Tutorial Series - NIRI Imaging Data Reduction with DRAGONS Documentation

Release 3.0.4

Kathleen Labrie

November 2022

Contents:

Document ID

PIPE-USER-119_NIRIImg-DRTutorial

CHAPTER 1

Overview

This is a collection of tutorials for the reduction of NIRI data with DRAGONS.

NIRI is a near-infrared imager. The spectroscopic features of NIRI have been decommissioned several years ago. As such, this tutorial series focuses on the reduction of NIRI imaging data.

In here are tutorials that you, the reader, can run and experiment with. This document comes with a downloadable data package that contains all the data you need to run the examples presented. Instructions on where to get that package and how to set things up are given in *Downloading the tutorial datasets*.

The NIRI tutorial series for the time being contains only one scientific example, the reduction of an extended source. This can be done in two different ways:

- From the terminal using the command line. (*Example 1-A*)
- From Python using the DRAGONS classes and functions. (*Example 1-B*)

We plan to add additional examples in the future.

CHAPTER 2

Setting up and tutorial datasets

2.1 Downloading the tutorial datasets

All the data needed to run this tutorial are found in the tutorial's data package:

http://www.gemini.edu/sciops/data/software/datapkgs/niriimg_tutorial_datapkg-v1.tar

Download it and unpack it somewhere convenient.

```
cd <somewhere convenient>
tar xvf niriimg_tutorial_datapkg-v1.tar
bunzip2 niriimg_tutorial/playdata/*.bz2
```

The datasets are found in the subdirectory `niriimg_tutorial/playdata`, and we will work in the subdirectory named `niriimg_tutorial/playground`.

Note: All the raw data can also be downloaded from the Gemini Observatory Archive. Using the tutorial data package is probably more convenient.

2.2 Datasets descriptions

2.2.1 Dataset for Example 1: Extended source with offset to sky

This is a NIRI imaging observation of an extended source, a galaxy showing as a dense field of stars. The observation sequence uses an offset to a nearby blank portion of the sky to monitor the sky levels since there are no area in the science observation that is not "contaminated" by the galaxy.

The calibrations we use for this example include:

- Darks for the science and sky offset frames.
- Flats, as a sequence of lamps-on and lamps-off exposures.

- Short darks to use with the flats to create a bad pixel mask.
- A set of standard star observations.

Here is the files breakdown. They are included in the tutorial data package. They can also be downloaded from the Gemini Observatory Archive (GOA).

Science	N20160102S0270-274 (on-target) N20160102S0275-279 (on-sky)
Science darks	N20160102S0423-432 (20 sec, like Science)
Flats	N20160102S0373-382 (lamps-on) N20160102S0363-372 (lamps-off)
Short darks	N20160103S0463-472
Standard star	N20160102S0295-299

A note about finding the short darks in the GOA. Those darks are used solely to create a fresh bad pixel mask (BPM). In the archive, the calibration association will not find those darks, they need to be searched for explicitly. If you need to find short darks for your program, do as follow:

- Set a date range around the dates of your science observations.
- Set **Instrument** to NIRI.
- Set **Obs.Type** to DARK.
- Set the exposure time to 1 second.

CHAPTER 3

Example 1-A: Extended source - Using the “reduce” command

In this example we will reduce a NIRI observation of an extended source using the “” command that is operated directly from the unix shell. Just open a terminal and load the DRAGONS conda environment to get started.

This observation is a simple dither on target, a galaxy, with offset to sky.

3.1 The dataset

If you have not already, download and unpack the tutorial’s data package. Refer to *Downloading the tutorial datasets* for the links and simple instructions.

The dataset specific to this example is described in:

> *Dataset for Example 1: Extended source with offset to sky*.

Here is a copy of the table for quick reference.

Science	N20160102S0270-274 (on-target) N20160102S0275-279 (on-sky)
Science darks	N20160102S0423-432 (20 sec, like Science)
Flats	N20160102S0373-382 (lamps-on) N20160102S0363-372 (lamps-off)
Short darks	N20160103S0463-472
Standard star	N20160102S0295-299

3.2 Set up the Local Calibration Manager

DRAGONS comes with a local calibration manager and a local light weight database that uses the same calibration association rules as the Gemini Observatory Archive. This allows "" to make requests for matching **processed** calibrations when needed to reduce a dataset.

Let's set up the local calibration manager for this session.

In `~/.geminidr/`, create or edit the configuration file `rsys.cfg` as follow:

```
[calibs]
standalone = True
database_dir = <where_the_data_package_is>/niriimg_tutorial/playground
```

This simply tells the system where to put the calibration database, the database that will keep track of the processed calibrations we are going to send to it.

Note: ~ in the path above refers to your home directory. Also, don't miss the dot in `.geminidr`.

Then initialize the calibration database:

```
caldb init
```

That's it. It is ready to use.

