined as CRPIX1=0, CRPIX2=-528
2020-09-11 14:37:35,879 - INFO - Exposure time 3.18s (duration 3.21s)
2020-09-11 14:37:36,022 - INFO - Wrote detector image: 20200911_143429_mirisim/det_
↪images/det_image_seq1_MIRIMAGE_P750Lexp1.fits
2020-09-11 14:37:36,023 - INFO - MIRI Simulation finished. Results have been saved to:
↪ 20200911_143429_mirisim
```

16.4.2 Examine some of the results

Now that the MIRISim simulation has completed, lets examine the results.

The first thing to note is that the outputs are placed in a date-labelled directory taking the form YYYYMMDD_hhmmss_mirisim. The name of the output directory is given in the last line of the MIRISim log above. Because the output directory is date-labelled, we can quantify which was the most recent run of MIRISim, and find its output directory using glob.glob

[10]:
```
outputdir = sorted(glob.glob('*_*_mirisim'),key=os.path.getmtime)[-1]    #[-1] takes␣
↪the last entry found

outputDirContents = os.listdir(outputdir)

directories = [name for name in outputDirContents if os.path.isdir(os.path.
↪join(outputdir,name))]
files = [name for name in outputDirContents if not os.path.isdir(os.path.
↪join(outputdir,name))]

print('The subdirectories in the outputdirectory are:\n{}'.format(directories))
print('The files in the outputdirectory are:\n{}'.format(files))
```

```
The subdirectories in the outputdirectory are:
['det_images', 'illum_models']
The files in the outputdirectory are:
['simulator.ini', 'ima_recommended_dither.dat', 'scene.ini', 'simulation.ini',
↪'mirisim.log']
```

The files contain the log which was also output to the terminal (mirisim.log) and copies of the .ini files used (or created from python inputs) to create the simulation. These versions of the .ini files can be used to re-create the run of the simulation

The directories contain various outputs of MIRISim:

- ** illum_models ** houses FITS images of the illuminations sent to the detector (sent to SCAsim - the simulator of the Sensor Chip Assembly). This should have the same format as the detector image, but without all of the detector effects and noise. There are FITS files produced for each exposure and dither position
- ** det_images ** houses FITS images of the final outputs of MIRISim. These detector images have all of the detector effects and noise incorporated. The number of detector images should be the same as the number of illumination models.

The headers of the detector images are formatted for ingest into the JWST pipeline.

For those on the MIRI Test Team, the following link takes you to Patrick Kavanagh's ipython notebook walkthrough of the current build of the JWST pipeline

(http://miri.ster.kuleuven.be/bin/view/Internal/TutorialsPipeline)

Which is also linked from the MIRISim twiki page.

Below is a small code snippet used to draw the images. it needs to be run, but doesn't produce any output directly (it's called later to show the output images)

[11]:
```
def show_outputs(MIRISim_outputdir,output_type):
    '''
    plot the specified channel of the MIRISim outputs
    :param MIRISim_outputdir:
        name of the date-labelled dir. holding the MIRISIM outputs
    :param output_type:
        type of output to process
        (e.g. illum_models, det_images or skycubes)
    '''

    infits = glob.glob('{}/{}/*.fits'.format(MIRISim_outputdir,output_type))[0]

    hdulist = fits.open(infits)
```

(continues on next page)

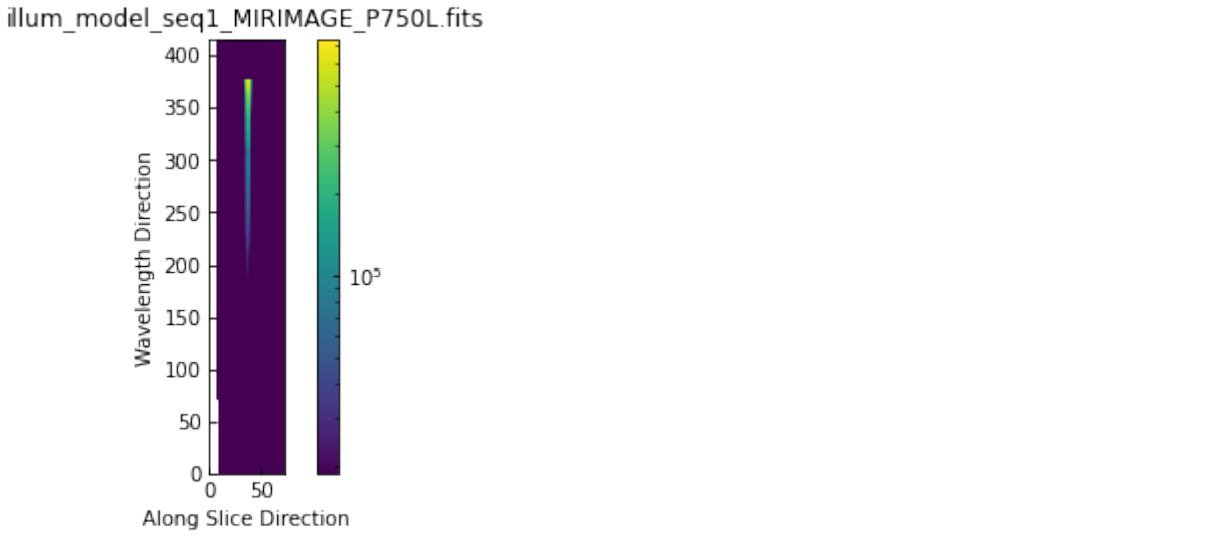
(continued from previous page)

