ISM3D Documentation

Release 0.3.dev1

Rui Xue

Aug 28, 2020

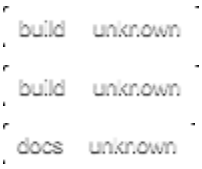
CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 ISM3D: Interferometric Source Modeling up to 3D



A Python package for simulating and modeling astronomical sources from radio interferometric observations

- Free software: BSD license
- Documentation: https://www.magclouds.org/ism3d
- Repo: https://github.com/r-xue/ism3d
- PyPI: https://pypi.org/project/ism3d

1.1.1 Features

Efficient forward-modeling of Galaxy Emission in Astronomical Data

- Construct high-precision spatially/spectroscopically-resolved galaxy emission models, from analytical or physical prescriptions of galaxy geometry, emissivity, kinematics, and dynamics.
- Implement various model elements (e.g., line, continuum, sky background) to provide a realistic presentation of expected galaxy sky emission. multi-frequency synthetic visibility or
- The galaxy model can incorporate with multiple line/continuum components.
- Perform simulated observations of galaxy emission model and render them within a wide range of astronomical data forms (e.g. radio interferometer visibility, radio single-dish or optical IFU spectral cube, multiple-band photometric images, 1D spectra, etc.)

Flexible Model Fitting/Optimization Interface

- We offer several model fitting/optimization algorithms under the same interface for a flexible user-friendly modeling experience through either a command-line approach or a serial of Python API functions).
- All modeling details are summarized in a single parameter file with a very flexible/readable syntax, which is easy to re-use for progressive modeling iterations.

- The package is specially optimized for efficiently performing joint model fitting on large heterogeneous multiple-wavelength datasets and performance Bayesian-based model sampling for robust statistical error estimation with proper prior assumption. This is done by taking the advantages of multi-threading computation, with careful memory footprint management.

ISM3D is originally designed to be a special tool to extract galaxy morphology and kinematics property from large modern interferometer datasets (specifically VLA and ALMA). We integrate some convenient utility-type functions (based on CASA 6's Python modules) for helping process, image, and visualize calibrated viability data from the interferometer data archive, as we realize the importance of properly preparing visibility data, which may require additional effort due to large dataset size and "complex" nature of interferometer visibility data forms.

On the other hand, although we have demonstrated that visibility-domain modeling can eliminate the non-linear imaging process for interferometer data and offers a unique advantage on the marginally-resolved moderate SNR data (e.g., notably for high-z observations), we note that ISM3D is still equivalently equipped to analyze imaging or data products under the same modeling and fitting framework.

Currently, ism3d is organized into several sub-packages

- arts: generate artificial sources up to 3d (position-position-frequency/wavelength/velocity) in the sparse array (a.k.a. cloudlet) or regular grid (e.g. spectral cube) (see examples in the yt documents)
- simuv: a simulator for radio interferometric observations
- simxy: a simulator for spectroscopic imaging observations, as well as photometric imaging and 1D spectroscopy (not fully implemented)
- modelling: a model fitting/optimizer framework
- uvhelper: help prepare and analyze "uv" data for modeling (using CASA6): A small Python library that provides a collection of utility functions helping process, analyze, and visualize interferometric visibility data.
- xyhelper: help prepare and analyze "xy" data for modeling
- utils: utility functions to support other modules
- maths: pseudo-random generator / other math-related functions
- plots: general plotting modules

While the current effort is still a work in progress: inconsistent documentation, many non-working function placeholders, etc., fast-paced changes are expected soon. Any comments and suggestions are appreciated and code/documentation contributions are very welcome.

CHAPTER

TWO

USER GUIDE

2.1 Installation

2.1.1 From Sources

The sources for ism3d can be downloaded from the Github repo.

You can either clone the public repository and install the copy of source with:

```
$ git clone git://github.com/r-xue/ism3d
$ pip install --user .
```

To do this in one step, you can also run this command in your terminal:

```
$ pip install --user --upgrade git+https://github.com/r-xue/ism3d.git
```

Dependencies

While most dependecies will be automatically checked and installed from the standard PyPI installation (specified in setup.cfg), we note that some *might* require manual installations on certain platforms (most likley on macOS):

- **required**, for Measurement Sets I/O
  - FINUFFT
  - casa6
- **required**, for galactic dynamics modeling from mass potentials
  - galpy
- **optional**, for moderate performance improvement
  - mkl_fft/mkl_random *(improve FFT speed)*
- **test-only**, for running certain benchmarking tests and tutorials
  - galario
  - python-casacore & casacore

For development, one can manually install most dependencies with:

```
$ pip install --user -r ./requirements_dev.txt
```

2.1.2 From Docker Hub

The `ism3d` repository contains a Dockerfile, that automatically builds the *rxastro/ism3d:dev* images hosted at the Docker Hub. The image contains a base Linux environment (based on Ubuntu 20.04) with the `ism3d` prototype, casa6, and other Python packages (astropy, Jupyter, etc.) already installed.

https://registry.hub.docker.com/r/rxastro/ism3d

Note: The Docker image `rxastro/ism3d:dev` is built upon the base image offered by the casa6-docker project:

https://registry.hub.docker.com/r/rxastro/casa6

The container launch instruction is the similar with that of `rxastro/casa6:latest`.

