uvmcmcfit Documentation

Release 0.1

Shane Bussmann

August 10, 2015

Contents

`uvmcmcfit` is Python software designed to allow users to fit parametric models to interferometric data. It is ideally suited to extract the maximum amount of information from marginally resolved observations with interferometers like the Atacama Large Millimeter Array (ALMA), Submillimeter Array (SMA), and Plateau de Bure Interferometer (PdBI).

`uvmcmcfit` has been thoroughly tested on continuum imaging from ALMA (Bussmann et al., in prep.), SMA (Bussmann et al. 2012, 2013), and PdBI (Miettenen et al., in prep.). It has also been tested on spectral line imaging from PdBI (Oteo et al. in prep.) and SMA (Bradli et al., in prep.).

CHAPTER 1

Why Use `uvmcmcfit`?

Here are three reasons why you might be interested in using `uvmcmcfit`.

1.1 Markov Chain Monte Carlo

`uvmcmcfit` uses D. Foreman-Mackey's emcee to do Markov Chain Monte Carlo (MCMC). emcee is an affine-invariant MCMC ensemble sampler that is efficient at identifying the best-fit model and the uncertainties on the parameters of the model.

1.2 Interferometry

Interferometers measure visibilities, not images. In many situations (e.g., strong gravitational lensing), it is advantageous to measure the goodness of fit from the visibilities rather than the deconvolved images.

`uvmcmcfit` includes a pure-Python adaptation of Miriad's `uvmodel` task that allows it to generate simulated visibilities given observed visibilities and a model image.

1.3 Gravitational Lensing

Many of the brightest galaxies that have been observed by interferometers are gravitationally lensed by an intervening galaxy or group of galaxies along the line of sight. Understanding the intrinsic properties of the lensed galaxies requires a lens model.

`uvmcmcfit` includes a simple ray-tracing routine that allows it to account for both strongly lensed systems (where multiple images of the lensed galaxy are detected) and weakly lensed systems (where only a single image of the lensed galaxy is detected).

CHAPTER 2

Installation

1. Install Python. I recommend the Anaconda python distribution.
2. Install pyyaml (pip install pyyaml)
3. Install emcee (pip install emcee)
4. Install `uvmcmcfit`
    - Download `uvmcmcfit` from github: https://github.com/sbussmann/uvmcmcfit
    - Add the directory containing `uvmcmcfit` to your PYTHONPATH

CHAPTER 3

Input Data Format

`uvmcmcfit` uses visibilities obtained from an interferometer as the primary input data. The format must be either *uvfits* (preferred) or CASA *ms*.

3.1 Which format should I use?

Use the CASA *ms* format if you are only willing to run `uvmcmcfit` inside the CASA shell. Note that doing this will mean you can't take advantage of the parallel processing features included in `uvmcmcfit`. For this reason, I suggest using *uvfits* as the input file format for the visibilities.

3.2 Do I really need to worry about the weights?!?

The weights in the visibility dataset are a critical component of `uvmcmcfit`. So yes, you do need to worry about them. `uvmcmcfit` uses the weights to determine how much to trust each visibility measurement. In this way, the relative value of the weights is important (similar to a deconvolution task like clean).

Not only that, but the amplitude of the weights matters as well. Consider two extreme scenarios: one in which the weights are very very tiny, and one in which they are very very large. In the former scenario, good and models become indistinguishable because the goodness-of-fit measurement is normalized by the weights, so the difference between a good model and a bad model becomes very very small. On the other hand, if the weights are too large, then two models which are both plausible fits might still show a significant difference in the goodness of fit measurement. The right value for the amplitude of the weights is somewhere in between.

From my personal experience exploring both ALMA and SMA continuum data, scaling the weights based on the scatter in the actual visibility datasets does a good job of setting the amplitude of the weights.

Note: In future versions of CASA, the plan (as I understand it) is to set both the relative and absolute value of the weights rigorously. For now, only the relative value of the weights is meaningful, so the following advice might be useful to you.

A CASA routine exists to re-compute the weights based on the scatter in the visibilities within a spectral window (spw). This task is called `statwt`. I found that this routine worked well within CASA, but when I exported the results to a uvfits file (using `exportuvfits`), the normalization of the weights was off. So I wrote my own version of CASA's `statwt`: `uvutil.statwt()`. Using it properly is more convoluted than I would like at the moment, but it works.

First, you need to export a uvfits file that contains visibility data as function of frequency. Let's call this file 'default-weights_frequency.uvfits'. You then call `uvutil.statwt()` as in the following example:

```
import uvutil
uvutil.statwt('defaultweights_frequency.uvfits', 'statwtweights_frequency.uvfits')
```

where 'defaultweights_frequency.uvfits' is the visibility dataset with the default weights and 'statwtweights_frequency.uvfits' is a new file that is created by `uvutil.statwt()` that has the weights recomputed according to the scatter in the visibilities within a given spectral window.

Note: You can use the `ExcludeChannels` option to exclude certain channels from being considered when computing the scatter in the visibilities (e.g. so that a strong line does not bias the rms measurement too high). Do this by defining a list that contains pairs of start and end channels. All channels between (and including) the start and end channels will be excluded when computing the rms scatter in the visibilities. For example, uvutil.statwt('defaultweights_frequency.uvfits', 'statwtweights_frequency.uvfits', ExcludeChannels=[45,145]) will ensure that channels 45 to 145 will not be considered when computing the rms scatter in the visibilities.

Next, import the new uvfits file back into CASA using `importuvfits`.

Next, average over the channels of interest to create the single-channel visibility dataset of interest. Do this in CASA using `split`.

Next export the output from `split` to a uvfits file using `exportuvfits`. This is the file that you will use as input for `uvmcmcfit`.

CHAPTER 4

Examples

This section contains three examples in order of increasing complexity to demonstrate usage of `uvmcmcfit`.

4.1 XMM101: a single, unlensed galaxy

This example shows how to run `uvmcmcfit` on the simplest of systems: a single galaxy that is unaffected by lensing of any kind.

4.1.1 Preliminary Setup Procedures

Inputs

1. Establish a directory that contains data for the specific target for which you wish to measure a lens model. This is the directory from which you will run the software. I call this "uvfit00" for the first run on a given dataset, "uvfit01" for the second, etc.
2. Inside this directory, you must ensure the following files are present:

- config.yaml: This is the configuration file that describes where the source of interest is located, what type of model to use for the lens and source, the name of the image of the target from your interferometric data, the name of the uvfits files containing the interferometric visibilities, and a few important processing options as well. Syntax is yaml.
- Image of the target from your interferometric data in fits format. This image is used to set the spatial resolution of the model image (modified by an optional oversampling parameter).
- interferometric visibilities in uvfits format.

Outputs

posteriorpdf.fits: model parameters for every MCMC iteration, in fits format. Use astropy (either the fits or table module) to inspect the results directly.

4.1.2 Configuring config.yaml

config.yaml contains the instructions needed by `uvmcmcfit` to initiate the model fitting process.

Required keywords

A few house-keeping parameters:

```
# Name of the target
ObjectName: XMM101

# Name of the fits image; the pixel scale in this image sets the pixel
# scale in the model image
ImageName: XMM101.concat.statwt.cont.mfs.fits

# Name of the uvfits visibility data; the weights should be scaled such
# that Sum(weights * real) ~ N_vis [see uvutil.statwt()]
UVData: XMM101.concat.statwt.cont.uvfits

# Number of walkers
Nwalkers: 24
```

Caution: The number of walkers used by emcee must be more than double the number of parameters). In this case, there are only 6 parameters, so the minimum number of walkers is 12. I selected 24 to be on the safe side.