You can add processed calibrations with `caldb add <filename>` (we will later), list the database content with `caldb list`, and `caldb remove <filename>` to remove a file from the database (it will **not** remove the file on disk.) (See the "" documentation for more details.)

3.3 Create file lists

This data set contains science and calibration frames. For some programs, it could have different observed targets and different exposure times depending on how you like to organize your raw data.

The DRAGONS data reduction pipeline does not organize the data for you. You have to do it. DRAGONS provides tools to help you with that.

The first step is to create input file lists. The tool "" helps with that. It uses Astrodata tags and "" to select the files and send the filenames to a text file that can then be fed to "". (See the for information about Astrodata.)

First, navigate to the `playground` directory in the unpacked data package.

3.3.1 Two lists for the darks

We have two sets of darks; one set for the science frames, the 20-second darks, and another for making the BPM, the 1-second darks. We will create two lists.

If you did not know the exposure times for the darks, you could have use a combination of "" to select all the darks (tag `DARK`) and feed that list to "" to show descriptor values, in this case `exposure_time`. (See the page for a complete list.)

```
dataselect ../playdata/*.fits --tags DARK | showd -d exposure_time

------------------------------------------------
filename                          exposure_time
------------------------------------------------
../playdata/N20160102S0423.fits          20.002
../playdata/N20160102S0424.fits          20.002
../playdata/N20160102S0425.fits          20.002
../playdata/N20160102S0426.fits          20.002
../playdata/N20160102S0427.fits          20.002
../playdata/N20160102S0428.fits          20.002
../playdata/N20160102S0429.fits          20.002
../playdata/N20160102S0430.fits          20.002
../playdata/N20160102S0431.fits          20.002
../playdata/N20160102S0432.fits          20.002
../playdata/N20160103S0463.fits           1.001
../playdata/N20160103S0464.fits           1.001
../playdata/N20160103S0465.fits           1.001
../playdata/N20160103S0466.fits           1.001
../playdata/N20160103S0467.fits           1.001
../playdata/N20160103S0468.fits           1.001
../playdata/N20160103S0469.fits           1.001
../playdata/N20160103S0470.fits           1.001
../playdata/N20160103S0471.fits           1.001
../playdata/N20160103S0472.fits           1.001
```

As one can see above the exposure times all have a small fractional increment. This is just a floating point inaccuracy somewhere in the software that generates the raw NIRI FITS files. As far as we are concerned here in this tutorial, we are dealing with 20-second and 1-second darks. The tool "" is smart enough to match those exposure times as "close enough". So, in our selection expression, we can use "1" and "20" and ignore the extra digits.

Note: If a perfect match to 1.001 were required, adding the option `--strict` in `dataselect` would ensure an exact match.

Let's create our two lists now.

```
dataselect ../playdata/*.fits --tags DARK --expr='exposure_time==1' -o darks1s.lis
dataselect ../playdata/*.fits --tags DARK --expr='exposure_time==20' -o darks20s.lis
```

3.3.2 A list for the flats

The flats are a sequence of lamp-on and lamp-off exposures. We just send all of them to one list.

```
dataselect ../playdata/*.fits --tags FLAT -o flats.lis
```

3.3.3 A list for the standard star

The standard stars at Gemini are normally taken as partner calibration.

You can see the `observation_class` of all the data using "". Here we will print the object name too.

```
showd ../playdata/*.fits -d observation_class,object

------------------------------------------------------------
filename                          observation_class     object
------------------------------------------------------------
../playdata/N20160102S0270.fits             science    SN2014J
...
../playdata/N20160102S0295.fits          partnerCal      FS 17
../playdata/N20160102S0296.fits          partnerCal      FS 17
../playdata/N20160102S0297.fits          partnerCal      FS 17
../playdata/N20160102S0298.fits          partnerCal      FS 17
../playdata/N20160102S0299.fits          partnerCal      FS 17
../playdata/N20160102S0363.fits              dayCal   GCALflat
...
../playdata/N20160103S0472.fits              dayCal       Dark
```

The list is abridged for presentation.

Our standard star is a "partnerCal" named "FS 17". Since it is unique, we can use either criterion to get our list.

```
dataselect ../playdata/*.fits --expr='observation_class=="partnerCal"' -o stdstar.lis
```

Or

```
dataselect ../playdata/*.fits --expr='object=="FS 17"' -o stdstar.lis
```

3.3.4 A list for the science observations

The science frames are all the `IMAGE` non-`FLAT` frames that are also not the standard. Since flats are tagged `FLAT` and `IMAGE`, we need to exclude the `FLAT` tag.

This translates to the following expression:

```
dataselect ../playdata/*.fits --tags IMAGE --xtags FLAT --expr='object!="FS 17"' -o␣
↪target.lis
```

One could have used the name of the science target too, like we did for selecting the standard star observation in the previous section. The example above shows how to *exclude* a tag if needed and was considered more educational.