Launch with an interactive shell

To launch the container with an interactive shell on a host with Docker Desktop running, just type:

```
$ docker run -it -v ~/Workspace:/root/WorkDir rxastro/ism3d:dev bash
```

This will download the image `rxastro/ism3d:dev`, start a container instance, and login as `root` (bravely...). It will also try to mount the host directory `~/Workspace` (assuming it exists) to `/root/WorkDir` of your container. After this, you can perform code development and data analysis in `/root/WorkDir` of your container (now pointing to `~/Workspace` on the host), with the access of tools/environment (e.g. **casa6**, **astropy**, etc.) residing in the image.

In case you would like to manually update local-cached images for whatever reasons, you probably want to run this before launching the container again:

```
$ docker pull rxastro/ism3d:dev
```

Run a Jupyter server

A Jupyter server has been built in the Docker image `rxastro/ism3d:dev` (see the Dockerfile content for its customization). This creates a useful feature of `rx.astro/ism3d:dev`: you can connect your host web browser to the Jupyter server running its container instance. This gives you a portable development environment semi-isolated from your host OS that offers all Jupyter-based features (e.g. Widgets) along with many Python packages: **casatools**, **casatasks**, **astropy**, **numpy**, **matplotlib**, and more.

To log in a `rxastro/ism3d:dev` container and start the Jupyter session,

```
user@host      $ docker run -v ~/Workspace:/root/WorkDir --env PORT=8890 -it -p 8890:
↪8890 rxastro/ism3d:dev bash
root@container $ jupyter-lab # start a Jupyter session
```

Then you can move back to the host, open a web browser, and connect it to the Jupyter server running on the guest OS:

```
user@host      $ firefox --new-window ${address:8890-with-token}
```

Launch with Singularity

Docker images can be imported to Singularity for HPC-based deployment:

```
$ singularity pull docker://rxastro/ism3d:dev
$ file casa6_latest.sif
$ singularity inspect casa6_latest.sif
$ singularity exec -H $HOME/vh:/Users/Rui casa6_latest.sif /bin/bash
```

Note: There are significant design differences between Docker and Singularity, and a detailed demonstration is beyond the scope of this documentation.

2.2 Usage

2.2.1 Use as a command-line tool

`ism` provides a user-friendly console command-line interface for basic simulation or model fitting tasks/workflows. The task assignment is specified by a parameter file the INI format, as well as some CLI optional keywords. This appraoch is in line with other traditional model fitting programs (e.g. Galfit, Tirific) and doesn't require Python programming in most common user cases.

```
Ruis-Mac-mini:~ Rui$ ism3d --help
usage: ism3d [-h] [-f] [-a] [-p] [-d] [-t] [-l LOGFILE] inpfile

The ISM3d CL entry point:
    ism3d path/example.inp

    model fitting:
        ism3d -f path/example.inp
    analyze fitting results (saved in FITS tables / HDFs?) and export model/data for␣
↪diagnostic plotting
        ism3d -a path/example.inp
    generate diagnostic plots
        ism3d -p path/example.inp

Note:
    for more complicated / customized user cases, one should build a workflow by
    calling modules/functions directly (e.g. hz_examples.py)



positional arguments:
inpfile               A parameter input file

optional arguments:
-h, --help            show this help message and exit
-f, --fit             perform parameter optimization
-a, --analyze         analyze the fitting results / exporting data+model
-p, --plot            generate diagnotisc plots
-d, --debug           Debug mode; prints extra statements
-t, --test            test mode; run benchmarking scripts
-l LOGFILE, --logfile LOGFILE
                        path to log file
```

2.2.2 Use Python APIs and utility functions

`ism3d` itself is written purely in Python, although most of its depedencies are not. You can use `ism3d` as a general utility library for your own program and build your modeling / data analysis workflow.

For example, to use the `invert` function in `ism3d`, try:

```
In [1]: from ism3d.uvhelper import invert

In [2]: help(invert)

Help on function invert in module ism3d.uvhelper.imager:

invert(vis='', imagename='', datacolumn='data', antenna='', weighting='briggs',␣
↪robust=1.0, npixels=0, cell=0.04, imsize=[128, 128], phasecenter='', specmode='cube
↪', start='', width='', nchan=-1, perchanweightdensity=True, restoringbeam='',␣
↪onlydm=False, pbmask=0, pblimit=0, exclude_list=['residual', 'residual.tt0',
↪'residual.tt1', 'sumwt', 'sumwt.tt0', 'sumwt.tt1', 'sumwt.tt2', 'model', 'model.tt0
↪', 'model.tt1'], **kwargs)
    Generate a compact dirty image from a MS dataset as a quick imaging snapshot;

    Note about the default setting:

    +   restoringbeam='' to preserve the original dirty beam shape:
        if "common" then additional undesired convolution will happen
    +   Another faster way to do this would be using the toolkits:
        imager.open('3C273XC1.MS')
        imager.defineimage(nx=256, ny=256, cellx='0.7arcsec', celly='0.7arcsec')
        imager.image(type='corrected', image='3C273XC1.dirty')
        imager.close()
        But it may be difficult to write a function to cover all setting already in␣
↪casatasks.tclean()

    note:
        apperantly tclean(start='',width='',nchan-1) will not follow the actual spw-
↪channel arrangeent:
        it will sort channel by frequency forst and then start the sequence.
        That means even when negative channel width will still get frequency-
↪increasing cube:
            for such case, width=-1,nchanel=240,start=239 will get a cube following␣
↪the channel-frequency arrangment.
```

example Barnett et al. (2018)

2.3 Credits

2.3.1 Author

- Rui Xue <rx.astro@gmail.com>

2.3.2 Contributors

None yet. Why not be the first?

