CASA Next Generation Infrastructure Documentation

Release 0.1b

Ryan Raba

Feb 07, 2020

CONTENTS

CHAPTER

ONE

README

Installation, setup and development rules for the CASA Next Generation Infrastructure

1.1 Documentation

API and User Manual: https://cngi-prototype.readthedocs.io

Visibility Processing Overview: https://colab.research.google.com/github/casangi/examples/blob/master/Visibility_overview.ipynb

Image Processing Overview: https://colab.research.google.com/github/casangi/examples/blob/master/Image_overview.ipynb

1.2 Organization

CNGI is organized in to modules as described below. Each module is responsible for a different functional area, such as file conversion, input / output, Visibility data operations, and Image data operations.

- conversion
- dio
- vis
- image
- direct
- gridding

1.3 Installation Requirements

Conversion functions depend on casatools from CASA 6 which is currently compatible with Python 3.6 only. CASA 6 also requires FORTRAN libraries be installed in the OS. These are not included in the dependency list so that the rest of CNGI functionality may be used without these constraints.

In the future, the conversion module may be moved outside of the CNGI package and distributed separately.

1.4 Pip Installation

```
python3 -m venv cngi
source cngi/bin/activate
#(maybe) sudo apt-get install libgfortran3
pip install --index-url https://casa-pip.nrao.edu/repository/pypi-casa-release/simple␣
↪casatools==6.0.0.27
pip install cngi-prototype
```

1.5 Conda Installation

```
conda create -n cngi python=3.6
conda activate cngi
#(maybe) sudo apt-get install libgfortran3
pip install --index-url https://casa-pip.nrao.edu/repository/pypi-casa-release/simple␣
↪casatools==6.0.0.27
pip install cngi-prototype
```

1.6 Installation from Source

```
git clone https://github.com/casangi/cngi_prototype.git
cd cngi_prototype
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python setup.py install --root=.
```

1.7 Building Documentation from Source

Follow steps to install cngi from source. Then navigate to the docs folder and execute the following:

```
sphinx-build -b html . ./build
```

View the documentation in your browser by navigating to:

```
file:///path/to/cngi/cngi_prototype/docs/build/index.html
```

1.8 Running CNGI in Parallel using Dask.distributed

To avoid thread collisions, when using the Dask.distributed Client, set the following environment variables.

```
export OMP_NUM_THREADS=1
export MKL_NUM_THREADS=1
export OPENBLAS_NUM_THREADS=1
```

1.9 Usage

You can import things several different ways. For example:

```
>>> from cngi import dio
>>> df = dio.read_image(...)
```

or

```
>>> from cngi.dio import read_image
>>> df = read_image(...)
```

or

```
>>> import cngi.dio as cdio
>>> df = cdio.read_image(...)
```

1.10 Run Tests

Download the test data from https://astrocloud.nrao.edu/s/Hacr42aZmJ3eb7i (or AWS S3) and place the files in `cngi_prototype/cngi/data/`. The test scripts can be found in `cngi_prototype/tests/`.

1.11 Design

READ THIS BEFORE YOU CONTRIBUTE CODE!!!

The CNGI code base is not object oriented, and instead follows a more functional paradigm. Objects are indeed used to hold Visibility and Image data, but they come directly from the underlying xarray/dask framework and are not extended in any way. The API consists of stateless Python functions only. They take in an Visibility or Image object and return a new Visibility or Image object with no global variables.

The cngi_prototype *repository* contains the cngi *package* along with supporting folders docs and tests. Within the cngi package there are a number of modules. Within each module there are one or more python files. CNGI adheres to a strict design philosophy with the following **RULES**:

1. Each file in a module must have exactly one function exposed to the external API (by docstring and __init__.py). The exposed function name should match the file name. This must be a stateless function, not a class.
2. Files in a module cannot import each other.
3. Files in separate modules cannot import each other.
4. A single special _helper module exists for internal functions meant to be shared across modules/files. But each module file should be as self contained as possible.
5. Nothing in _helper may be exposed to the external API.

```
cngi_prototype
|-- cngi
|    |-- module1
|    |     |-- __init__.py
|    |     |-- file1.py
|    |     |-- file2.py
|    |     | ...
```

(continues on next page)

(continued from previous page)

```
|    |-- module2
|    |      |-- __init__.py
|    |      |-- file3.py
|    |      |-- file4.py
|    |      | ...
|    |-- _helper
|    |      |-- __init__.py
|    |      |-- file5.py
|    |      |-- file6.py
|    |      | ...
|-- docs
|    | ...
|-- tests
|    | ...
|-- requirements.txt
|-- setup.py
```

File1, file2, file3 and file4 MUST be documented in the API exactly as they appear. They must NOT import each other.