3.4 Master Dark

We first create the master dark for the science target, then add it to the calibration database. The name of the output master dark, `N20160102S0423_dark.fits`, is written to the screen at the end of the process.

```
reduce @darks20s.lis
caldb add N20160102S0423_dark.fits
```

The `@` character before the name of the input file is the "at-file" syntax. More details can be found in the documentation.

Note: The file name of the output processed dark is the file name of the first file in the list with *_dark* appended as a suffix. This the general naming scheme used by "".

3.5 Bad Pixel Mask

The DRAGONS Gemini data reduction package, `geminidr`, comes with a static NIRI bad pixel mask (BPM) that gets automatically added to all the NIRI data as they gets processed. The user can also create a *supplemental*, fresher BPM from the flats and recent short darks. That new BPM is later fed to "" as a *user BPM* to be combined with the static BPM. Using both the static and a fresh BPM from recent data lead to a better representation of the bad pixels. It is an optional but recommended step.

The flats and the short darks are the inputs.

The flats must be passed first to the input list to ensure that the recipe library associated with NIRI flats is selected. We will not use the default recipe but rather the special recipe from that library called `makeProcessedBPM`.

```
reduce @flats.lis @darks1s.lis -r makeProcessedBPM
```

The BPM produced is named `N20160102S0373_bpm.fits`.

The local calibration manager does not yet support BPMs so we cannot add it to the database. It is a future feature. Until then we have to pass it manually to "" to use it, as we will show below.

3.6 Master Flat Field

A NIRI master flat is created from a series of lamp-on and lamp-off exposures. Each flavor is stacked, then the lamp-off stack is subtracted from the lamp-on stack.

We create the master flat field and add it to the calibration database as follow:

```
reduce @flats.lis -p addDQ:user_bpm=N20160102S0373_bpm.fits
caldb add N20160102S0373_flat.fits
```

Note how we pass in the BPM we created in the previous step. The `addDQ` primitive, one of the primitives in the recipe, has an input parameter named `user_bpm`. We assign our BPM to that input parameter.

To see the list of available input parameters and their defaults, use the tool "". It needs the name of a file on which the primitive will be run because the defaults are adjusted to match the input data.

```
showpars ../playdata/N20160102S0363.fits addDQ
```

```
Dataset tagged as set(['FLAT', 'LAMPOFF', 'NORTH', 'AZEL_TARGET', 'IMAGE', 'GCAL
_IR_OFF', 'GCALFLAT', 'RAW', 'GEMINI', 'NON_SIDEREAL', 'CAL', 'UNPREPARED', 'NIR
I'])
Settable parameters on 'addDQ':
========================================
 Name                   Current setting

suffix                '_dqAdded'           Filename suffix
non_linear            False                Flag non-linear pixels?
time                  120.0                Persistence time (seconds)
        Valid Range = [0.0,inf)
illum_mask            None                 Name of illumination mask
static_bpm            'default'            Static bad pixel mask
user_bpm              None                 User bad pixel mask
add_illum_mask        False                Apply illumination mask?
latency               True                 Apply latency for saturated pixels?
```

3.7 Standard Star

The standard star is reduced more or less the same way as the science target (next section) except that darks frames are not obtained for standard star observations. Therefore the dark correction needs to be turned off.

The processed flat field that we added earlier to the local calibration database will be fetched automatically. The user BPM (optional, but recommended) needs to be specified by the user.

```
reduce @stdstar.lis -p addDQ:user_bpm=N20160102S0373_bpm.fits darkCorrect:do_cal=skip
```

3.8 Science Observations

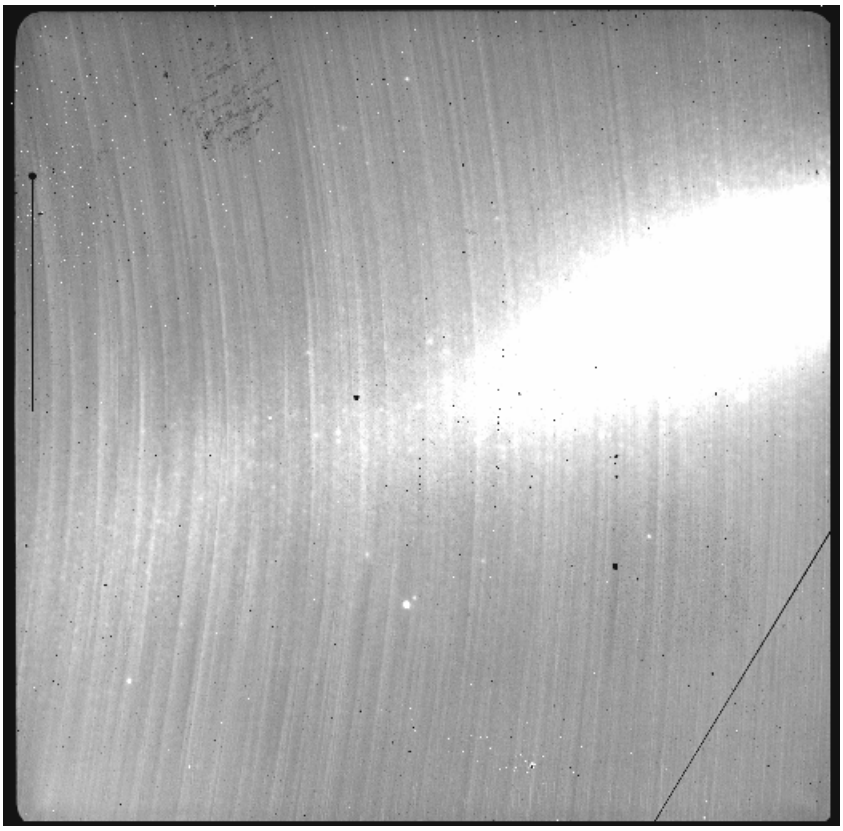
The science target is an extended source. We need to turn off the scaling of the sky because the target fills the field of view and does not represent a reasonable sky background. If scaling is not turned off *in this particular case*, it results in an over-subtraction of the sky frame.