Now for parameters that describe the geometry of the system. You must define at least one region. The first region should be named `Region0`, the second `Region1`, etc. Pay attention to the indentation; the remaining keywords must be indented to indicate they are sub-components of `Region0`. For each region, you must define a RA and Dec center, an angular radial extent that contains the emission which you are attempting to model, and at least one source.

The first source should be named `Source0`, the second source should be named `Source1`, etc. Sources are elliptical Gaussians. Each source must have the following parameters: the total intrinsic flux density (IntrinsicFlux [mJy]), the effective radius defined as sqrt(a*b) (EffectiveRadius [arcsec]), the offset in RA and Dec from RACentroid and DecCentroid (DeltaRA and DeltaDec [arcsec]), the axial ratio (AxialRatio), and the position angle in degrees east of north (PositionAngle [degrees]).

For each source parameter, you must specify the lower and upper limits as well as how to initialize the walkers for that parameter. This is done using the following syntax: `Limits: [lower limit, lower initialization, upper initialization, upper limit]`. So, for example, in the code snippet below for XMM101, `Source0` is permitted to have a total intrinsic flux density ranging from 1 to 25 mJy, but is initialized with a uniform probability density distribution between 5 and 10 mJy.

```
# First region
Region0:

    # Right Ascension and Declination center of the model image (degrees)::
    RACentroid: 36.449395
    DecCentroid: -4.2974618

    # Angular radial extent of the model image (arcsec)
    RadialExtent: 1.5

    # Source0
    Source0:

        # total intrinsic flux density
        IntrinsicFlux:
            Limits: [1.0, 5.0, 10.0, 25.0]

        # effective radius of elliptical Gaussian [sqrt(a*b)] (arcsec)
        EffectiveRadius:
            Limits: [0.01, 0.01, 1.2, 1.2]
        # Offset in RA and Dec from RACentroid and DecCentroid (arcseconds)
        DeltaRA:
            Limits: [-0.4, -0.2, 0.2, 0.4]
        DeltaDec:
            Limits: [-0.4, -0.2, 0.2, 0.4]

        # axial ratio = semi-minor axis / semi-major axis
        AxialRatio:
            Limits: [0.2, 0.3, 1.0, 1.0]

        # position angle (degrees east of north)
        PositionAngle:
            Limits: [0.0, 0.0, 180.0, 180.0]
```

Optional keywords

By default, the maximum likelihood estimate is used to measure the goodness of fit. Alternatively, you may use the chi-squared value as the goodness of fit criterion via:

```
# Goodness of fit measurement
LogLike: chi2
```

By default, parallel processing is not used. To use parallel processing on a single machine, set the Nthreads variable to a number greater than 1. For example,

```
# Number of threads for multi-processing on a single computer
Nthreads: 2
```

If you have access to a computer cluster with many compute cores, you can use Message Passing Interface to greatly speed up the modeling process:

```
# Use Message Passing Interface
MPI: True
Nthreads: 1
```

Caution: Nthreads must be equal to 1 if using MPI!

If you want to compare the model results with an image obtained at another wavelength (e.g., an *HST* image), you must specify the location of the alternative image as well as the telescope and filter used to obtain the image:

```
# Alternative image name (used only for comparing with best-fit model)
OpticalImage: XMM101_F110W.fits

# Telescope and filter of alternative image
OpticalTag: HST F110W
```

4.1.3 Running `uvmcmcfit`

Assuming `uvmcmcfit` has been installed correctly (in this example I assume it is located in $uvsrc), usage should be as simple as navigating to the directory where you have placed the *.fits image, *.uvfits dataset, and config.yaml files. Then run:

```
python $uvsrc/uvmcmcfit.py
```

4.1.4 Inspecting the Model Results for XMM101

It is essential to inspect the results of the model fitting process to ensure the best-fit model and the associated parameter uncertainties are robust and acceptable. `uvmcmcfit` provides a suite of visualization routines that will assist you in this task.

Accessing the Data Directly

The model fit results are stored in a fits table called *posteriorpdf.fits*. You can inspect the results directly using the `astropy.table` module:

```
# import astropy's table module
import astropy.table as Table

# read the fit results
fitresults = Table.read('posteriorpdf.fits')

# get and print the column header names
keys = fitresults.keys()
print(keys)

# you can plot whatever aspect of fitresults you want from here...
```

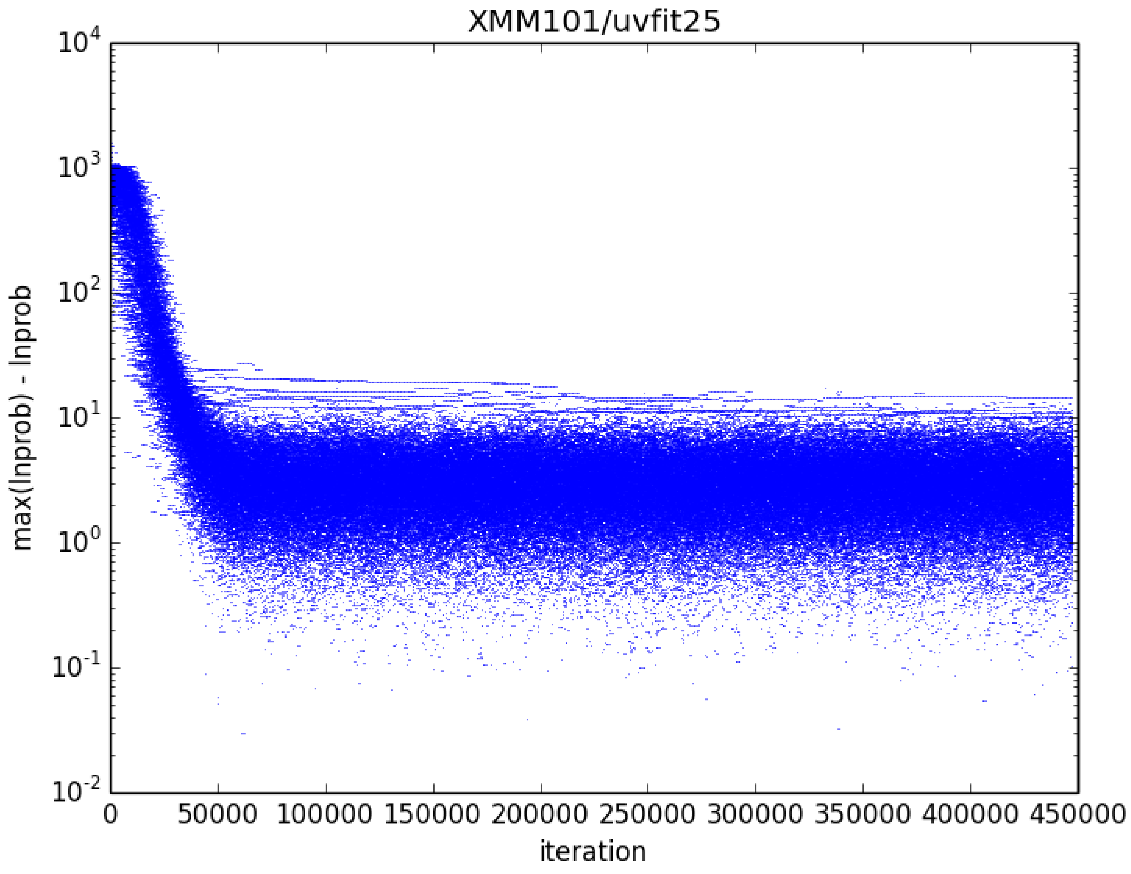
Testing Convergence

The first step in determining whether you have an acceptable model is to plot the convergence profile of the goodness of fit estimate (by default, the maximum likelihood estimate). You can do this using `visualize.convergence()`

```
import visualize
visualize.convergence()
```

This routine produces *convergence.png*, a plot of the difference between max(lnprob) and lnprob as a function of iteration. This turns out to be a convenient way to visualize the convergence, but it does mean that the best-fit model is shown at the bottom of the plot.