File5 and file6 must NOT be documented in the API. They may be imported by file1 - 4.

__init__.py dictates what is seen by the API and importable by other functions.

1.12 Coding Standards

Documentation is generated using Sphinx, with the autodoc and napoleon extensions enabled. Function docstrings should be written in NumPy style. For compatibility with Sphinx, import statements should generally be underneath function definitions, not at the top of the file.

A complete set of formal and enforced coding standards have not yet been formally adopted. Some alternatives under consideration are:

- Google's style guide
- Python Software Foundation's style guide
- Following conventions established by PyData projects (examples one and two)

We are evaluating the adoption of PEP 484 convention, mypy, or param for type-checking, and flake8 or pylint for enforcement.

1.12.1 README

Installation, setup and development rules for the CASA Next Generation Infrastructure

Documentation

API and User Manual: https://cngi-prototype.readthedocs.io

Visibility Processing Overview: https://colab.research.google.com/github/casangi/examples/blob/master/Visibility_overview.ipynb

Image Processing Overview: https://colab.research.google.com/github/casangi/examples/blob/master/Image_overview.ipynb

Organization

CNGI is organized in to modules as described below. Each module is responsible for a different functional area, such as file conversion, input / output, Visibility data operations, and Image data operations.

- conversion
- dio
- vis
- image
- direct
- gridding

Installation Requirements

Conversion functions depend on casatools from CASA 6 which is currently compatible with Python 3.6 only. CASA 6 also requires FORTRAN libraries be installed in the OS. These are not included in the dependency list so that the rest of CNGI functionality may be used without these constraints.

In the future, the conversion module may be moved outside of the CNGI package and distributed separately.

Pip Installation

```
python3 -m venv cngi
source cngi/bin/activate
#(maybe) sudo apt-get install libgfortran3
pip install --index-url https://casa-pip.nrao.edu/repository/pypi-casa-release/simple␣
↪casatools==6.0.0.27
pip install cngi-prototype
```

Conda Installation

```
conda create -n cngi python=3.6
conda activate cngi
#(maybe) sudo apt-get install libgfortran3
pip install --index-url https://casa-pip.nrao.edu/repository/pypi-casa-release/simple␣
↪casatools==6.0.0.27
pip install cngi-prototype
```

Installation from Source

```
git clone https://github.com/casangi/cngi_prototype.git
cd cngi_prototype
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python setup.py install --root=.
```

Building Documentation from Source

Follow steps to install cngi from source. Then navigate to the docs folder and execute the following:

```
sphinx-build -b html . ./build
```

View the documentation in your browser by navigating to:

```
file:///path/to/cngi/cngi_prototype/docs/build/index.html
```

Running CNGI in Parallel using Dask.distributed

To avoid thread collisions, when using the Dask.distributed Client, set the following environment variables.

```
export OMP_NUM_THREADS=1
export MKL_NUM_THREADS=1
export OPENBLAS_NUM_THREADS=1
```

Usage

You can import things several different ways. For example:

```
>>> from cngi import dio
>>> df = dio.read_image(...)
```

or

```
>>> from cngi.dio import read_image
>>> df = read_image(...)
```

or

```
>>> import cngi.dio as cdio
>>> df = cdio.read_image(...)
```

Run Tests

Download the test data from https://astrocloud.nrao.edu/s/Hacr42aZmJ3eb7i (or AWS S3) and place the files in `cngi_prototype/cngi/data/`. The test scripts can be found in `cngi_prototype/tests/`.

Design

READ THIS BEFORE YOU CONTRIBUTE CODE!!!

The CNGI code base is not object oriented, and instead follows a more functional paradigm. Objects are indeed used to hold Visibility and Image data, but they come directly from the underlying xarray/dask framework and are not extended in any way. The API consists of stateless Python functions only. They take in an Visibility or Image object and return a new Visibility or Image object with no global variables.

The cngi_prototype *repository* contains the cngi *package* along with supporting folders docs and tests. Within the cngi package there are a number of modules. Within each module there are one or more python files. CNGI adheres to a strict design philosophy with the following **RULES**:

1. Each file in a module must have exactly one function exposed to the external API (by docstring and __init__.py). The exposed function name should match the file name. This must be a stateless function, not a class.
2. Files in a module cannot import each other.
3. Files in separate modules cannot import each other.
4. A single special _helper module exists for internal functions meant to be shared across modules/files. But each module file should be as self contained as possible.
5. Nothing in _helper may be exposed to the external API.

```
cngi_prototype
|-- cngi
|    |-- module1
|    |     |-- __init__.py
|    |     |-- file1.py
|    |     |-- file2.py
|    |     | ...
|    |-- module2
|    |     |-- __init__.py
|    |     |-- file3.py
|    |     |-- file4.py
|    |     | ...
|    |-- _helper
|    |     |-- __init__.py
|    |     |-- file5.py
|    |     |-- file6.py
|    |     | ...
|-- docs
|    | ...
|-- tests
|    | ...
|-- requirements.txt
|-- setup.py
```

File1, file2, file3 and file4 MUST be documented in the API exactly as they appear. They must NOT import each other.