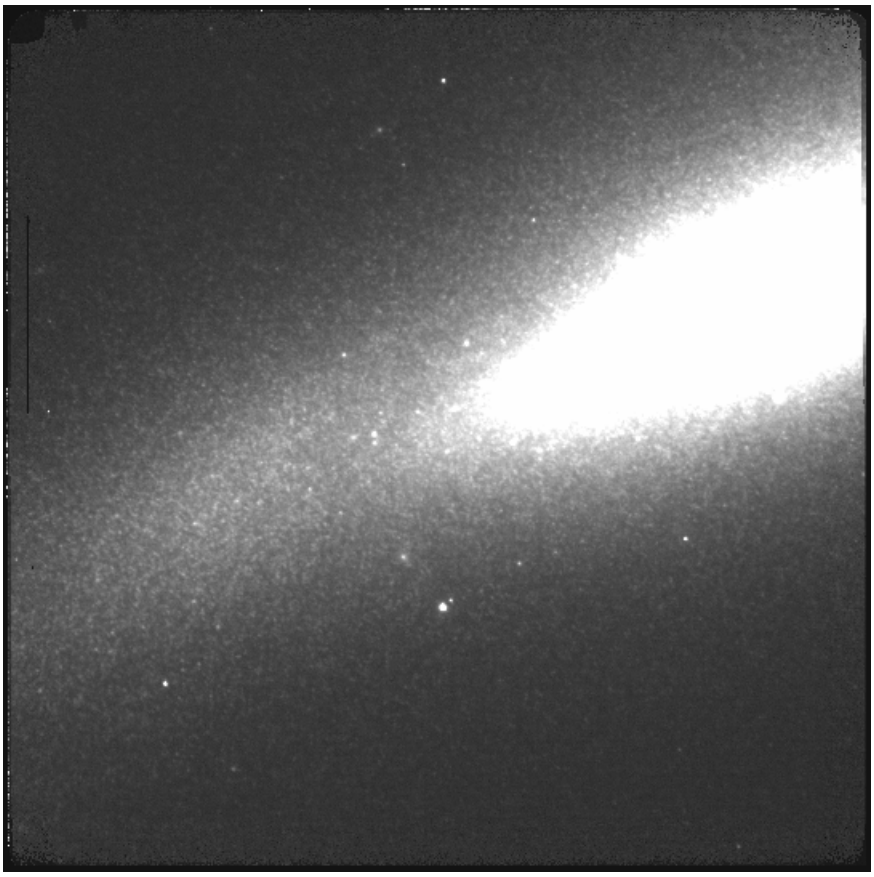
The sky frame comes from off-target sky observations. We feed the pipeline all the on-target and off-target frames. The software will split the on-target and the off-target appropriately.

The master dark and the master flat will be retrieved automatically from the local calibration database. Again, the user BPM needs to be specified on the command line.

The output stack units are in electrons (header keyword BUNIT=electrons). The output stack is stored in a multi-extension FITS (MEF) file. The science signal is in the "SCI" extension, the variance is in the "VAR" extension, and the data quality plane (mask) is in the "DQ" extension.

```
reduce @target.lis -p addDQ:user_bpm=N20160102S0373_bpm.fits skyCorrect:scale_
↪sky=False
```





The attentive reader will note that the reduced image is slightly larger than the individual raw image. This is because of the telescope was dithered between each observation leading to a slightly larger final field of view than that of each individual image. The stacked product is *not* cropped to the common area, rather the image size is adjusted to include the complete area covered by the whole sequence. Of course the areas covered by less than the full stack of images will have a lower signal-to-noise. The final MEF file has three named extensions, the science (SCI), the variance (VAR), and the data quality plane (DQ).