```
    hdu_index = 1
    if len(hdulist[hdu_index].data.shape) > 3:
        integ,frames,nx,ny = hdulist[hdu_index].data.shape
        image = hdulist[hdu_index].data[integ-1,frames-1,:,:]


    else:
        image = hdulist[hdu_index].data[:,:]

    norm = colors.LogNorm(image.mean() + 0.5 * image.std(), image.max(), clip=True)
    plt.imshow(image,origin = 'lower', cmap = cm.viridis,interpolation = 'nearest',
↪norm = norm)
    plt.title('{}'.format(infits.split('/')[-1]))
    plt.xlabel('Along Slice Direction')
    plt.ylabel('Wavelength Direction')


    plt.colorbar()
```

16.5 Viewing an illumination model

Below shows an example of an output illumination model

[12]:
```
show_outputs(outputdir,'illum_models')
```



16.5.1 Viewing the final detector image

The detector images are the final output of MIRISim, and have data structures and formatting consistent with what will come from MIRI itself. The data format is JWST pipeline ready. Below an example image of the last frame of the last integration is shown. The units are DN.

Additionally, below the FITS header information for the SCIENCE extension is listed

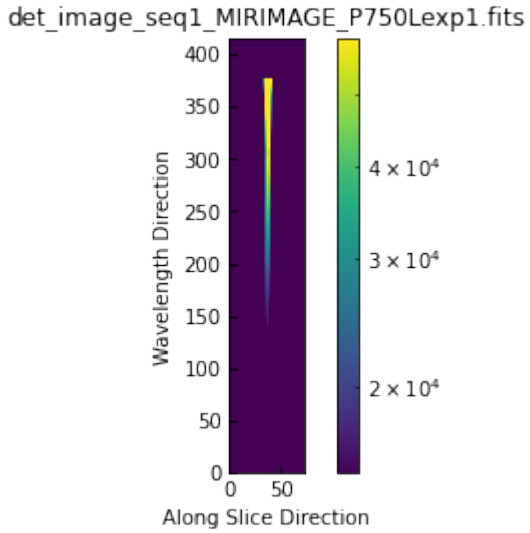
[13]:
```
show_outputs(outputdir,'det_images')

infits = glob.glob('{}/det_images/*.fits'.format(outputdir))[0]
hdulist = fits.open(infits)
hdulist[1].header
```

[13]:
```
XTENSION= 'IMAGE   '           / Image extension
BITPIX  =                  -32 / array data type
NAXIS   =                    4 / number of array dimensions
NAXIS1  =                   72
NAXIS2  =                  416
NAXIS3  =                   20
NAXIS4  =                    1
PCOUNT  =                    0 / number of parameters
GCOUNT  =                    1 / number of groups
EXTNAME = 'SCI     '           / extension name

        Information about the coordinates in the file

RADESYS = 'ICRS    '           / Name of the coordinate reference frame

        Information about the data array

BUNIT   = 'DN      '           / Units of the data array

        Spacecraft pointing information

RA_V1   =   0.1052311315773884 / [deg] RA of telescope V1 axis
DEC_V1  =  0.09581792869591674 / [deg] Dec of telescope V1 axis
PA_V3   =                -0.0 / [deg] Position angle of telescope V3 axis

        WCS parameters

WCSAXES =                    3 / number of World Coordinate System axes
CRPIX1  =                    0 / axis 1 coordinate of the reference pixel
CRPIX2  =                 -528 / axis 2 coordinate of the reference pixel
CRPIX3  =                    0 / axis 3 coordinate of the reference pixel
CRVAL1  =                  0.0 / first axis value at the reference pixel
CRVAL2  =                  0.0 / second axis value at the reference pixel
CRVAL3  =                  0.0 / third axis value at the reference pixel
CTYPE1  = '' / first axis coordinate type
CTYPE2  = '' / second axis coordinate type
CTYPE3  = '' / third axis coordinate type
CUNIT1  = '' / first axis units
CUNIT2  = '' / second axis units
CUNIT3  = '' / third axis units
CDELT1  =                  1.0 / first axis increment per pixel
CDELT2  =                  1.0 / second axis increment per pixel
CDELT3  =                  1.0 / third axis increment per pixel
V2_REF  =   -378.8320736785982 / [arcsec] Telescope v2 coordinate of the referen
```

(continues on next page)

(continued from previous page)

```
V3_REF  =   -344.9445433053003 / [arcsec] Telescope v3 coordinate of the referen
VPARITY =                   -1 / Relative sense of rotation between Ideal xy and
V3I_YANG=                  0.0 / [deg] Angle from V3 axis to Ideal y axis
RA_REF  =                  0.0 / [deg] Right ascension of the reference point
DEC_REF =                  0.0 / [deg] Declination of the reference point
ROLL_REF=                 -0.0 / [deg] V3 roll angle at the ref point (N over E)
EXTVER  =                    1 / extension value
```



[ ]:

1. A. Glasse, G. H. Rieke, E. Bauwens, M. Garc'ıa-Mar'ın, M. E. Ressler, S. Rost, T. V. Tikkanen, B. Vandenbussche, and G. S. Wright. The Mid-Infrared Instrument for the James Webb Space Telescope, IX: Predicted Sensitivity. *pasp* , 127:686, July 2015. arXiv:1508.02427, doi:10.1086/682259.

INDEX