2.4 History

2.4.1 0.3.dev1 (2020-06-20)

- rename the project to "ism3d"
- merge uvrx into ism3d, code refactoring
- implement an imager based on nufft
- update docs/

2.4.2 0.2.dev2 (2020-03-26)

- use nufft to replace galario
- use casa6 instead of py-casacore

2.4.3 0.1.dev1 (2019-08-07)

- First developmental version

CHAPTER

THREE

TECHNICAL SPECIFICATION/NOTES

3.1 Parameter File Format

3.1.1 Basics

The syntax of **ism3d** input parameter files (.inp or .ini) strictly follows the INI format, which can be easily parsed by the built-in Python module configparser. The choice was made after a comparison among several readable plain-text syntaxes options (e.g., json/yaml/xml/TOML), mainly due to the simplifcity and wide support from many other languages for the INI format. It's also very readable and easy to write, much less verbose than other syntaxs, and support inline comments (starting with "#"). Foundamentally, the file content is always divided into **sections**, each of which contains keys with values.

However, INI is not designed as a data interchange format (not strongly typed), and configparser natively do not guess datatypes of values and always storing them internally as string, INI files are also typically limited to two levels and doesn't support arbitary nesting. To support more advantages features need for our purpose, we build an under-layer interpreter to further convert the string-based input values into datatype-sensitive nested dictionary using ASTEVAL. This nested dictionary not only specifiy individual source models, but also contain the metadata of observed data and control model rendering and fitting workflows. Here, we describe some useful features built-in the parameter file intepreter and how to use sections to manage modeling workflows.

3.1.2 Interpreter

DataType/Expression Support

All values associated with individual keys will be evaluated by ASTEVAL, which treats strings as mathematical expressions and statements (an alterantive to ast.literal_eval() or the built-in "evil" eval()). Therefore, all values specified within parameter files should be written as Python literals(e.g., string representations of Python objects). They are evaluated and mapped into desired data types (see the example below). Beside the default built-in Python datatypes, we also include some objects from astropy (e.g., `astropy.units` and `astropy.coordinates.SkyCoord`). Specifcailly, we add the following notation into the ASTEVAL symtable:

```
u           : astropy.units
Angle       : astropy.coordiante.Angle
Quantity    : astropy.units.quantity
SkyCoord    : astropy.coordinates.SkyCoord
np          : numpy
```

```
[disk1]
type          =      'disk3d'
import        =      'basics'
```

(continues on next page)

(continued from previous page)

```
z           =       4.06
xypos       =       SkyCoord(ra=0,dec=0,unit="deg")
contflux     =      (0.00001 * u.Unit('Jy'),20* u.Unit('GHz'),1.0)
pa          =       (70+90)*u.deg
inc         =       60.00*u.deg
sbProf       =      ('norm2d',40*u.kpc)
```

While ASTEVAL generally determines the value content and type, we note that the converted datatypes for some individual parameters will further passed through further parsing and converted into restricted object types. For example, `xypos` will be saved as a SkyCoord object although it can be written as a two-element tuple, a string, or a litereal expression of SkyCoord.

In addition, the built-in interpreter in `ism3d` will check parsed datatype with the expected datatype of each keyword, and provide warning when unexpected values are detected (e.g. the position angle keyword `pa` should not be a string).

The datatype-sensitive extension from traditional INI files is similar to the design built in the TOML format and its parsers. However, we decide to use a custome interpreter to not only handle basic datatype, but also support advanced datatypes and literal mathematical expression. The built-in interpreter will decide and check on value datatype expected for individual keywords (as the `xypos` example above)

```
[object1]
...
xypos       =       SkyCoord(ra=0,dec=0,unit="deg")     # option 1
xypos       =       '23h46m09.4373s +12d49m19.2479s'    # option 2
xypos       =       (189.2933333-1*40/3600,62.3711111+1*15/3600)  # option 3
...
```

If the first ASTEVAL evaluation fails, the program will:

> do split() -> re-evaluate on each element -> assemble the results into a list -> assign the list to the keyword
> This feature is only designed to work a backend solution for some special paramater input files

The data/output file paths are specified in relative to the working directory (where you lunch GMaKE, either using CLI or a Python script). To clear up any confusion, absolute paths can also be used.