For XMM101, I needed about 50,000 iterations to reach convergence. The worst model fit has max(lnprob) - lnprob ~ 2e3. Both values are fairly typical for a single unlensed source like XMM101.



Plot the Best-fit Model

The second step is to plot the best-fit model and make sure that the model accurately reproduces the data. You can do this using `visualize.bestFit()`.

Some Preliminaries

Caution: You must run `visualize.bestFit()` from inside a CASA terminal OR you must install MIRIAD and add the following line to config.yaml:

```
UseMiriad: True
```

Note: To run `visualize.bestFit()` from inside CASA, follow these steps

1. Install casa-python. This makes it easy to install custom python packages in CASA using pip.
2. Install `pyyaml` and `astropy` into your CASA python environment.
   - `casa-pip install pyyaml`
   - `casa-pip install astropy`

3. Inspect $HOME/.casa/init.py and ensure that it contains a link to the directory where `pyyaml` and `astropy` were installed. In my case, the file already had the following:

   import site site.addsitedir("/Users/rbussman/.casa/lib/python2.7/site-packages")

   So, I had to add the following lines:

   site.addsitedir("/Users/rbussman/.casa/lib/python/site-packages") site.addsitedir("/Users/rbussman/python/uvmcmcfit")

   This allowed CASA to recognize that `pyyaml` and `uvmcmcfit` were installed. You may have placed `uvmcmcfit` in a different directory, so modify these instructions accordingly.

Note: To install MIRIAD on Mac, try the MIRIAD MacPorts page

Caution: If you use MIRIAD to make images of the best-fit model, you must create a special cshell script called *image.csh*. This file should contain the instructions needed for MIRIAD to invert and deconvolve the simulated visibilities from the best-fit model.

The Simplest Best-fit Plot

Generating a simple plot of the best-fit model should be straightforward:

```
import visualize
visualize.bestFit()
```

If you run this routine in CASA, you will enter an interactive cleaning session.

Note: See this ALMA tutorial for help on interactive cleaning.

After the cleaning session finishes, two plots will be produced like the ones shown below.

Left panel: ALMA 870um imaging of XMM101 (red contours, starting at +/-3-sigma and increasing by factors of sqrt(2)) overlaid on the best-fit model from `uvmcmcfit` (grayscale). The half-power shape of the source `Source0` in this case, is shown by a magenta ellipse. The shape of the synthesized beam is represented by the hatched black ellipse.

Right panel: Same as left panel, but showing the residual image after subtracting the best-fit model simulated visibilities from the observed visibilities. White and black contours trace positive and negative contours, respectively.

In this case, the model provides a good fit. There is evidence that the source is elongated in a direction perpendicular to the semi-major axis of the synthesized beam.

Comparison to Alternative Imaging

You can also compare the best-fit model to an image at an alternative wavelength (e.g., to compare lens properties with an optical or near-IR image of the lens). Do this by adding the following keyword:

```
visualize.bestFit(showOptical=True)
```

You should get the same results as above, but with an additional plot showing the comparison with the alternative image. Below is an example comparing the ALMA 870um imaging and best-fit model with *HST* F110W imaging.

Additional Options

You can turn off interactive cleaning in CASA:

```
visualize.bestFit(interactive=False)
```

`visualize.bestFit()` produces a large number of intermediate stage files that are automatically deleted as the last step in the program. These can sometimes be useful for debugging. To stop the program from automatically deleting all intermediate files, use the following option:

```
visualize.bestFit(cleanup=False)
```

Plot Several Acceptable Model Fits

It's nice to have visual confirmation that the best-fit model gives an acceptable fit to the data. It's even better to see that a random draw from the posterior probability density function (PDF) also gives an acceptable fit to the data. This is easily done using:

```
import visualize
visualize.goodFits()
```

By default, `visualize.goodFits()` produces plots of 12 model fits randomly drawn from the posterior PDF. You can adjust this with the `Nfits` keyword argument.

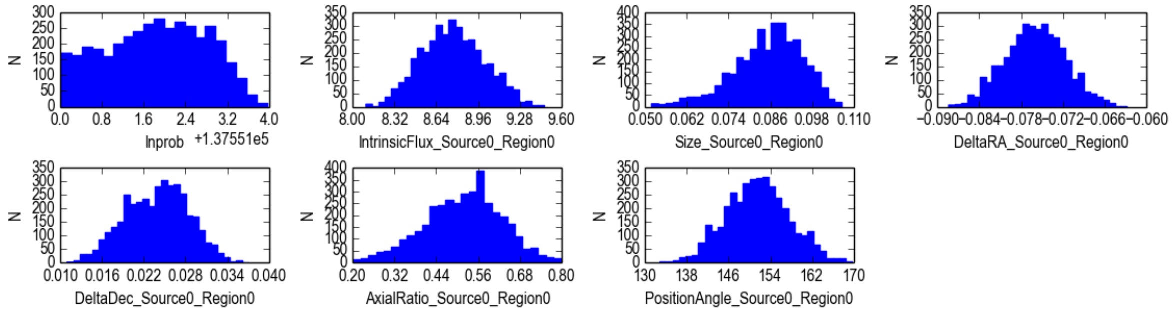
Note: Other than `Nfits`, `visualize.goodFits()` takes the same optional keyword arguments as `visualize.bestFit()`.

Plot the Posterior PDFs

You can examine the posterior probability density functions to get a quantitative look at the results of the model fitting process using `visualize.posteriorPDF()`:

```
import visualize
visualize.posteriorPDF()
```

This will produce a series of histograms showing the posterior probability distribution functions for every parameter in the model.



This routine also prints the average and 1-sigma rms uncertainty on each parameter of the model. We can see from the posterior PDF histograms that XMM101 appears to be elongated (axial ratio = 0.52 +/- 0.11) and has a total flux density at 870um of 8.76 +/- 0.24 mJy. It has an effective radius of 0.085 +/- 0.010 arcsec, which translates to a FWHM of 0.2 arcsec. At z=2 (the actual redshift of this object is unknown currently, but the *Herschel* photometry indicates z~2) this corresponds to a physical size of 1.7 kpc.