CHAPTER 4

Example 1-B: Extended source - Using the “Reduce” class

A reduction can be initiated from the command line as shown in *Example 1-A: Extended source - Using the “reduce” command* and it can also be done programmatically as we will show here. The classes and modules of the RecipeSystem can be accessed directly for those who want to write Python programs to drive their reduction. In this example we replicate the command line reduction from Example 1-A, this time using the Python interface instead of the command line. Of course what is shown here could be packaged in modules for greater automation.

4.1 The dataset

If you have not already, download and unpack the tutorial’s data package. Refer to *Downloading the tutorial datasets* for the links and simple instructions.

The dataset specific to this example is described in:

> *Dataset for Example 1: Extended source with offset to sky*.

Here is a copy of the table for quick reference.

Science	N20160102S0270-274 (on-target) N20160102S0275-279 (on-sky)
Science darks	N20160102S0423-432 (20 sec, like Science)
Flats	N20160102S0373-382 (lamps-on) N20160102S0363-372 (lamps-off)
Short darks	N20160103S0463-472
Standard star	N20160102S0295-299

4.2 Setting up

First, navigate to your work directory in the unpacked data package.

The first steps are to import libraries, set up the calibration manager, and set the logger.

4.2.1 Importing libraries

```
1  import glob
2
3  import astrodata
4  import gemini_instruments
5  from recipe_system.reduction.coreReduce import Reduce
6  from recipe_system import cal_service
7  from gempy.adlibrary import dataselect
```

The `dataselect` module will be used to create file lists for the darks, the flats and the science observations. The `cal_service` package is our interface to the local calibration database. Finally, the `Reduce` class is used to set up and run the data reduction.

4.2.2 Setting up the logger

We recommend using the DRAGONS logger. (See also *Double messaging issue*.)

```
9  from gempy.utils import logutils
10 logutils.config(file_name='niri_tutorial.log')
```

4.2.3 Set up the Local Calibration Manager

DRAGONS comes with a local calibration manager and a local, light weight database that uses the same calibration association rules as the Gemini Observatory Archive. This allows the `Reduce` instance to make requests for matching **processed** calibrations when needed to reduce a dataset.

Let's set up the local calibration manager for this session.

In `~/.geminidr/`, edit the configuration file `rsys.cfg` as follow:

```
[calibs]
standalone = True
database_dir = <where_the_data_package_is>/niriimg_tutorial/playground
```

This tells the system where to put the calibration database, the database that will keep track of the processed calibration we are going to send to it.

Note: The tilde (~) in the path above refers to your home directory. Also, mind the dot in `.geminidr`.

The calibration database is initialized and the calibration service is configured like this:

```
caldb = cal_service.CalibrationService()
caldb.config()
caldb.init()

cal_service.set_calservice()
```

The calibration service is now ready to use. If you need more details, check the "" documentation in the Recipe System User Manual.

4.3 Create file lists

The next step is to create input file lists. The module `dataselect` helps with that. It uses Astrodata tags and to select the files and store the filenames to a Python list that can then be fed to the `Reduce` class. (See the for information about Astrodata and for a list of .)

The first list we create is a list of all the files in the `playdata` directory.

```
all_files = glob.glob('../playdata/*.fits')
all_files.sort()
```

We will search that list for files with specific characteristics. We use the `all_files list` as an input to the function `dataselect.select_data()` . The function's signature is:

```
select_data(inputs, tags=[], xtags=[], expression='True')
```

We show several usage examples below.

4.3.1 Two lists for the darks

We have two sets of darks; one set for the science frames, the 20-second darks, and another for making the BPM, the 1-second darks. We will create two lists.

If you did not know the exposure times for the darks, you could use `dataselect` as follows to see the exposure times of all the darks in the directory. We use the tag `DARK` and the descriptor `exposure_time`.

```
all_darks = dataselect.select_data(all_files, ['DARK'])
for dark in all_darks:
    ad = astrodata.open(dark)
    print(dark, '  ', ad.exposure_time())
```

```
../playdata/N20160102S0423.fits    20.002
../playdata/N20160102S0424.fits    20.002
../playdata/N20160102S0425.fits    20.002
../playdata/N20160102S0426.fits    20.002
../playdata/N20160102S0427.fits    20.002
../playdata/N20160102S0428.fits    20.002
../playdata/N20160102S0429.fits    20.002
../playdata/N20160102S0430.fits    20.002
../playdata/N20160102S0431.fits    20.002
../playdata/N20160102S0432.fits    20.002
../playdata/N20160103S0463.fits    1.001
../playdata/N20160103S0464.fits    1.001
../playdata/N20160103S0465.fits    1.001
../playdata/N20160103S0466.fits    1.001
../playdata/N20160103S0467.fits    1.001
../playdata/N20160103S0468.fits    1.001
../playdata/N20160103S0469.fits    1.001
../playdata/N20160103S0470.fits    1.001
../playdata/N20160103S0471.fits    1.001
../playdata/N20160103S0472.fits    1.001
```

As one can see above the exposure times all have a small fractional increment. This is just a floating point inaccuracy somewhere in the software that generates the raw NIRI FITS files. As far as we are concerned here in this tutorial, we are dealing with 20-second and 1-second darks. The function `dataselect` is smart enough to match those exposure times as "close enough". So, in our selection expression, we can use "1" and "20" and ignore the extra digits.