File5 and file6 must NOT be documented in the API. They may be imported by file1 - 4.

__init__.py dictates what is seen by the API and importable by other functions.

Coding Standards

Documentation is generated using Sphinx, with the autodoc and napoleon extensions enabled. Function docstrings should be written in NumPy style. For compatibility with Sphinx, import statements should generally be underneath function definitions, not at the top of the file.

A complete set of formal and enforced coding standards have not yet been formally adopted. Some alternatives under consideration are:

- Google's style guide
- Python Software Foundation's style guide
- Following conventions established by PyData projects (examples one and two)

We are evaluating the adoption of PEP 484 convention, mypy, or param for type-checking, and flake8 or pylint for enforcement.

1.12.2 Conversion

Legacy CASA uses a custom MS format while CNGI uses the standard Zarr format. These functions allow conversion between the two as well as directly from the telescope archival science data model (ASDM) (future growth). Note that both the MS and Zarr formats are directories, not single files.

This package has a dependency on legacy CASA / casacore and will be separated in the future to its own distribution apart from the rest of the CNGI package.

To access these functions, use your favorite variation of: `import cngi.conversion`

asdm_to_zarr	
image_to_zarr	Convert legacy format Image or FITS format image to xarray Image Dataset compatible zarr format
ms_to_zarr	Convert legacy format MS to xarray Visibility Dataset compatible zarr format
ms_to_zarr_numba	Convert legacy format MS to xarray Visibility Dataset compatible zarr format
read_legacy_ms	
zarr_to_asdm	
zarr_to_image	
zarr_to_ms	

asdm_to_zarr

asdm_to_zarr(*infile*, *outfile=None*)

> **Todo:** This function is not yet implemented
>
> Convert ASDM format to xarray Visibility Dataset zarr format (future)
>
> **Parameters**
>
> - **infile** (*str*) – Input ASDM filename
> - **outfile** (*str*) – Output zarr filename. If None, will use infile name with .zarr extension
>
> **Returns** Success status
>
> **Return type** bool

image_to_zarr

image_to_zarr(*infile*, *outfile=None*, *artifacts=None*)

Convert legacy format Image or FITS format image to xarray Image Dataset compatible zarr format

This function requires CASA6 casatools module.

Parameters

- **infile** (*str*) – Input image filename (.image or .fits format)
- **outfile** (*str*) – Output zarr filename. If None, will use infile name with .img.zarr extension

- **artifacts** (*list of str*) – List of other image artifacts to include if present with infile. Default None uses `['mask','model','pb','psf','residual', 'sumwt','weight']`

ms_to_zarr

ms_to_zarr(*infile*, *outfile=None*, *ddi=None*, *compressor=None*, *chunk_shape=(100*, *400*, *20*, *1)*)

Convert legacy format MS to xarray Visibility Dataset compatible zarr format

This function requires CASA6 casatools module.

Parameters

- **infile** (*str*) – Input MS filename
- **outfile** (*str*) – Output zarr filename. If None, will use infile name with .vis.zarr extension
- **ddi** (*int*) – Specific ddi to convert. Leave as None to convert entire MS
- **compressor** (*numcodecs.blosc.Blosc*) – The blosc compressor to use when saving the converted data to disk using zarr. If None the zstd compression algorithm used with compression level 2.
- **chunk_shape** (*4-D tuple of ints*) – Shape of desired chunking in the form of (time, baseline, channel, polarization). Default is (100, 400, 20, 1) Note: chunk size is the product of the four numbers, and data is batch processed by time axis, so that will drive memory needed for conversion.

ms_to_zarr_numba

ms_to_zarr_numba(*infile*, *outfile=None*, *ddi=None*, *compressor=None*)

Convert legacy format MS to xarray Visibility Dataset compatible zarr format

This function requires CASA6 casatools module.