You can also see how the posterior PDF of every parameter in the model changes as a function of iteration using `visualize.evolvePDF()`:

```
visualize.evolvePDF()
```

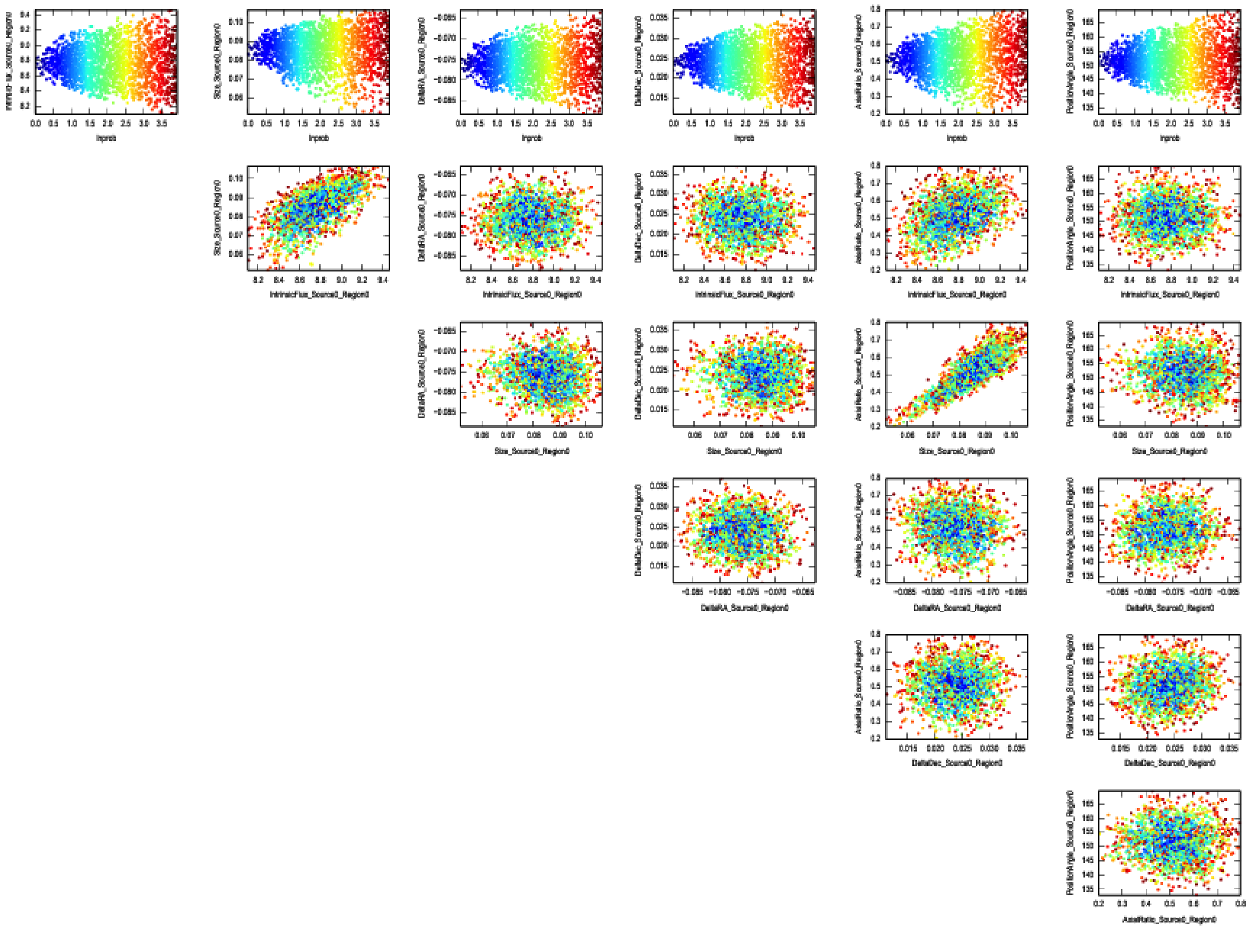
This function essentially produces a posteriorPDF every `stepsize` iterations. The default is `stepsize = 50000`. You can then view the evolution in the PDF using a viewer application like Preview.

Plot the Covariance Matrix

You might wish to examine how the various parameters are correlated with each other. You can do this by plotting the covariance matrix using `visualize.covariance()`:

```
import visualize
visualize.covariance()
```

This should produce a plot which looks similar to the image below. There appears to be a strong correlation between the axial ratio, size, and total flux density of the source.



4.2 ADFS07: A Single, Strongly Lensed Galaxy

This example shows how to run `uvmcmcfit` on the simplest strongly lensed system: a single galaxy that is lensed by a single intervening galaxy along the line of sight.

4.2.1 Preliminary Setup Procedures

Inputs

1. Establish a directory that contains data for the specific target for which you wish to measure a lens model. This is the directory from which you will run the software. I call this "uvfit00" for the first run on a given dataset, "uvfit01" for the second, etc.
2. Inside this directory, you must ensure the following files are present:

- config.yaml: This is the configuration file that describes where the source of interest is located, what type of model to use for the lens and source, the name of the image of the target from your interferometric data, the name of the uvfits files containing the interferometric visibilities, and a few important processing options as well. Syntax is yaml.
- Image of the target from your interferometric data in fits format. This image is used to set the spatial resolution of the model image (modified by an optional oversampling parameter).
- interferometric visibilities in uvfits format.

Outputs

posteriorpdf.fits: model parameters for every MCMC iteration, in fits format. Use astropy (either the fits or table module) to inspect the results directly.

4.2.2 Configuring config.yaml

config.yaml contains the instructions needed by `uvmcmcfit` to initiate the model fitting process.

Required keywords

A few house-keeping parameters:

```
# Name of the target
ObjectName: ADFS07

# Name of the fits image; the pixel scale in this image sets the pixel
# scale in the model image
ImageName: ADFS07.X28f.selfcal.statwt.cont.mfs.fits

# Name of the uvfits visibility data; the weights should be scaled such
# that Sum(weights * real) ~ N_vis [see uvutil.statwt()]
UVData: ADFS07.X28f.selfcal.statwt.cont.uvfits

# Number of walkers
Nwalkers: 44
```

Caution: The number of walkers used by emcee must be more than double the number of parameters). In this case, there are 11 parameters (5 for the lens, 6 for the source), so the minimum number of walkers is 22. I selected 44 to be on the safe side.

Now for parameters that describe the geometry of the system. You must define at least one region. The first region should be named `Region0`, the second `Region1`, etc. Pay attention to the indentation; the remaining keywords must be indented to indicate they are sub-components of `Region0`. For each region, you must define a RA and Dec center, an angular radial extent that contains the emission which you are attempting to model, and at least one source. You have the option to use the `Oversample` keyword to specify an integer factor by which to increase the resolution of the model image relative to the data image (i.e., relative to the resolution in the image specified by `ImageName`).

The first source should be named `Source0`, the second source should be named `Source1`, etc. Sources are elliptical Gaussians. Each source must have the following parameters: the total intrinsic flux density (IntrinsicFlux [mJy]), the effective radius (EffectiveRadius, defined as sqrt(a*b) [arcsec]), the offset in RA and Dec from RACentroid and DecCentroid (DeltaRA and DeltaDec [arcsec]), the axial ratio (AxialRatio, defined as b/a), and the position angle in degrees east of north (PositionAngle [degrees]).

For each source parameter, you must specify the lower and upper limits as well as how to initialize the walkers for that parameter. This is done using the following syntax: `Limits:  [lower limit, lower initialization, upper initialization, upper limit]`. So, for example, in the code snippet below for ADFS07, `Source0` is permitted to have a total intrinsic flux density ranging from 0.1 to 30 mJy, but is initialized with a uniform probability density distribution between 1 and 5 mJy.

You may account for the deflection of light rays caused by the presence of a galaxy or group of galaxies acting as a gravitational lens by specifying one or more lenses. The first lens should be named `Lens0`, the second lens should be named `Lens1`, etc.

Lenses are assumed to be singular isothermal ellipsoids. They are parameterized by: the Einstein radius (EinsteinRadius [arcsec]), the offset in RA and Dec from RACentroid and DecCentroid (DeltaRA and DeltaDec [arcsec]), the axial ratio (AxialRatio), and the position angle in degrees east of north (PositionAngle [degrees]).

Lens parameters are specified in the same way as source parameters: `Limits:  [lower limit, lower initialization, upper initialization, upper limit]`.