Note: If a perfect match to 1.001 were required, simply set the argument `strict` to `True` in `dataselect.expr_parser`, eg. `dataselect.expr_parser(expression, strict=True)`.

Let us create our two lists now. The filenames will be stored in the variables `darks1s` and `darks20s`.

```
darks1s = dataselect.select_data(
    all_files,
    ['DARK'],
    [],
    dataselect.expr_parser('exposure_time==1')
)

darks20s = dataselect.select_data(
    all_files,
    ['DARK'],
    [],
    dataselect.expr_parser('exposure_time==20')
)
```

Note: All expression need to be processed with `dataselect.expr_parser`.

4.3.2 A list for the flats

The flats are a sequence of lamp-on and lamp-off exposures. We just send all of them to one list.

```
flats = dataselect.select_data(all_files, ['FLAT'])
```

4.3.3 A list for the standard star

The standard star sequence is a series of datasets identified as "FS 17". There are no keywords in the NIRI header identifying this target as a special standard star target. We need to use the target name to select only observations from that star and not our science target.

```
stdstar = dataselect.select_data(
    all_files,
    [],
    [],
    dataselect.expr_parser('object=="FS 17"')
)
```

4.3.4 A list for the science observations

The science frames are all `IMAGE` non-`FLAT` that are also not the standard. Since flats are tagged `FLAT` and `IMAGE`, we need to exclude the `FLAT` tag.

This translate to the following sequence:

```
target = dataselect.select_data(
    all_files,
    ['IMAGE'],
    ['FLAT'],
    dataselect.expr_parser('object!="FS 17"')
)
```

One could have used the name of the science target too, like we did for selecting the standard star observation in the previous section. The example above shows how to *exclude* a tag if needed and was considered more educational.

4.4 Master Dark

We first create the master dark for the science target, then add it to the calibration database. The name of the output master dark is `N20160102S0423_dark.fits`. The output is written to disk and its name is stored in the `Reduce` instance. The calibration service expects the name of a file on disk.

```
reduce_darks = Reduce()
reduce_darks.files.extend(darks20s)
reduce_darks.runr()

caldb.add_cal(reduce_darks.output_filenames[0])
```

The `Reduce` class is our reduction "controller". This is where we collect all the information necessary for the reduction. In this case, the only information necessary is the list of input files which we add to the `files` attribute. The `Reduce.runr{}` method is where the recipe search is triggered and where it is executed.

Note: The file name of the output processed dark is the file name of the first file in the list with *_dark* appended as a suffix. This the general naming scheme used by the `Recipe System`.

4.5 Bad Pixel Mask

The DRAGONS Gemini data reduction package, `geminidr`, comes with a static NIRI bad pixel mask (BPM) that gets automatically added to all the NIRI data as they get processed. The user can also create a *supplemental*, fresher BPM from the flats and recent short darks. That new BPM is later fed to the reduction process as a *user BPM* to be combined with the static BPM. Using both the static and a fresh BPM from recent data lead to a better representation of the bad pixels. It is an optional but recommended step.

The flats and the short darks are the inputs.

The flats must be passed first to the input list to ensure that the recipe library associated with NIRI flats is selected. We will not use the default recipe but rather the special recipe from that library called `makeProcessedBPM`.

```
reduce_bpm = Reduce()
reduce_bpm.files.extend(flats)
reduce_bpm.files.extend(darks1s)
reduce_bpm.recipename = 'makeProcessedBPM'
reduce_bpm.runr()

bpm = reduce_bpm.output_filenames[0]
```

The BPM produced is named `N20160102S0373_bpm.fits`.

The local calibration manager does not yet support BPMs so we cannot add it to the database. It is a future feature. Until then we have to pass it manually to the `Reduce` instance to use it, as we will show below.

4.6 Master Flat Field

A NIRI master flat is created from a series of lamp-on and lamp-off exposures. Each flavor is stacked, then the lamp-off stack is subtracted from the lamp-on stack.

We create the master flat field and add it to the calibration database as follow:

```
reduce_flats = Reduce()
reduce_flats.files.extend(flats)
reduce_flats.uparms = [('addDQ:user_bpm', bpm)]
reduce_flats.runr()

caldb.add_cal(reduce_flats.output_filenames[0])
```

Note how we pass in the BPM we created in the previous step. The `addDQ` primitive, one of the primitives in the recipe, has an input parameter named `user_bpm`. We assign our BPM to that input parameter. The value of `uparms` needs to be a `list` of `Tuples`.