Section/Keyword Reference

The keywords from one section will not inference keywords from another section with a different section name. In another word, a keyword will only reside within a section (or local)

Section referencing is introduced to link the parameters from one section (with dedicated purposes) to one keyword of another section. For example, we can link a section describing gravitional potential to the rotation curve of a disk model (see `basics.rcProf`):

> rcProf = ('potential','pots')

Another important application of section referencing is sharing common parameters between different sections. Specifcailly, a set of keyword which can be "imported" into another input section using the syntax `import = 'section_name'` Then a group of parameters can be shared across several sections of different purpose For example, if two emission lines arised from the same thin-disk galaxy geometry but with different spatial distribution in the disk plane, we can write the parameters describing disk geometry into one paramater section and only reference them from other two sections actually specifiying line models. This will simplying model fitting and reduce redudent paramater requirements (see `co43.import` and `ci10.import`):

```
import              =   'basics'
```

Note that the place where this line is will be important as inp2mod will overwrite existing keywords by design

Keyword referencing can be used to "tie" different parameters within a section or even across differen section. In the example below, `ci10.lineflux` is set to a third of the value of `co43.lineflux` as:

```
lineflux          =  co43['lineflux']/3
```

In general, 'A[b]', "b@A", or "${A:b}" is interpreted as the value of the keyword "b" from Section "A". Value slicing is supported, as 'A[b][2]', "b[2]@A", or "${A:b}[2]" is interpreted as the third element of the "b" value from Section "A". e.g. "pa@co21disk" = the position angle of the components named "co21disk" This capability is similar to the function of configparser.ExtendedInterpolation. More advanced mathmetical expressions and keyword reference can be combiend together, for example:

```
px = 2.0*(py@objz)
px = sqrt(2*(py@objz))
```

as long as it follows the correct referening syntax and the expression can be understood by ASTEVAL.

```
[basics]
object          =    'bx610'
z               =    2.21
pa              =    -54.22660862671279 * u.Unit('deg')
inc             =    46.24786218695453 * u.Unit('deg')
xypos           =    '23h46m09.4373s +12d49m19.2479s' # or SkyCoord('23h46m09.4373s␣
↪+12d49m19.2479s',frame='icrs')
vsys            =      104 * u.Unit('km / s')
rcProf          =      ('potential','pots')
vSigma          =        50.430210605487204 * u.Unit('km / s')
vrot_rpcorr     =      True

[pots]
type            =      'potential'
import          =      'basics'
expdisk         =      (1000.2938616145047 * u.Unit('solMass / pc2'),3.5 * u.Unit('kpc
↪'))
dexpdisk        =       (10 * u.Unit('solMass / pc3'),3.5 * u.Unit('kpc'),0.5 * u.Unit(
↪'kpc'))
nm3expdisk      =       (10 * u.Unit('solMass / pc3'),3.5 * u.Unit('kpc'),0.5 * u.Unit(
↪'kpc'),True)
nfw             =      (500000000000.0 * u.Unit('solMass'),2.21)
isochrone       =      (10**10*u.Unit('solMass'),0.1*u.kpc)
kepler          =       10**10*u.Unit('solMass')

[co43]
type                =  'disk3d'
import              =  'basics'
note                =  'CO 4-3 of BX610 in BB2'
vis                 =   '../data/bx610/alma/2013.1.00059.S/bb3.ms.pt2'
restfreq            =  461.0407682 * u.Unit('GHz')
lineflux            =  1.4 * u.Unit('Jy km / s')
sbProf              =  ('sersic2d',2.0*u.kpc,1)
vbProf              =  ('sech',0.25*u.kpc)

[ci10]
type                =  'disk3d'
import              =  'basics'
note                =  'CI 1-0 of BX610 in BB2'
vis                 =  '../data/bx610/alma/2013.1.00059.S/bb1.ms.pt2'
```

(continues on next page)

(continued from previous page)

```
restfreq            =  492.160651 * u.Unit('GHz')
lineflux            =  co43['lineflux']/3
sbProf              =  ('sersic2d',3.0*u.kpc,1)
vbProf              =  ('sech',0.5*u.kpc)
```

3.1.3 Section Types

As any INI configuration file, the basic strcuture of **ism3d** parameter file compose of **sections**. Related key-value pairs are grouped into arbitarily named sections, names of which appear in square brackets (`[` and `]`).

Each section usually describes one specific aspect of parameterized models or provides controls on planned workflow tasks, i.e. Source Specifications, Modeling Workflows, MISC.

One named parameter section generally fall into one of the following categories:

Source Model (type='disk3d' or type='apmodel')

A "source-model" section describes the physical properties of a emission component (either line or continuum), with the section name as its identification. The keyword `type` can be *disk3d* or *apmodel*, which determine how the emission model is constructured and rendered.

Dynamical Model (type='potential')

A "dynamics-model" section describes the properties of a dynamical component, and its content will be used to specify kinematic model (e.g. rotational curve, veloccity dispersion) for line emission component(s). Its section name (or i.d.) is usually referenced in the keyword *rcProf* of "source-model" sections. By combing a "dynamics-model" model with the prescription of line emission spatial distribution (from a "source-model" section, `ism3d` can construct the representation of a line emission in a spectral cube dimension.

Lense Model (type='lens')

A "lens-model" section describe the properties of a lensing model betwen the observer and simulated source

Database (type='data')

A "database" section specify the metadata of actual or simulated data which will be used by `ism3d`. The content can include file paths of FITS image/spectral-cube to be models or MeasurementSet to be fitted, or any ancillary data useful for modelling (PSF images, uncertainty images/cube)

Workflow Management with Reserved Section Names

Several section names are resevered for workflow managment (especially useful for the CLI interface)

`[ism3d.optimize]`

The `ism3d.optimize` section specify how the program performs parameter optimization (i.e. model fitting)

`[ism3d.analyze]`

The `ism3d.analyze` section control how the program runs diagnostics analysis and plotting from modelling results.