Note: It is sometimes desirable to specify the permissible range on a given parameter relative to another parameter of the model. For example, you might wish to force `Source0` to be north of `Lens0`. You can accomplish this by adding a line under the `Limits` specification for `DeltaDec` for `Source0` that says `FixedTo:  Region0 Lens0 DeltaDec`. This makes the program understand that DecSource0 = DecCentroid + DeltaDecLens0 + DeltaDecSource0, rather than simply DecSource0 = DecCentroid + DeltaDecSource0. The example below shows how to fix the RA and Dec of `Source0` relative to the RA and Dec of `Lens0`.

```
# First region
Region0:

    # Right Ascension and Declination center of the model image (degrees)::
    RACentroid: 70.4745
    DecCentroid: -54.064856

    # Angular radial extent of the model image (arcsec)
    RadialExtent: 2.2

    # [OPTIONAL]
    # Integer factor by which to increase resolution of model image
    Oversample: 2

    # Source0
    Source0:

        # total intrinsic flux density
        IntrinsicFlux:
            Limits: [0.1, 1.0, 5.0, 30.0]

        # effective radius of elliptical Gaussian [sqrt(a*b)] (arcsec)
        EffectiveRadius:
            Limits: [0.01, 0.1, 0.4, 1.5]
        # Offset in RA and Dec from RALens0 and DecLens0 (arcseconds)
        DeltaRA:
            FixedTo: Region0 Lens0 DeltaRA
            Limits: [-1.7, -0.3, 0.3, 1.7]
        DeltaDec:
            FixedTo: Region0 Lens0 DeltaDec
            Limits: [-1.7, -0.3, 0.3, 1.7]

        # axial ratio = semi-minor axis / semi-major axis
        AxialRatio:
            Limits: [0.2, 0.3, 1.0, 1.0]

        # position angle (degrees east of north)
        PositionAngle:
            Limits: [0.0, 0.0, 180.0, 180.0]

    # Lens0
    Lens0:

        # Einstein radius
        EinsteinRadius:
            Limits: [0.6, 1.0, 1.3, 2.0]

        # Offset in RA and Dec from RACentroid and DecCentroid (arcseconds)
        DeltaRA:
            Limits: [-0.8, -0.3, 0.0, 0.4]
        DeltaDec:
            Limits: [-0.6, -0.3, 0.0, 0.3]

        # axial ratio = semi-minor axis / semi-major axis
        AxialRatio:
            Limits: [0.3, 0.5, 0.9, 1.0]

        # position angle (degrees east of north)
        PositionAngle:
            Limits: [0.0, 0.0, 180.0, 180.0]
```

Optional keywords

By default, the maximum likelihood estimate is used to measure the goodness of fit. Alternatively, you may use the chi-squared value as the goodness of fit criterion via:

```
# Goodness of fit measurement
LogLike: chi2
```

By default, parallel processing is not used. To use parallel processing on a single machine, set the Nthreads variable to a number greater than 1. For example,

```
# Number of threads for multi-processing on a single computer
Nthreads: 2
```

If you have access to a computer cluster with many compute cores, you can use Message Passing Interface to greatly speed up the modeling process:

```
# Use Message Passing Interface
MPI: True
Nthreads: 1
```

Caution: Nthreads must be equal to 1 if using MPI!

If you want to compare the model results with an image obtained at another wavelength (e.g., an *HST* image), you must specify the location of the alternative image as well as the telescope and filter used to obtain the image:

```
# Alternative image name (used only for comparing with best-fit model)
OpticalImage: ADFS07_F110W.fits

# Telescope and filter of alternative image
OpticalTag: HST F110W
```

4.2.3 Running `uvmcmcfit`

Assuming `uvmcmcfit` has been installed correctly (in this example I assume it is located in $uvsrc), usage should be as simple as navigating to the directory where you have placed the *.fits image, *.uvfits dataset, and config.yaml files. Then run:

```
python $uvsrc/uvmcmcfit.py
```

4.2.4 Inspecting the Model Results for ADFS07

It is essential to inspect the results of the model fitting process to ensure the best-fit model and the associated parameter uncertainties are robust and acceptable. `uvmcmcfit` provides a suite of visualization routines that will assist you in this task.

Accessing the Data Directly

The model fit results are stored in a fits table called *posteriorpdf.fits*. You can inspect the results directly using the `astropy.table` module:

```
# import astropy's table module
import astropy.table as Table

# read the fit results
fitresults = Table.read('posteriorpdf.fits')

# get and print the column header names
keys = fitresults.keys()
print(keys)

# you can plot whatever aspect of fitresults you want from here...
```

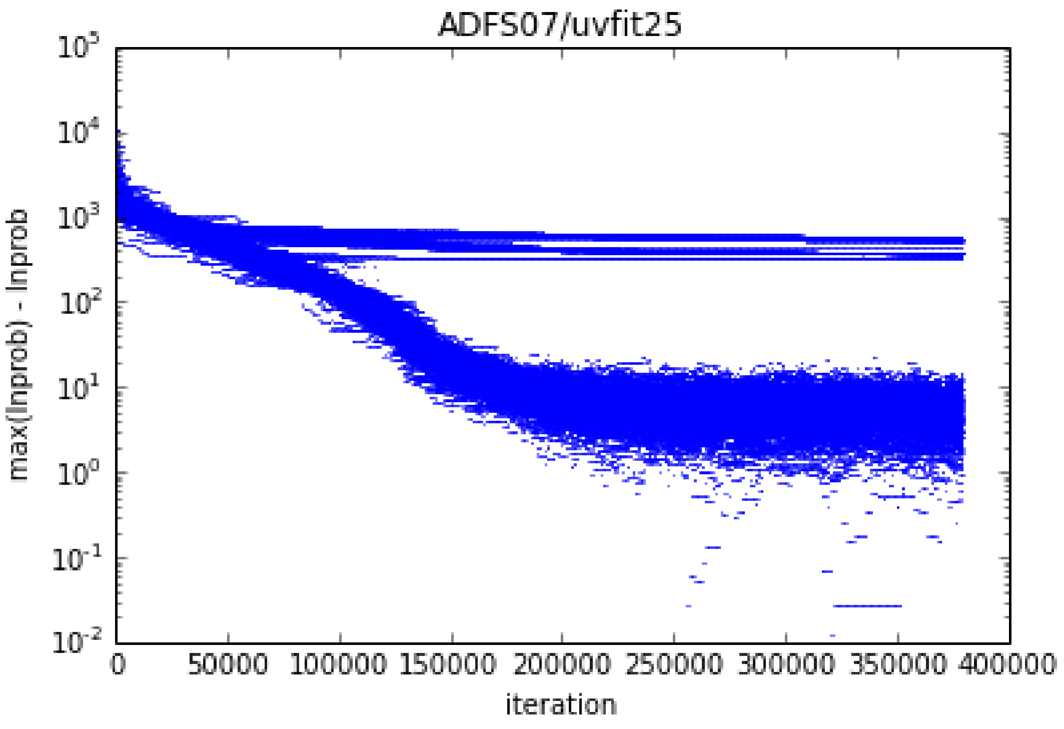
Testing Convergence

The first step in determining whether you have an acceptable model is to plot the convergence profile of the goodness of fit estimate (by default, the maximum likelihood estimate). You can do this using `visualize.convergence()`

```
import visualize
visualize.convergence()
```

This routine produces *convergence.png*, a plot of the difference between max(lnprob) and lnprob as a function of iteration. This turns out to be a convenient way to visualize the convergence, but it does mean that the best-fit model is shown at the bottom of the plot.

For ADFS07, I needed about 250,000 iterations to reach convergence. The worst model fit has max(lnprob) - lnprob ~ 1e5. Both values are fairly typical for a single strongly lensed source like ADFS07.