To see the list of available input parameters and their defaults, use the command line tool `showpars` from a terminal. It needs the name of a file on which the primitive will be run because the defaults are adjusted to match the input data.

```
showpars ../playdata/N20160102S0363.fits addDQ
```

```
Dataset tagged as set(['FLAT', 'LAMPOFF', 'NORTH', 'AZEL_TARGET', 'IMAGE', 'GCAL
_IR_OFF', 'GCALFLAT', 'RAW', 'GEMINI', 'NON_SIDEREAL', 'CAL', 'UNPREPARED', 'NIR
I'])
Settable parameters on 'addDQ':
=======================================
 Name                    Current setting

suffix                '_dqAdded'           Filename suffix
non_linear            False                Flag non-linear pixels?
time                  120.0                Persistence time (seconds)
        Valid Range = [0.0,inf)
illum_mask            None                 Name of illumination mask
static_bpm            'default'            Static bad pixel mask
user_bpm              None                 User bad pixel mask
add_illum_mask        False                Apply illumination mask?
latency               True                 Apply latency for saturated pixels?
```

4.7 Standard Star

The standard star is reduced more or less the same way as the science target (next section) except that dark frames are not obtained for standard star observations. Therefore the dark correction needs to be turned off.

The processed flat field that we added earlier to the local calibration database will be fetched automatically. The user BPM (optional, but recommended) needs to be specified by the user.

```
reduce_std = Reduce()
reduce_std.files.extend(stdstar)
reduce_std.uparms = [('addDQ:user_bpm', bpm)]
reduce_std.uparms.append(('darkCorrect:do_cal', 'skip'))
reduce_std.runr()
```

4.8 Science Observations

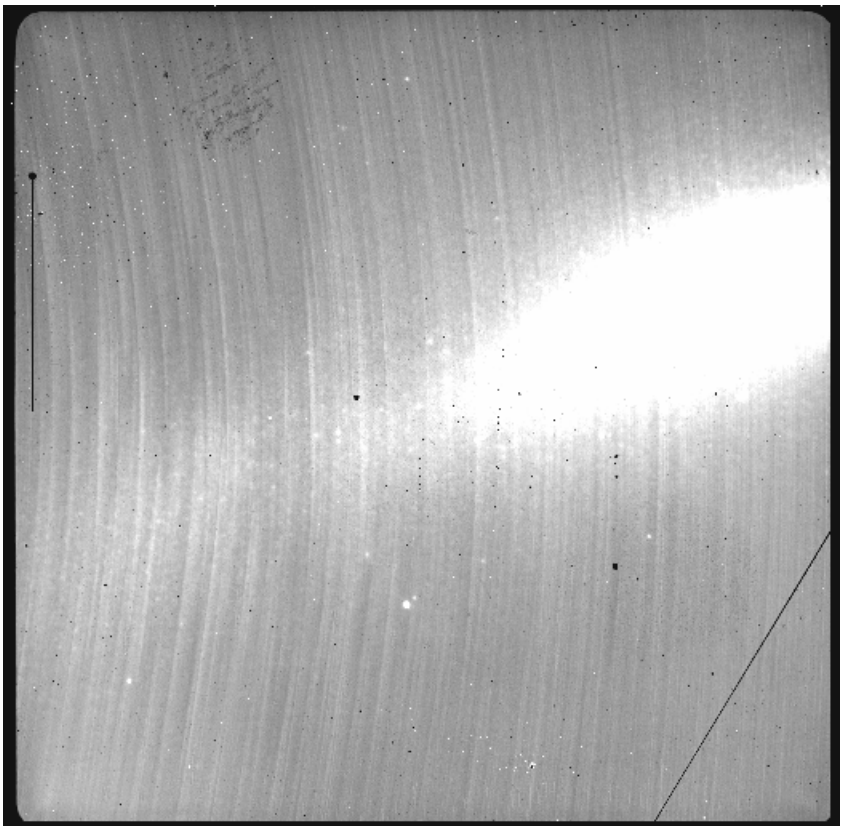
The science target is an extended source. We need to turn off the scaling of the sky because the target fills the field of view and does not represent a reasonable sky background. If scaling is not turned off in this particular case, it results in an over-subtraction of the sky frame.