`[ism3d.general]`

The `ism3d.general` section provide general configuration information for `ism3d` (e.g., specifying the location of output files or working folders)

```
################################################################################
↪####
[disk1] # specifications of disk1
################################################################################
↪####

type           =       'disk3d'
import         =       'basics'
z            =         4.06
xypos        =         (189.2933333+0*40/3600,62.3711111+1*15/3600)
contflux      =        (0.00001 * u.Unit('Jy'),20* u.Unit('GHz'),8.0)
pa           =         70.00*u.deg
inc          =         60.00*u.deg
sbProf        =        ('norm2d',40*u.kpc)

################################################################################
↪#####
[disk2] # specifications of disk2 (eqvauilent to disk1, despite different rendering␣
↪option
################################################################################
↪#####

type           =        'apmodel'
z            =         4.0548
xypos        =         (189.2933333+0*40/3600,62.3711111+0*15/3600)
contflux     =         (0.001 * u.Unit('Jy'),46 * u.Unit('GHz'),3.0)
sbProf       =         ('Gaussian2D',40*u.kpc,20*u.kpc,70*u.deg)

################################################################################
↪####
[sie1] # lense model
################################################################################
↪####

type         = 'lens'
xypos        =         '12h37m11.89s +62d22m11.8s'
lsProf       =   ('sie',10*u.arcsec,0.5,70*u.deg)
```

3.1.4 Keyword-Value

We provide a detailed table of all available keywords for different section types (see above)

CHAPTER

FOUR

DEVELOPMENT GUIDE

4.1 Background

The package was initially developed upon the algoirthm used by Xue, Fu, Isbell et al. (2018) ApJL.848.11 [[ApJL](http://iopscience.iop.org/article/10.3847/2041-8213/aad9a9)/[arXiv](http://arxiv.org/abs/1807.04291)/[ADS](http://adsabs.harvard.edu/abs/2018ApJ...864L..11X)], but quickly evolve into a brand-new package with some brand-new implementaion of some under-line algorithm, including but not limited to:

- A Python model rendering module of simulating galaxy emission with complex morphology and kinematics structure, including line emission from a rotating disk galaxy with spiral arms structurate (inspired by GIPSY/galmod, GALFIT)
- I/O of visibilities in CASA measurements dataset (via. CASA6) and imlement visibility-based model simulation/fitting capacibity.
- A high-precsion and efffiencet robust algoirthm on simulating a large visibility set from arbitary sky emission models (a lightweight CASA simulator altenrative suitabe for model fitting)
- A user-friendly plain-tex based parameter file syntax for performing model simulation/fitting in a highly flexible fashion

4.2 Design Goals

- flexible and expandable modeling algorithm implementation
- provide various parameter optimization choices and robust error estimations
- the comparison can be made in either image and visibility domain
- handle multiple images/cubes simultaneously and treat blended line/objects together
- straightforward parameter setting: the emission model was built upon parameters describing “physical” properties of each object; the emission models can be easiliy translated into the data obatined in radio (frequency) or optical (wavelength) with pre-defined system response functions
- provide data/model visulizations (xy/pv-moms) and clear diagnostic plots for parameter fitting goodness (chi^2/posterior-prob)

4.3 Acknowledgement

University of Iowa

This project is built-upon several other open-sources projects (e.g. astropy,emcee,lmfit), which are constributed by numeriousc devlopers, which are credited here.

This research was supported in part through computational resources provided by The University of Iowa, Iowa City, Iowa.

4.4 See also

GMaKE is inspired by several well-developed open-source packages, which are usually designed for specific user cases and data types, or provide some equivalent module functions/algorithms within GMaKE. We greatly appreciate their open-source knowledge sharing efforts, and list them here as important reference for the general topics here with a short description. offer a subset of GMaKE's capabilities or serve as a backbone of some specific usage cases (i.a) Some package offering some similar functions or moldeing capacibilt with what GMaKE offer, although they are mostly specified for specific data form or optimized for some user case covered by GMaKE. We list them here as reference for this general topic here:

- galmod

  Likely the first popular implementation of the tiled-ring algorithm

- Bbarolo

  Improve upon the tiled-ring algorithm from galmod, with full model fitting features (likely for a single-line/disk user case)

- KinMS

  another Python and IDL implantation of tilet-ring model method to simulate spectral-cube from a rotating disk galaxy

- TiRiFiC

  a fortran-based spectral line models with full fitting function

- GALFIT

  widely used 2D images from more sophisticated galaxy morphology functions

- galario

  a C-based package wiich can transfer model images to visibilities for UV-based fitting (see note here)

- UVMULTIFIT

  a packge for, required instatltion into a stand-alone Python package

- PDSPY
- EHT-imaging
- smili
- NFFT

  a packge for, required instatltion into a stand-alone Python package

- GalSim
- lenstronomy

4.5 Reference

Here a list of web pages related to CASA or visbility data in general. In additon, I include a list of relevant packages/memos as reference for this general topic (uv-data analysis / visualization)