Parameters

- **infile** (*str*) – Input MS filename
- **outfile** (*str*) – Output zarr filename. If None, will use infile name with .vis.zarr extension
- **ddi** (*int*) – Specific ddi to convert. Leave as None to convert entire MS
- **compressor** (*blosc*) – The blosc compressor to use when saving the converted data to disk using zarr. If None the zstd compression algorithm used with compression level 2.

read_legacy_ms

read_legacy_ms(*infile*, *ddi=None*)

Todo: This function is not yet implemented

Read legacy CASA MS format data directly to an xarray Visibility Dataset

Parameters

- **infile** (*str*) – Input MS filename
- **ddi** (*int*) – Data Description ID in MS to read. If None, defaults to 0

Returns New Visibility Dataset of MS contents

Return type xarray Dataset

zarr_to_asdm

zarr_to_asdm(*infile*, *outfile=None*)

Todo: This function is not yet implemented

Convert xarray Visibility Dataset from zarr format to ASDM format (future)

Parameters

- **infile** (*str*) – Input zarr filename
- **outfile** (*str*) – Output ASDM filename. If None, will use infile name with .asdm extension

Returns Success status

Return type bool

zarr_to_image

zarr_to_image(*infile*, *outfile=None*)

Todo: This function is not yet implemented

Convert xarray Image Dataset compatible zarr format to legacy CASA Image format or FITS format

Parameters

- **infile** (*str*) – Input zarr image filename
- **outfile** (*str*) – Output image filename. If None, will use infile name with .image extension

zarr_to_ms

zarr_to_ms(*infile*, *format='ms'*, *outfile=None*)

Todo: This function is not yet implemented

Convert xarray Visibility Dataset from zarr format to Legacy CASA MS or FITS format

Parameters

- **infile** (*str*) – Input zarr filename
- **format** (*str*) – Conversion output format, 'ms' or 'fits'. Default = 'ms'
- **outfile** (*str*) – Output MS filename. If None, will use infile name with .ms extension

Returns Success status

Return type bool

1.12.3 Data Input / Output

Most CNGI functions operate on xarray Datasets while the data is stored on disk in Zarr format. These functions allow the transition back and forth between the two.

To access these functions, use your favorite variation of: `import cngi.dio`

describe_vis	Summarize the contents of a zarr format Visibility directory on disk
read_image	Read xarray zarr format image from disk
read_vis	Read zarr format Visibility data from disk to xarray Dataset
write_image	Write image dataset to xarray zarr format on disk
write_vis	Write xarray Visibility Dataset to zarr format on disk

describe_vis

describe_vis (*infile*)

Summarize the contents of a zarr format Visibility directory on disk

Parameters **infile** (*str*) – input filename of zarr Visibility data

Returns Summary information

Return type pandas.core.frame.DataFrame

read_image

read_image (*infile*)

Read xarray zarr format image from disk

Parameters **infile** (*str*) – input zarr image filename

Returns New xarray Dataset of image contents

Return type xarray.core.dataset.Dataset

read_vis

read_vis (*infile*, *ddi=0*)

Read zarr format Visibility data from disk to xarray Dataset

Parameters

- **infile** (*str*) – input Visibility filename
- **ddi** (*int*) – Data Description ID of Visibility data to read. Defaults to 0

Returns New xarray Dataset of Visibility data contents

Return type xarray.core.dataset.Dataset

write_image

write_image(*xds*, *outfile='image.zarr'*)
Write image dataset to xarray zarr format on disk

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – image Dataset to write to disk
- **outfile** (*str*) – output filename, generally ends in .zarr

write_vis

write_vis(*xds*, *outfile='vis.zarr'*, *ddi=0*, *append=True*)
Write xarray Visibility Dataset to zarr format on disk

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – Visibility Dataset to write to disk
- **outfile** (*str*) – output filename, generally ends in .zarr
- **int** (*ddi*) – Data Description ID of Visibility data to write. Defaults to 0
- **append** (*bool*) – Append this DDI in to an existing zarr directory. False will erase old zarr directory. Default=True

1.12.4 Visibilities

These functions examine or manipulate Visibility data in the xarray Dataset (xds) format. They take an xds as input and return a new xds or some other structure as output. Some may operate directly on the zarr data store on disk.

The input xarray Dataset is never modified.