Plot the Best-fit Model

The second step is to plot the best-fit model and make sure that the model accurately reproduces the data. You can do this using `visualize.bestFit()`.

Some Preliminaries

Caution: You must run `visualize.bestFit()` from inside a CASA terminal OR you must install MIRIAD and add the following line to config.yaml:

```
UseMiriad: True
```

Note: To run `visualize.bestFit()` from inside CASA, follow these steps

1. Install casa-python. This makes it easy to install custom python packages in CASA using pip.
2. Install `pyyaml` and `astropy` into your CASA python environment.
   - `casa-pip install pyyaml`
   - `casa-pip install astropy`

3. Inspect $HOME/.casa/init.py and ensure that it contains a link to the directory where `pyyaml` and `astropy` were installed. In my case, the file already had the following:

   import site site.addsitedir("/Users/rbussman/.casa/lib/python2.7/site-packages")

   So, I had to add the following lines:

   site.addsitedir("/Users/rbussman/.casa/lib/python/site-packages") site.addsitedir("/Users/rbussman/python/uvmcmcfit")

   This allowed CASA to recognize that `pyyaml` and `uvmcmcfit` were installed. You may have placed `uvmcmcfit` in a different directory, so modify these instructions accordingly.

Note: To install MIRIAD on Mac, try the MIRIAD MacPorts page

Caution: If you use MIRIAD to make images of the best-fit model, you must create a special cshell script called *image.csh*. This file should contain the instructions needed for MIRIAD to invert and deconvolve the simulated visibilities from the best-fit model.

The Simplest Best-fit Plot

Generating a simple plot of the best-fit model should be straightforward:

```
import visualize
visualize.bestFit()
```

If you run this routine in CASA, you will enter an interactive cleaning session.

Note: See this ALMA tutorial for help on interactive cleaning.

After the cleaning session finishes, two plots will be produced like the ones shown below.

Left panel: ALMA 870um imaging of ADFS07 (red contours, starting at +/-3-sigma and increasing by factors of sqrt(2)) overlaid on the best-fit model from `uvmcmcfit` (grayscale). The half-power shape of the source `Source0` in this case, is shown by a magenta ellipse. The shape of the synthesized beam is represented by the hatched black ellipse.

Right panel: Same as left panel, but showing the residual image after subtracting the best-fit model simulated visibilities from the observed visibilities. White and black contours trace positive and negative contours, respectively.

In this case, the model provides a good fit. There is evidence that the source is elongated in a direction perpendicular to the semi-major axis of the synthesized beam.

Comparison to Alternative Imaging

You can also compare the best-fit model to an image at an alternative wavelength (e.g., to compare lens properties with an optical or near-IR image of the lens). Do this by adding the following keyword:

```
visualize.bestFit(showOptical=True)
```

You should get the same results as above, but with an additional plot showing the comparison with the alternative image. Below is an example comparing the ALMA 870um imaging and best-fit model with *HST* F110W imaging.

Here, the postion of the lens according to the best-fit model is offset from the position of the lens according to the *HST* image by ~0.6 arcsec. The astrometry in the *HST* image is tied to a Gemini-South optical image of this source to ~0.1 arcsec rms precision. However the Gemini-South image astrometry is tied to 2MASS imaging and therefore has an absolute uncertainty of ~0.5 arcsec rms. So this result is a little farther off than we might expect, but it's not worth losing sleep over. Data from the VISTA Hemisphere Survey (VHS) will be helpful for this application due to the deeper imaging and better image quality of the VHS.

Additional Options

You can turn off interactive cleaning in CASA:

```
visualize.bestFit(interactive=False)
```

`visualize.bestFit()` produces a large number of intermediate stage files that are automatically deleted as the last step in the program. These can sometimes be useful for debugging. To stop the program from automatically deleting all intermediate files, use the following option:

```
visualize.bestFit(cleanup=False)
```

Plot Several Acceptable Model Fits

It's nice to have visual confirmation that the best-fit model gives an acceptable fit to the data. It's even better to see that a random draw from the posterior probability density function (PDF) also gives an acceptable fit to the data. This is easily done using:

```
import visualize
visualize.goodFits()
```

By default, `visualize.goodFits()` produces plots of 12 model fits randomly drawn from the posterior PDF. You can adjust this with the `Nfits` keyword argument.

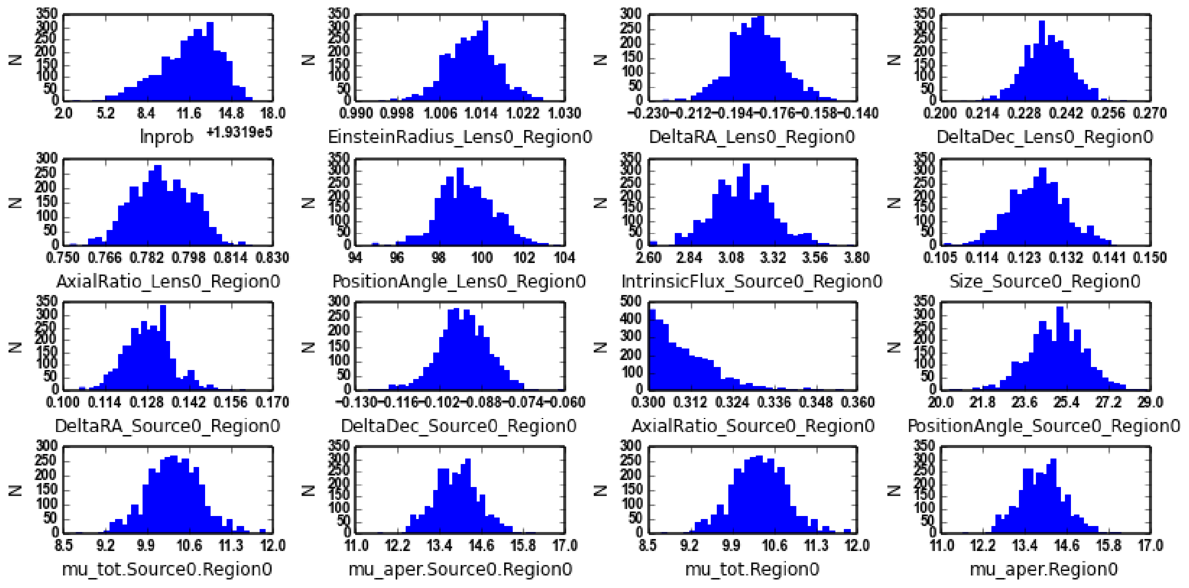
Note: Other than `Nfits`, `visualize.goodFits()` takes the same optional keyword arguments as `visualize.bestFit()`.

Plot the Posterior PDFs

You can examine the posterior probability density functions to get a quantitative look at the results of the model fitting process using `visualize.posteriorPDF()`:

```
import visualize
visualize.posteriorPDF()
```

This will produce a series of histograms showing the posterior probability distribution functions for every parameter in the model.



This routine also prints the average and 1-sigma rms uncertainty on each parameter of the model.

You can also see how the posterior PDF of every parameter in the model changes as a function of iteration using `visualize.evolvePDF()`:

```
visualize.evolvePDF()
```

This function essentially produces a posteriorPDF every `stepsize` iterations. The default is `stepsize = 50000`. You can then view the evolution in the PDF using a viewer application like Preview.

Plot the Covariance Matrix

You might wish to examine how the various parameters are correlated with each other. You can do this by plotting the covariance matrix using `visualize.covariance()`:

```
import visualize
visualize.covariance()
```

This will produce an enormous plot showing correlations between all of the model parameters. The plot is too big to reproduce here, but you can use Preview to zoom in on any panel of interest. There is evidence that `DeltaRA` for `Lens0` is highly anti-correlated with `DeltaRA` for `Source0`. This is expected since the source position has been fixed relative to the lens position in this model fit. The same holds true for `DeltaDec`.