4.5.1 CASA-related

1. casacore, python-casacore, casarest
2. CASA6
3. CASA Task Reference Manual and Toolkit Reference Manual
4. CASA @ AWS
5. CASA6 repositary from NRAO
6. Simulation Guides
7. CASA Memos and Fundamentals in casadoc
8. CASAcore Notes
9. CASA news
10. CASA ngi

4.5.2 See also

- Analysis Utilities
- casatools.simulator
- suncasa
- pyuvdata
- Radio Astronomy Simulation, Calibration and Imaging Library (replacing Algorithm Reference Library, ARL)
- uvplot
- casa imager documentation

4.5.3 Guildlines

- LSST DM Developer Guide — LSST DM Developer Guide Current documentation
- Astropy Documentation — Astropy
- Python Developer's Guide — Python Developer's Guide
- Sphinx Documentation

4.5.4 Documentation Good Examples

COQ docs Pyramid docs

4.5.5 Sphinx Guildlines

Kernel docs Sphinx Tutorial

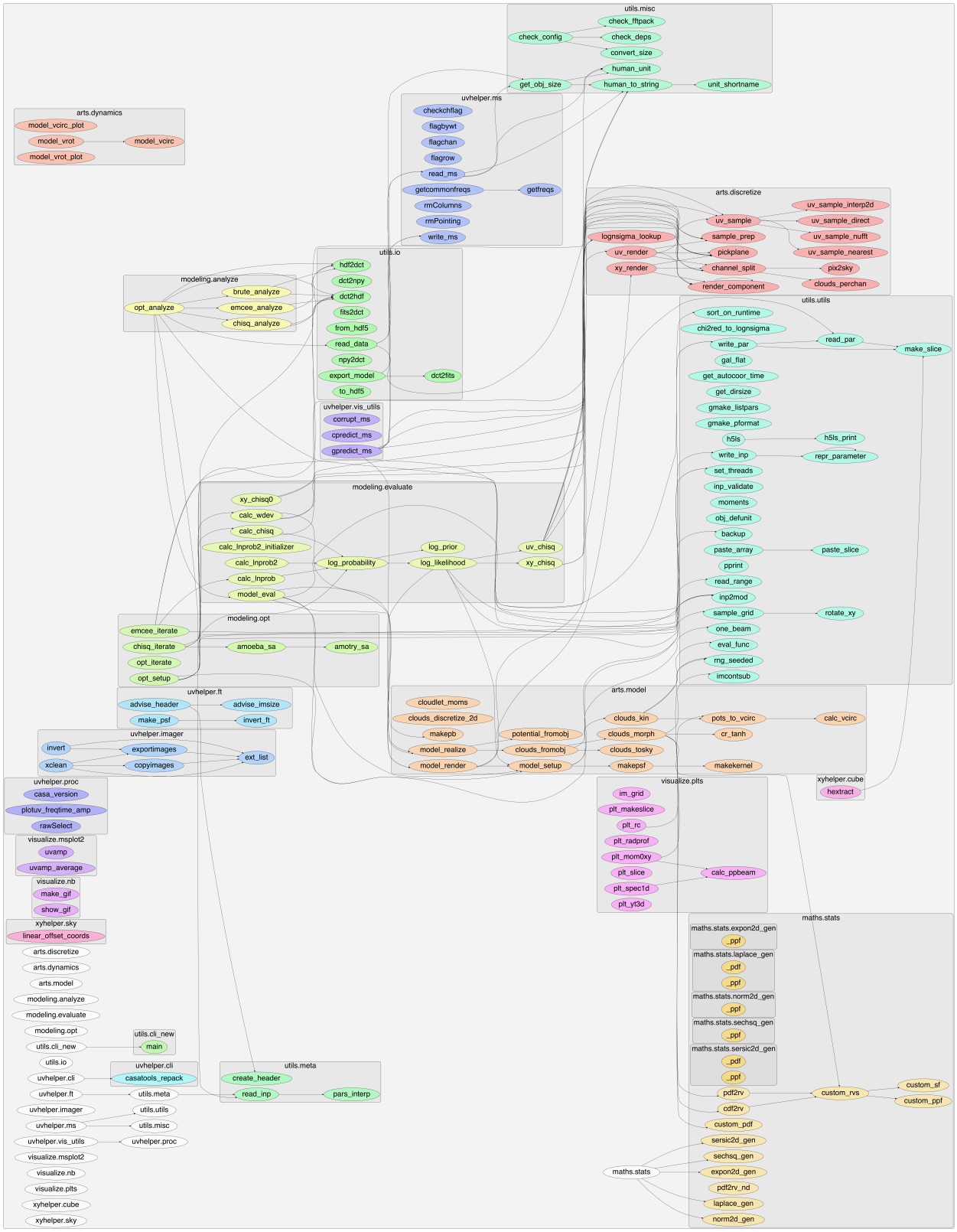
4.6 ISM3D Dictionary

Here I highlight major ism3d subpackage/modules and some complementary tools offering equivalent functions