To access these functions, use your favorite variation of: `import cngi.vis`

applyflags	Apply flag variables to other data in Visibility Dataset
chanaverage	Average data across channels
hanningsmooth	
joinspw	
recalculateuvw	
regridspw	
timeaverage	Average data across time axis
uvcontsub	
uvmodelfit	
visualize	Plot a preview of any xarray DataArray contents

applyflags

applyflags(*xds*, *flags=None*)
Apply flag variables to other data in Visibility Dataset

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **flags** (*list of str*) – list of data var names to use as flags. Default None uses all bool types

Returns output Visibility Dataset

Return type xarray.core.dataset.Dataset

chanaverage

chanaverage(*xds*, *width=1*)

Average data across channels

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **width** (*int*) – number of adjacent channels to average. Default=1 (no change)

Returns New Visibility Dataset

Return type xarray.core.dataset.Dataset

hanningsmooth

hanningsmooth(*xds*, *field=None*, *spw=None*, *timerange=None*, *uvrange=None*, *antenna=None*, *scan=None*)

Todo: This function is not yet implemented

Perform a running mean across the spectral axis with a triangle as a smoothing kernel

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **field** (*int*) – field selection. If None, use all fields
- **spw** (*int*) – spw selection. If None, use all spws
- **timerange** (*int*) – time selection. If None, use all times
- **uvrange** (*int*) – uvrange selection. If None, use all uvranges
- **antenna** (*int*) – antenna selection. If None, use all antennas
- **scan** (*int*) – scan selection. If None, use all scans

Returns New Visibility Dataset with updated data

Return type xarray.core.dataset.Dataset

joinspw

joinspw(*xds1*, *xds2*)

Todo: This function is not yet implemented

Concatenate together two Visibility Datasets of compatible shape

Parameters

- **xds1** (*xarray.core.dataset.Dataset*) – first Visibility Dataset to join
- **xds2** (*xarray.core.dataset.Dataset*) – second Visibility Dataset to join

Returns New Visibility Dataset with combined contents

Return type xarray.core.dataset.Dataset

recalculateuvw

recalculateuvw(*xds*, *field=None*, *refcode=None*, *reuse=True*, *phasecenter=None*)

Todo: This function is not yet implemented

Recalulate UVW and shift data to new phase center

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **field** (*int*) – fields to operate on. None = all
- **refcode** (*str*) – reference frame to convert UVW coordinates to
- **reuse** (*bool*) – base UVW calculation on the old values
- **phasecenter** (*float*) – direction of new phase center. None = no change

Returns New Visibility Dataset with updated data

Return type xarray.core.dataset.Dataset

regridspw

regridspw(*xds*, *field=None*, *spw=None*, *timerange=None*, *uvrange=None*, *antenna=None*, *scan=None*, *mode='channel'*, *nchan=None*, *start=0*, *width=1*, *interpolation='linear'*, *phasecenter=None*, *restfreq=None*, *outframe=None*, *veltype='radio'*)

Todo: This function is not yet implemented

Transform channel labels and visibilities to a spectral reference frame which is appropriate for analysis, e.g. from TOPO to LSRK or to correct for doppler shifts throughout the time of observation

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **field** (*int*) – field selection. If None, use all fields
- **spw** (*int*) – spw selection. If None, use all spws
- **timerange** (*int*) – time selection. If None, use all times
- **uvrange** (*int*) – uvrange selection. If None, use all uvranges
- **antenna** (*int*) – antenna selection. If None, use all antennas
- **scan** (*int*) – scan selection. If None, use all scans
- **mode** (*str*) – regridding mode
- **nchan** (*int*) – number of channels in output spw. None=all
- **start** (*int*) – first input channel to use
- **width** (*int*) – number of input channels to average
- **interpolation** (*str*) – spectral interpolation method
- **phasecenter** (*int*) – image phase center position or field index
- **restfreq** (*float*) – rest frequency
- **outframe** (*str*) – output frame, None=keep input frame
- **veltype** (*str*) – velocity definition

Returns New Visibility Dataset with updated data

Return type xarray.core.dataset.Dataset

timeaverage

timeaverage (*xds*, *width=1*, *timebin=None*, *timespan='both'*)

Average data across time axis

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **width** (*int*) – number of adjacent times to average (fast), used when timebin is None. Default=1 (no change)
- **(future)** (*timespan*) – time bin width to averaging (in seconds) (slow - requires interpolation). Default None uses width parameter
- **(future)** – Span of the timebin. Allowed values are 'scan', 'state' or 'both'

Returns New Visibility Dataset

Return type xarray.core.dataset.Dataset

uvcontsub

uvcontsub (*xds*, *field=None*, *fitspw=None*, *combine=None*, *solint='int'*, *fitorder=0*)

Todo: This function is not yet implemented

Estimate continuum emission and subtract it from visibilities

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **field** (*int*) – field selection. If None, use all fields
- **fitspw** (*int*) – spw:channel selection for fitting
- **combine** (*str*) – data axis to combine for the continuum
- **solint** (*str*) – continuum fit timescale
- **fitorder** (*int*) – polynomial order for the fits

Returns New Visibility Dataset with updated data

Return type xarray.core.dataset.Dataset

uvmodelfit

uvmodelfit(*xds, field=None, spw=None, timerange=None, uvrange=None, antenna=None, scan=None, niter=5, comptype='p', sourcepar=[1, 0, 0], varypar=[]*)