4.3 CDFS_M0: A Complex, Multi-component System

This example shows how to run `uvmcmcfit` on a complex system: a strongly lensed galaxy, a weakly lensed galaxy, and a sub-mm bright lens.

4.3.1 Preliminary Setup Procedures

Inputs

1. Establish a directory that contains data for the specific target for which you wish to measure a lens model. This is the directory from which you will run the software. I call this "uvfit00" for the first run on a given dataset, "uvfit01" for the second, etc.
2. Inside this directory, you must ensure the following files are present:

- config.yaml: This is the configuration file that describes where the source of interest is located, what type of model to use for the lens and source, the name of the image of the target from your interferometric data, the name of the uvfits files containing the interferometric visibilities, and a few important processing options as well. Syntax is yaml.
- Image of the target from your interferometric data in fits format. This image is used to set the spatial resolution of the model image (modified by an optional oversampling parameter).
- interferometric visibilities in uvfits format.

Outputs

posteriorpdf.fits: model parameters for every MCMC iteration, in fits format. Use astropy (either the fits or table module) to inspect the results directly.

4.3.2 Configuring config.yaml

config.yaml contains the instructions needed by `uvmcmcfit` to initiate the model fitting process.

Required keywords

A few house-keeping parameters:

```
# Name of the target
ObjectName: CDFS_M0

# Name of the fits image; the pixel scale in this image sets the pixel
# scale in the model image
ImageName: CDFS_M0.Xabf.selfcal.statwt.cont.mfs.fits

# Name of the uvfits visibility data; the weights should be scaled such
# that Sum(weights * real) ~ N_vis [see uvutil.statwt()]
UVData: CDFS_M0.Xabf.selfcal.statwt.cont.uvfits

# Number of walkers
Nwalkers: 92
```

Caution: The number of walkers used by emcee must be more than double the number of parameters). In this case, there are 23 parameters (5 for the lens, 6 for each of three sources), so the minimum number of walkers is 46. I selected 92 to be on the safe side.

Now for parameters that describe the geometry of the system. You must define at least one region. The first region should be named `Region0`, the second `Region1`, etc. Pay attention to the indentation; the remaining keywords must be indented to indicate they are sub-components of `Region0`. For each region, you must define a RA and Dec center, an angular radial extent that contains the emission which you are attempting to model, and at least one source. You have the option to use the `Oversample` keyword to specify an integer factor by which to increase the resolution of the model image relative to the data image (i.e., relative to the resolution in the image specified by `ImageName`).

The first source should be named `Source0`, the second source should be named `Source1`, etc. Sources are elliptical Gaussians. Each source must have the following parameters: the total intrinsic flux density (IntrinsicFlux [mJy]), the effective radius (EffectiveRadius, defined as sqrt(a*b) [arcsec]), the offset in RA and Dec from RACentroid and DecCentroid (DeltaRA and DeltaDec [arcsec]), the axial ratio (AxialRatio, defined as b/a), and the position angle in degrees east of north (PositionAngle [degrees]).

For each source parameter, you must specify the lower and upper limits as well as how to initialize the walkers for that parameter. This is done using the following syntax: `Limits:  [lower limit, lower initialization, upper initialization, upper limit]`. So, for example, in the code snippet below for CDFS_M0, `Source0` is permitted to have a total intrinsic flux density ranging from 0.1 to 30 mJy, but is initialized with a uniform probability density distribution between 1 and 5 mJy.

You may account for the deflection of light rays caused by the presence of a galaxy or group of galaxies acting as a gravitational lens by specifying one or more lenses. The first lens should be named `Lens0`, the second lens should be named `Lens1`, etc.

Lenses are assumed to be singular isothermal ellipsoids. They are parameterized by: the Einstein radius (EinsteinRadius [arcsec]), the offset in RA and Dec from RACentroid and DecCentroid (DeltaRA and DeltaDec [arcsec]), the axial ratio (AxialRatio), and the position angle in degrees east of north (PositionAngle [degrees]).

Lens parameters are specified in the same way as source parameters: `Limits:  [lower limit, lower initialization, upper initialization, upper limit]`.

Note: It is sometimes desirable to specify the permissible range on a given parameter relative to another parameter of the model. For example, you might wish to force `Source0` to be north of `Lens0`. You can accomplish this by adding a line under the `Limits` specification for `DeltaDec` for `Source0` that says `FixedTo:  Region0 Lens0 DeltaDec`. This makes the program understand that DecSource0 = DecCentroid + DeltaDecLens0 + DeltaDecSource0, rather than simply DecSource0 = DecCentroid + DeltaDecSource0. The example below shows how to fix the RA and Dec of `Source0` relative to the RA and Dec of `Lens0`.

Note: In some cases, both the lens itself and the lensed emission is detected by the interferometer. The best way I have found to deal with this situation is to create two regions with the same position center and angular extent, `Region0` and `Region1`, corresponding to the lensed and lens emission, respectively. These regions will be modeled simultaneously, so that there is no need to do a “lens-subtraction” prior to modeling the lensed emission. An example based on CDFS_M0 follows below.

Note: You can fix a parameter in the model to a given value by specifying both the lower and upper initialization to have that same value. An example of how to do this is shown for `DeltaRA` and `DeltaDec` in `Source0` of `Region1`.

```
# First region: contains emission from the lensed galaxies
Region0:

    # Right Ascension and Declination center of the model image (degrees)::
    RACentroid: 51.966712
    DecCentroid: -29.152889

    # Angular radial extent of the model image (arcsec)
    RadialExtent: 3.0

    # [OPTIONAL]
    # Integer factor by which to increase resolution of model image
    Oversample: 2
```

```
# Source0 -- this source is strongly lensed
Source0:

    # total intrinsic flux density
    IntrinsicFlux:
        Limits: [0.1, 1.0, 2.0, 30.0]

    # effective radius of elliptical Gaussian [sqrt(a*b)] (arcsec)
    EffectiveRadius:
        Limits: [0.01, 0.1, 0.4, 1.5]

    # Offset in RA and Dec from RALens0 and DecLens0 (arcseconds)
    DeltaRA:
        FixedTo: Region0 Lens0 DeltaRA
        Limits: [-1.7, -0.4, -0.3, 1.7]
    DeltaDec:
        FixedTo: Region0 Lens0 DeltaDec
        Limits: [-0.7, 0.1, 0.2, 0.7]

    # axial ratio = semi-minor axis / semi-major axis
    AxialRatio:
        Limits: [0.3, 0.3, 1.0, 1.0]

    # position angle (degrees east of north)
    PositionAngle:
        Limits: [0.0, 0.0, 180.0, 180.0]

# Source1 -- this source is weakly lensed
Source1:

    # total intrinsic flux density
    IntrinsicFlux:
        Limits: [0.1, 6.0, 8.0, 30.0]

    # effective radius of elliptical Gaussian [sqrt(a*b)] (arcsec)
    EffectiveRadius:
        Limits: [0.01, 0.1, 0.4, 1.5]

    # Offset in RA and Dec from RACentroid and DecCentroid (arcseconds)
    DeltaRA:
        Limits: [-1.2, -0.5, 0.0, 0.2]
    DeltaDec:
        Limits: [0.5, 1.2, 1.8, 2.5]

    # axial ratio = semi-minor axis / semi-major axis = b/a
    AxialRatio:
        Limits: [0.3, 0.3, 1.0, 1.0]

    # position angle (degrees east of north)
    PositionAngle:
        Limits: [0.0, 0.0, 180.0, 180.0]

# Lens0
Lens0:

    # Einstein radius
    EinsteinRadius:
        Limits: [0.4, 1.0, 1.5, 2.0]
        # Offset in RA and Dec from RACentroid and DecCentroid (arcseconds)
        DeltaRA:
            Limits: [0.1, 0.2, 0.25, 0.3]
        DeltaDec:
            Limits: [-1.9, -1.8, -1.75, -1.7]

        # axial ratio = semi-minor axis / semi-major axis
        AxialRatio:
            Limits: [0.3, 0.7, 0.9, 1.0]

        # position angle (degrees east of north)
        PositionAngle:
            Limits: [0.0, 0.0, 180.0, 180.0]

# Second region: contains emission from the lens
Region1:

    # Right Ascension and Declination center of the model image (degrees)::
    RACentroid: 51.966712
    DecCentroid: -29.152889

    # Angular radial extent of the model image (arcsec)
    RadialExtent: 3.0

    # [OPTIONAL]
    # Integer factor by which to increase resolution of model image
    Oversample: 2

    # Source0 -- this is the lens
    Source0:

        # total intrinsic flux density
        IntrinsicFlux:
            Limits: [0.1, 5.0, 6.0, 30.0]

        # effective radius of elliptical Gaussian [sqrt(a*b)] (arcsec)
        EffectiveRadius:
            Limits: [0.01, 0.1, 0.2, 0.5]

        # Offset in RA and Dec from RALens0 and DecLens0 (arcseconds)
        # I assume the center of the gravitational potential is coincident
        # with the emission centroid of the lensing galaxy
        DeltaRA:
            FixedTo: Region0 Lens0 DeltaRA
            Limits: [-1.7, 0.0, 0.0, 1.7]
        DeltaDec:
            FixedTo: Region0 Lens0 DeltaDec
            Limits: [-0.7, 0.0, 0.0, 0.7]

        # axial ratio = semi-minor axis / semi-major axis
        AxialRatio:
            Limits: [0.3, 0.3, 1.0, 1.0]

        # position angle (degrees east of north)
        PositionAngle:
            Limits: [0.0, 0.0, 180.0, 180.0]
```