Table 1: Title

ism3d Module/Function	description	equivalent to / inspired by
ism3d.arts.disk3d	generate cloudlet-based disk model	kinmspy, galmod, barolo3d
ism3d.arts/.simxy/.modeling	source modeling in the image domain up to 3d (position-position-frequency/wavelength, regular or sparse)	galfit (2D), TiRiFiC (3D), Barolo3d (3D)
ism3d.arts/.simuv/.modeling	source modeling in the visibility domain up to 3d (uu-vv-frequency/wavelength, sparse)	UVMULTIFIT (3D?)
ims3d.uvhelper.uv_render	render source model int a uv data frame	galario
ism3d.uvhelper.invert_ft	invert uv data into dirty map	miriad.invert, casa.tclean

4.7 ISM3D call graph



CHAPTER

FIVE

APPENDIX

5.1 ism3d.uvhelper: visibility imaging

5.1.1 Setup

We first import essential API functions / modules from `ism3d` and other libraries

Used ISM3D Functions:

- `im3d.logger.logger_config`
- `im3d.logger.logger_status`

```
nb_dir=_dh[0]
os.chdir(nb_dir+'/../output/mockup')
sys.path.append(nb_dir)
from notebook_setup import *

%matplotlib inline
%config InlineBackend.figure_format = "png" #  'png', 'retina', 'jpeg', 'svg', 'pdf'.
%reload_ext wurlitzer
%reload_ext memory_profiler
%reload_ext line_profiler

ism3d.logger_config(logfile='ism3d.log',loglevel='INFO',logfilelevel='INFO',
↪log2term=False)

print(''+ism3d.__version__)
print('working dir: {}\n'.format(os.getcwd()))
```

```
0.3.dev1
working dir: /Users/Rui/Resilio/Workspace/projects/ism3d/models/output/mockup
```

5.1.2 Data Import

We import the visibility data from CASA measurement sets into the internal `uvdata` variable (essential a simple nested dict, not class yet) and also save them into compressed HDF5 for compact storage and easy retrieve.

Here three ALMA/VLA datasets are used:

- mockup1: based on the VLA GN20 observation 1-channel
- mockup2: based on the ALMA G09 observation 1-channel
- mockup3: based on the ALMA G09 observation 240-channel, only 2mins integration
- mockup3: based on the ALMA G09 observation 240-channel, all on-source data

Used ISM3D Functions:

- `im3d.uvhelper.io.to_hdf5`
- `ism3d.uvhelper.ms.rmPointing`
- `ism3d.uvhelper.ms.read_ms`

```
os.system('rm -rf '+'mockup1_basis.ms')
mstransform(vis='../data/gn20/vla/AC974.100409.ms',outputvis='mockup1_basis.ms',
            spw='0:60',datacolumn='data')

os.system('rm -rf '+'mockup2_basis.ms')
mstransform(vis='../data/g09/alma/bb4.ms',outputvis='mockup2_basis.ms',
            spw='*:60',datacolumn='data')
rmPointing('mockup2_basis.ms',verbose=False)

os.system('rm -rf '+'mockup3_basis.ms')
mstransform(vis='../../data/g09/alma/bb4.ms',outputvis='mockup3_basis.ms',
            spw='',timerange='06:08:00~06:10:00',datacolumn='data')
rmPointing('mockup3_basis.ms',verbose=False)

os.system('rm -rf '+'mockup4_basis.ms')
os.system('ln -s ../../data/g09/alma/bb4.ms '+'mockup4_basis.ms')

for model_name in ['mockup1','mockup2','mockup3','mockup4']:

    # option 1
    #uvdata={}
    #read_ms(vis=model_name+'_basis.ms',dataset=uvdata,keyrule='basename')

    # option 2
    uvdata=read_ms(vis=model_name+'_basis.ms')

    # save to / reterive from .h5
    to_hdf5(uvdata,outname=model_name+'_basis.h5',checkname=False,compression='gzip')
```

5.1.3 Imaging

We image the visibility using two different approched implemented in `ism3d`: * `ism3d.uvhelper.invert`: a function wrapping around casa.tclean etc. to create dirty maps in an organized fahsion * `ism3d.uvhelper.invert_ft`: the same purpose as above, but based on FINUFFT

Used ISM3D Functions:

- `ism3d.uvhelper.imager.invert`
- `ism3d.uvhelper.io.from_hdf5`
- `ism3d.uvhelper.invert`
- `ism3d.uvhelper.invert_ft`
- `ism3d.uvhelper.make_psf`
- `ism3d.uvhelper.ft.advise_header`
- `ism3d.xyhelper.cube.hextract`

```
model_name='mockup4'
uvdata=from_hdf5(model_name+'_basis.h5')
```

```
<Figure size 432x288 with 0 Axes>
```

```
header=advise_header(uvdata['uvw'],
                     uvdata['phasecenter'],
                     uvdata['chanfreq'],
                     uvdata['chanwidth'],
                     antsize=12*u.m,sortbyfreq=True)

cell=header['CDELT2']<<u.deg
imsize=header['NAXIS1']

print(imsize,cell.to(u.arcsec))

tic= time.time()
invert(vis=model_name+'_basis.ms',
       imagename=model_name+'_basis.images/casa',
       weighting='natural',specmode='cubedata',width='',start='',nchan=-1, # width=-1,
↪start=239,nchan=-1,
       cell=cell.to_value(u.arcsec),imsize=[imsize,imsize],onlydm=False,
↪dropstokes=True)
toc= time.time()
print("Elapsed Time: {:>8.2f} seconds # {} \n".format(toc-tic,'ism3d.uvhelper.imager.
↪invert'))

tic= time.time()
%memit cube=invert_ft(uvdata=uvdata,header=header,sortbyfreq=True).astype(np.float32)
%memit psf=(make_psf(uvdata=uvdata,header=header,sortbyfreq=True)).astype(np.float32)
toc= time.time()
print("Elapsed Time: {:>8.2f} seconds # {} \n".format(toc-tic,'ism3d.uvhelper.ft.
↪invert_ft/.make_psf'))

fits.writeto(model_name+'_basis.images/nufft.image.fits',cube.T,header,overwrite=True)
fits.writeto(model_name+'_basis.images/nufft.psf.fits',psf.T,header,overwrite=True)

for version in ['image','psf']:
```