Todo: This function is not yet implemented

Fit simple analytic source component models directly to visibility data

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **field** (*int*) – field selection. If None, use all fields
- **spw** (*int*) – spw selection. If None, use all spws
- **timerange** (*int*) – time selection. If None, use all times
- **uvrange** (*int*) – uvrange selection. If None, use all uvranges
- **antenna** (*int*) – antenna selection. If None, use all antennas
- **scan** (*int*) – scan selection. If None, use all scans
- **niter** (*int*) – number of fitting iteractions to execute
- **comptype** (*str*) – component type (p=point source, g=ell. gauss. d=ell. disk)
- **sourcepar** (*list*) – starting fuess (flux, xoff, yoff, bmajaxrat, bpa)
- **varypar** (*list*) – parameters that may vary in the fit

Returns New Visibility Dataset with updated data

Return type xarray.core.dataset.Dataset

visualize

visualize(*xda*, *axis=None*, *tsize=250*)

Plot a preview of any xarray DataArray contents

Parameters

- **xda** (*xarray.core.dataarray.DataArray*) – input DataArray
- **axis** (*str or list*) – DataArray coordinate(s) to plot against data. Default None uses range
- **tsize** (*int*) – target size of the preview plot (might be smaller). Default is 250 points per axis

Returns

Return type Open matplotlib window

1.12.5 Image

These functions examine or manipulate Image data in the xarray Dataset (xds) format. They take an xds as input and return a new xds or some other structure as output. Some may operate directly on the zarr data store on disk.

The input xarray Dataset is never modified.

To access these functions, use your favorite variation of: `import cngi.image`

contsub	
ellipsefit	
mask	Create a new mask Data variable in the Dataset
moment	Collapse an n-dimensional image cube into a moment by taking a linear combination of individual planes
preview	Preview the selected image component
rebin	Rebin an n-dimensional image across any single (spatial or spectral) axis
reframe	
region	Create a new region Data variable in the Dataset
regrid	
rmfit	
smooth	Smooth data along n-dimensions of the image cube
specfit	
specflux	
spxfit	

contsub

contsub(*ds*)

Todo: This function is not yet implemented

Continuum subtraction of an image cube

Perform a polynomial baseline fit to the specified channels from an image and subtract it from all channels

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

ellipsefit

ellipsefit(*ds*)

Todo: This function is not yet implemented

Fit one or more elliptical gaussian components on an image region

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

mask

mask(*xds*, *name='mask1'*, *ra=None*, *dec=None*, *pixels=None*, *pol=-1*, *channels=-1*)
Create a new mask Data variable in the Dataset

Note: This function currently only supports rectangles and integer pixel boundaries

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Image
- **name** (*str*) – dataset variable name for mask, overwrites if already present
- **ra** (*list*) – right ascension coordinate range in the form of [min, max]. Default None means all
- **dec** (*list*) – declination coordinate range in the form of [min, max]. Default None means all
- **pixels** (*numpy.ndarray*) – array of shape (N,2) containing pixel box. AND'd with ra/dec
- **pol** (*int or list*) – polarization dimension(s) to include in mask. Default of -1 means all
- **channels** (*int or list*) – channel dimension(s) to include in mask. Default of -1 means all

Returns output Image

Return type xarray.core.dataset.Dataset

moment

moment(*ds*, ***kwargs*)
Collapse an n-dimensional image cube into a moment by taking a linear combination of individual planes

Note: This implementation still needs to implement additional moment codes, and verify behavior of implemented moment codes.

Parameters

- **ds** (*xarray.core.dataset.Dataset*) – input Image
- **axis** (*str, optional*) – specified axis along which to reduce for moment generation, Default='frequency'
- **code** (*int, optional*) – number that selects which moment to calculate from the following list

  -1 - mean value of the spectrum (default)

  0 - integrated value of the spectrum

  1 - intensity weighted coordinate; traditionally used to get 'velocity fields'

  2 - intensity weighted dispersion of the coordinate; traditionally used to get 'velocity dispersion'

  3 - median of I

  4 - median coordinate

  5 - standard deviation about the mean of the spectrum

  6 - root mean square of the spectrum

  7 - absolute mean deviation of the spectrum

  8 - maximum value of the spectrum

  9 - coordinate of the maximum value of the spectrum

  10 - minimum value of the spectrum

  11 - coordinate of the minimum value of the spectrum
- ****kwargs** – Arbitrary keyword arguments

Returns output Image

Return type xarray.core.dataset.Dataset

preview

preview(*xds*, *variable='image'*, *region=None*, *pol=0*, *channels=0*, *tsize=250*)