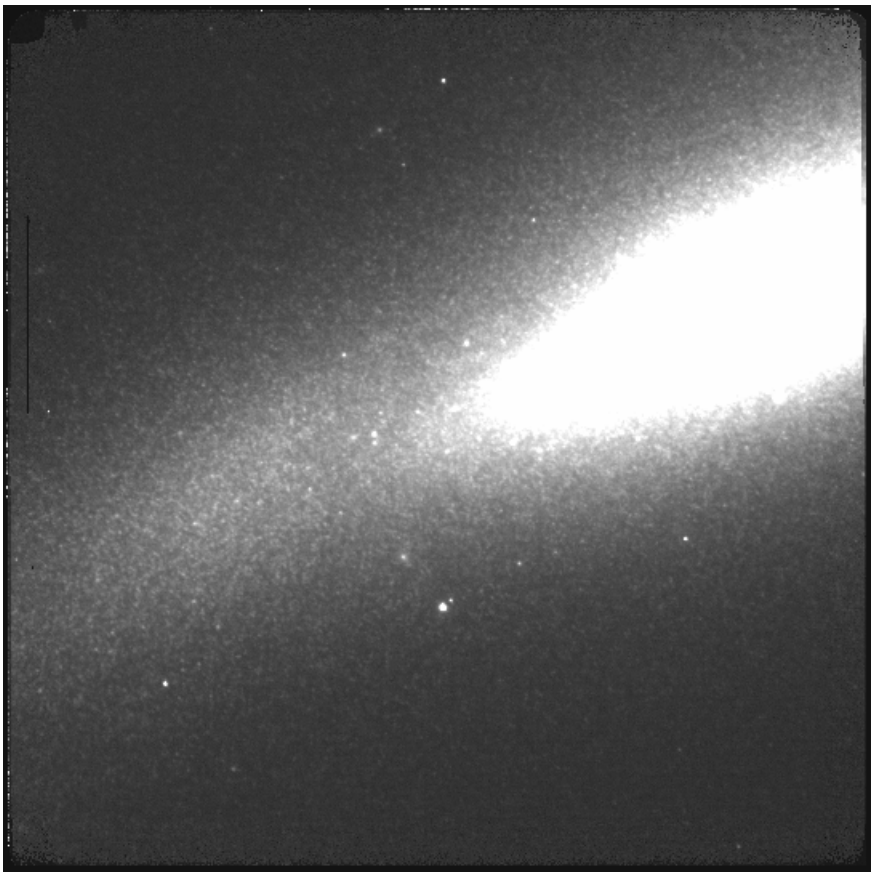
The sky frame comes from off-target sky observations. We feed the pipeline all the on-target and off-target frames. The software will split the on-target and the off-target appropriately.

The master dark and the master flat will be retrieved automatically from the local calibration database. Again, the user BPM needs to be specified as the `user_bpm` argument to `addDQ`.

The output stack units are in electrons (header keyword BUNIT=electrons). The output stack is stored in a multi-extension FITS (MEF) file. The science signal is in the "SCI" extension, the variance is in the "VAR" extension, and the data quality plane (mask) is in the "DQ" extension.

```
reduce_target = Reduce()
reduce_target.files.extend(target)
reduce_target.uparms = [('addDQ:user_bpm', bpm)]
reduce_target.uparms.append(('skyCorrect:scale_sky', False))
reduce_target.runr()
```





The attentive reader will note that the reduced image is slightly larger than the individual raw image. This is because of the telescope was dithered between each observation leading to a slightly larger final field of view than that of each individual image. The stacked product is *not* cropped to the common area, rather the image size is adjusted to include the complete area covered by the whole sequence. Of course the areas covered by less than the full stack of images will have a lower signal-to-noise.

CHAPTER 5

Tips and Tricks

This is a collection of tips and tricks that can be useful for reducing different data, or to do it slightly differently from what is presented in the example.

5.1 Bypassing automatic calibration association

We can think of two reasons why a user might want to bypass the calibration manager and the automatic processed calibration association. The first is to override the automatic selection, to force the use of a different processed calibration than what the system finds. The second is if there is a problem with the calibration manager and it is not working for some reason.

Whatever the specific situation, the following syntax can be used to bypass the calibration manager and set the input processed calibration yourself:

```
$ reduce @target.lis --user_cal processed_dark:N20160102S0423_dark.fits processed_
↪flat:N20160102S0373_flat.fits
```

The list of recognized processed calibration is:

- processed_arc
- processed_bias
- processed_dark
- processed_flat
- processed_fringe
- processed_standard

CHAPTER 6

Issues and Limitations

6.1 Memory Issues

Some primitives use a lot of RAM memory and they can cause a crash. Memory management in Python is notoriously difficult. The DRAGONS's team is constantly trying to improve memory management within `astrodata` and the DRAGONS recipes and primitives. If an "Out of memory" crash happens to you, if possible for your observation sequence, try to run the pipeline on fewer images at the time, like for each dither pattern sequence separately.

Then to align and stack the pieces, run the `alignAndStack` recipe:

```
$ reduce @list_of_stacks -r alignAndStack
```

For NIRI, this issue is relatively rare given that the NIRI detector is fairly small, but it could happen when trying to reduce a very large number of frames in one go.

6.2 Double messaging issue

If you run `Reduce` without setting up a logger, you will notice that the output messages appear twice. To prevent this behaviour set up a logger. This will send one of the output stream to a file, keeping the other on the screen. We recommend using the DRAGONS logger located in the `logutils` module and its `config()` function:

```
1 from gempy.utils import logutils
2 logutils.config(file_name='niri_tutorial.log')
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search