(continues on next page)

(continued from previous page)

```
    tcube,thdr=fits.getdata(model_name+'_basis.images/casa.'+version+'.fits',
↪header=True)
    cube,hdr=fits.getdata(model_name+'_basis.images/nufft.'+version+'.fits',
↪header=True)
    cube_diff=cube-tcube
    fits.writeto(model_name+'_basis.images/diff.'+version+'.fits',cube_diff,thdr,
↪overwrite=True)

if  model_name=='mockup4' or model_name=='mockup3':
    # get ride of the first plan of mockup4 as it's partially flagged with different␣
↪PSF.
    for version in ['nufft.image','nufft.psf','casa.image','casa.psf','casa.pb','diff.
↪image','diff.psf']:
        data,header=fits.getdata(model_name+'_basis.images/'+version+'.fits',
↪header=True)
        data_sub,header_sub=hextract(data, header, np.s_[1:,:,:])
        fits.writeto(model_name+'_basis.images/'+version+'.fits',data_sub,header_sub,
↪overwrite=True)
```

```
288 0.23221417798405314 arcsec
```

```
0%....10....20....30....40....50....60....70....80....90....100%

0%....10....20....30....40....50....60....70....80....90....100%
2020-07-06 04:48:23 WARN    task_tclean::SIImageStore::restore (file casa-source/code/
↪synthesis/ImagerObjects/SIImageStore.cc, line 2089)   Restoring with an empty model␣
↪image. Only residuals will be processed to form the output restored image.
```

```
Elapsed Time:   331.42 seconds # ism3d.uvhelper.imager.invert

peak memory: 4260.89 MiB, increment: 538.22 MiB
peak memory: 4390.84 MiB, increment: 129.93 MiB
Elapsed Time:    54.79 seconds # ism3d.uvhelper.ft.invert_ft/.make_psf
```

```
<Figure size 432x288 with 0 Axes>
```

5.1.4 Visualize

Here we demostrate the visulization capabiliy of ism3d. Specifically, we plot the results from two imaging appraoching and compare their precisions.

Used ISM3D Functions:

- `ism3d.visualize.nb.make_gif`
- `ism3d.visualize.nb.show_gif`
- `ism3d.visualize.plts.im_grid`

```
units=[] ; images=[] ; titles=[]; vmaxs=[]; vmins=[]

for version in ['casa.image','casa.psf','casa.pb','nufft.image','nufft.psf',None,
↪'diff.image','diff.psf']:
    if version is not None:
        data,hdr=fits.getdata(model_name+'_basis.images/'+version+'.fits',header=True)
```

(continues on next page)

(continued from previous page)

```
        titles.append(version)
        if 'psf' in titles[-1]:
            images.append(data); units.append("Jy/beam")
        else:
            images.append(data*1e3); units.append("mJy/beam")
        vmaxs.append(np.nanmax(images[-1]))
        vmins.append(np.nanmin(images[-1]))
    else:
        titles.append(None);  images.append(None); units.append(None); vmaxs.
↪append(None); vmins.append(None)


w = WCS(hdr).celestial
coord = SkyCoord(hdr['CRVAL1'], hdr['CRVAL2'], unit="deg")
offset_w=linear_offset_coords(w,coord)
nchan=hdr['NAXIS3']
stepchan= int(np.maximum(np.floor(int(nchan/5)),1))

fignames=[]
for ichan in range(0,nchan,stepchan):

    #clear_output(wait=True)
    figname=model_name+'_basis.images/chmap/ch{:03d}'.format(ichan)+'.pdf'
    images0=[None if image is None else image[ichan,:,:] for image in images]
    titles0=[None if title is None else title+'['+'{}'.format(ichan)+']' for title in␣
↪titles ]
    im_grid(images0,offset_w,units=units,titles=titles0,nxy=(3,3),figsize=(9,9),
↪figname=figname,vmins=vmins,vmaxs=vmaxs) ;
    fignames.append(figname)

make_gif(fignames,model_name+'_basis.images/chmap.gif')
show_gif(model_name+'_basis.images/chmap.gif')
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

REFERENCES

Barnett A., Magland J., & Klinteberg L. A parallel non-uniform fast Fourier transform library based on an "exponential of semicircle" kernel. *arXiv e-prints*, arXiv:1808.06736 (2018).