Preview the selected image component

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input Image
- **variable** (*str*) – dataset variable to plot. Default is image
- **region** (*str*) – dataset variable to use as a region/mask.
- **pol** (*int*) – polarization dimension index to plot. Default is 0
- **channels** (*int or list*) – channel dimension index or indices to plot. Default is 0
- **tsize** (*int*) – target size of the preview image (might be smaller). Default is 250 pixels

Returns

Return type Open matplotlib window

rebin

rebin(*ds*, ***kwargs*)
Rebin an n-dimensional image across any single (spatial or spectral) axis

---

Todo: Check for and apply all masks and regions

Accept arguments that control which DataArray is rebinned

Improve performance when framework client has processes=True

Note: The new Dataset generated by the current implementation of this function will lose its metadata attributes. See https://github.com/pydata/xarray/issues/3376

Parameters

- **ds** (*xarray.core.dataset.Dataset*) – input Image
- **factor** (*int, optional*) – scaling factor for binning, Default=1
- **axis** (*str, optional*) – dataset dimension upon which to rebin, Default='frequency'
- ****kwargs** – Arbitrary keyword arguments

Returns output Image

Return type xarray.core.dataset.Dataset

reframe

reframe(*ds*)

Todo: This function is not yet implemented

Change the velocity system of an image

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

region

region(*xds*, *name='region1'*, *ra=None*, *dec=None*, *pixels=None*, *pol=-1*, *channels=-1*)
Create a new region Data variable in the Dataset

Note: This function currently only supports rectangles and integer pixel boundaries

---

Parameters

- **xds** (*xarray.core.dataset.Dataset*) – input image dataset

- **name** (*str*) – dataset variable name for region, overwrites if already present
- **ra** (*list*) – right ascension coordinate range in the form of [min, max]. Default None means all
- **dec** (*list*) – declination coordinate range in the form of [min, max]. Default None means all
- **pixels** (*array_like*) – array of shape (N,2) containing pixel box. OR'd with ra/dec
- **pol** (*int or list*) – polarization dimension(s) to include in region. Default of -1 means all
- **channels** (*int or list*) – channel dimension(s) to include in region. Default of -1 means all

Returns New Dataset

Return type xarray.core.dataset.Dataset

regrid

regrid(*ds*)

Todo: This function is not yet implemented

Regrid one image on to the coordinate system of another

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

rmfit

rmfit(*ds*)

Todo: This function is not yet implemented

Generate the rotation measure by performing a least square fit with Stokes Q and U axes

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

smooth

smooth(*ds*)
Smooth data along n-dimensions of the image cube

Todo: Verify

Handle masking

Handle polarization and channel selection

Support more smoothing kernels

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

specfit

specfit(*ds*)

Todo: This function is not yet implemented

Perform polynomial, gaussian and lorentzian spectral line fits in the image cube

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

specflux

specflux(*ds*)

Todo: This function is not yet implemented

Calculate the flux as a function of frequency and velocity over the selected region

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

spxfit

spxfit(*ds*)

Todo: This function is not yet implemented

Fit a power logarithmic polynomial to pixel values along specified axis

Parameters **ds** (*xarray.core.dataset.Dataset*) – input Image

Returns output Image

Return type xarray.core.dataset.Dataset

1.12.6 Direct Access

These functions allow direct access to the underlying Dask processing engine.

To access these functions, use your favorite variation of: `import cngi.direct`

GetFrameworkClient	Return the CNGI framework scheduler client
InitializeFramework	Initialize the CNGI framework

GetFrameworkClient

GetFrameworkClient()
Return the CNGI framework scheduler client

Returns Client from Dask Distributed for use by Dask objects

Return type distributed.client.Client

InitializeFramework

InitializeFramework(*workers=2*, *memory='8GB'*, *processes=True*, ***kwargs*)
Initialize the CNGI framework

This sets up the processing environment such that all future calls to Dask Dataframes, arrays, etc will automatically use the scheduler that is configured here.

Parameters

- **workers** (*int*) – Number of processor cores to use, Default=2
- **memory** (*str*) – Max memory allocated to each worker in string format. Default='8GB'
- **processes** (*bool*) – Whether to use processes (True) or threads (False), Default=True
- **threads_per_worker** (*int*) – Only used if processes = True. Number of threads per python worker process, Default=1

Returns Client from Dask Distributed for use by Dask objects

Return type distributed.client.Client

1.12.7 Gridding

Example notional Non-Uniform FFT Gridding function(s) provided here to assess performance of infrastructure package.

To access these functions, use your favorite variation of: `import cngi.gridding`

calc_image_cell_size	Calculates an appropriate number of pixels and and cell size for imaging a measurement set.
create_prolate_spheroidal_kernel	Create PSWF to serve as gridding kernel
create_prolate_spheroidal_kernel_1D	

Continued on next page

Table 6 – continued from previous page

dirty_image	Grids visibilities from Visibility Dataset and returns dirty Image Dataset.
grid	Grids visibilities from Visibility Dataset.
prolate_spheroidal_function	Calculate PSWF using an old SDE routine re-written in Python
standard_grid	Grids visibilities.

calc_image_cell_size

calc_image_cell_size (*vis_xds*, *min_dish_diameter*)

Calculates an appropriate number of pixels and and cell size for imaging a measurement set. It uses the perfectly-illuminated circular aperture approximation to determine the field of view and 7 pixel per beam for the cell size.

Parameters

- **vis_xds** (*xarray.core.dataset.Dataset*) – input Visibility Dataset
- **min_dish_diameter** (*float*) – smallest antenna diameter

Returns

- **imsize** (*list of ints*) – number of pixels for each spatial dimension.
- **cell** (*list of ints*) – Cell size is arcseconds.

create_prolate_spheroidal_kernel

create_prolate_spheroidal_kernel (*oversampling*, *support*, *n_uv*)

Create PSWF to serve as gridding kernel

Parameters

- **oversampling** (*int*) – oversampling//2 is the index of the zero value of the oversampling value
- **support** (*int*) – support//2 is the index of the zero value of the support values
- **n_uv** (*int array*) – (2) number of pixels in u,v space

Returns

- **kernel** (*numpy.ndarray*)
- **kernel_image** (*numpy.ndarray*)

create_prolate_spheroidal_kernel_1D

create_prolate_spheroidal_kernel_1D (*oversampling*, *support*)

dirty_image

dirty_image (*vis_dataset*, *grid_parms*)

Grids visibilities from Visibility Dataset and returns dirty Image Dataset. If to_disk is set to true the data is saved to disk. :param vis_xds: input Visibility Dataset :type vis_xds: xarray.core.dataset.Dataset :param grid_parms: keys ('chan_mode','imsize','cell','oversampling','support','to_disk','outfile') :type grid_parms: dictionary

Returns **dirty_image_xds**

Return type xarray.core.dataset.Dataset

grid

grid(*vis_dataset*, *grid_parms*)

Grids visibilities from Visibility Dataset. If to_disk is set to true the data is saved to disk. :param vis_xds: input Visibility Dataset :type vis_xds: xarray.core.dataset.Dataset :param grid_parms: keys ('chan_mode','imsize','cell','oversampling','support','to_disk','outfile') :type grid_parms: dictionary

Returns **grid_xds**

Return type xarray.core.dataset.Dataset

prolate_spheroidal_function

prolate_spheroidal_function(*u*)

Calculate PSWF using an old SDE routine re-written in Python

Find Spheroidal function with M = 6, alpha = 1 using the rational approximations discussed by Fred Schwab in 'Indirect Imaging'.

This routine was checked against Fred's SPHFN routine, and agreed to about the 7th significant digit.

The griddata function is (1-NU**2)*GRDSF(NU) where NU is the distance to the edge. The grid correction function is just 1/GRDSF(NU) where NU is now the distance to the edge of the image.

standard_grid

standard_grid(*grid_data*, *uvw*, *weight*, *flag_row*, *flag*, *freq_chan*, *chan_map*, *n_chan*, *pol_map*, *cgk_1D*, *grid_parms*)

Grids visibilities.

Parameters

- **grid**(*complex array (n_chan, n_pol, n_u, n_v)*)–
- **sum_weight**(*float array (n_chan, n_pol)*)–
- **uvw**(*float array (n_time, n_baseline, 3)*)–
- **freq_chan**(*float array (n_chan)*)–
- **chan_map**(*int array (n_chan)*)–
- **pol_map**(*int array (n_pol)*)–
- **weight**(*float array (n_time, n_baseline, n_vis_chan)*)–
- **flag_row**(*boolean array (n_time, n_baseline)*)–
- **flag**(*boolean array (n_time, n_baseline, n_chan, n_pol)*)–
- **cgk_1D**(*float array (oversampling*(support//2 + 1))*)–
- **grid_parms** (*dictionary ('n_imag_chan','n_imag_pol','n_uv', 'delta_lm','oversampling','support')*)–

Returns

- **grid** (*complex array (n_imag_chan,n_imag_pol,n_u,n_v)*)
- **sum_weight** (*float array (n_imag_chan,n_imag_pol)*)

INDEX

Z