Optional keywords

By default, the maximum likelihood estimate is used to measure the goodness of fit. Alternatively, you may use the chi-squared value as the goodness of fit criterion via:

```
# Goodness of fit measurement
LogLike: chi2
```

By default, parallel processing is not used. To use parallel processing on a single machine, set the Nthreads variable to a number greater than 1. For example,

```
# Number of threads for multi-processing on a single computer
Nthreads: 2
```

If you have access to a computer cluster with many compute cores, you can use Message Passing Interface to greatly speed up the modeling process:

```
# Use Message Passing Interface
MPI: True
Nthreads: 1
```

Caution: Nthreads must be equal to 1 if using MPI!

If you want to compare the model results with an image obtained at another wavelength (e.g., an *HST* image), you must specify the location of the alternative image as well as the telescope and filter used to obtain the image:

```
# Alternative image name (used only for comparing with best-fit model)
OpticalImage: CDFS_M0_Ks.fits

# Telescope and filter of alternative image
OpticalTag: VIDEO Ks
```

4.3.3 Running `uvmcmcfit`

Assuming `uvmcmcfit` has been installed correctly (in this example I assume it is located in $uvsrc), usage should be as simple as navigating to the directory where you have placed the *.fits image, *.uvfits dataset, and config.yaml files. Then run:

```
python $uvsrc/uvmcmcfit.py
```

4.3.4 Inspecting the Model Results for CDFS_M0

It is essential to inspect the results of the model fitting process to ensure the best-fit model and the associated parameter uncertainties are robust and acceptable. `uvmcmcfit` provides a suite of visualization routines that will assist you in this task.

Accessing the Data Directly

The model fit results are stored in a fits table called *posteriorpdf.fits*. You can inspect the results directly using the `astropy.table` module:

```
# import astropy's table module
import astropy.table as Table
```

```
# read the fit results
fitresults = Table.read('posteriorpdf.fits')

# get and print the column header names
keys = fitresults.keys()
print(keys)

# you can plot whatever aspect of fitresults you want from here...
```

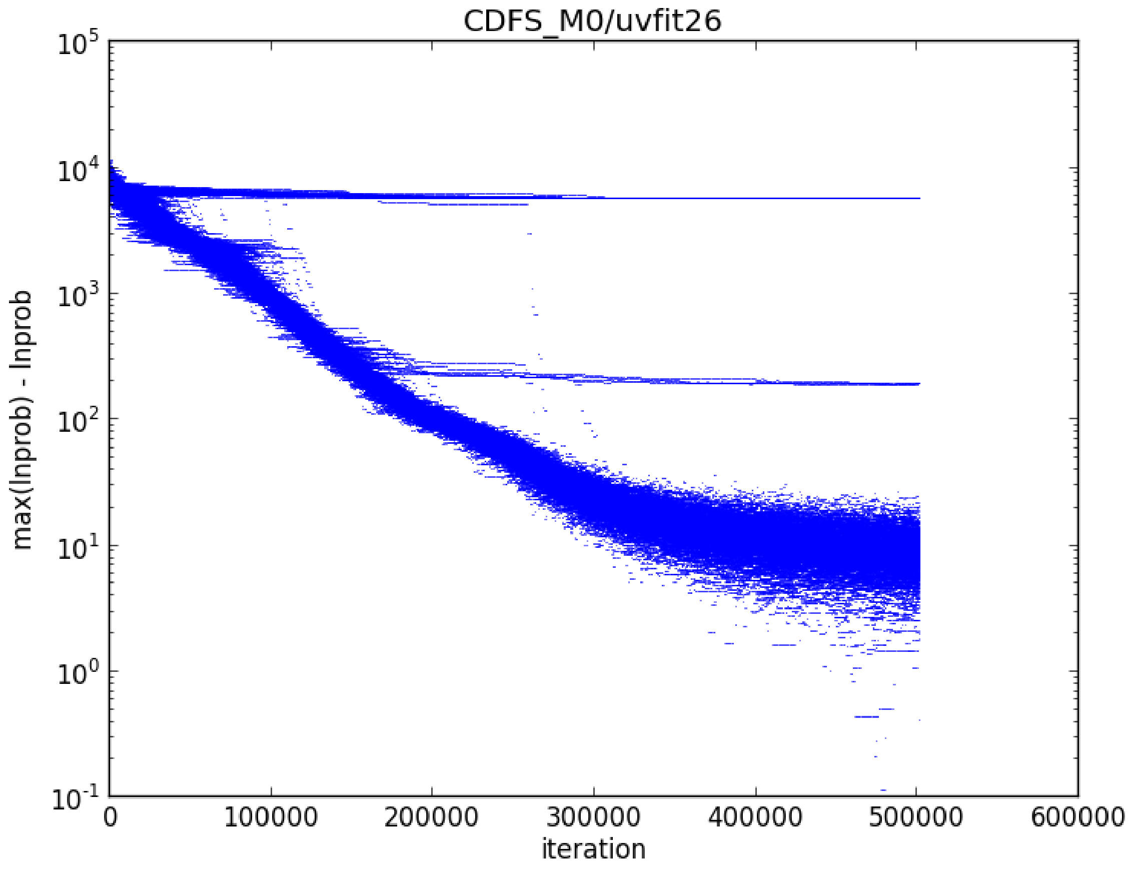
Testing Convergence

The first step in determining whether you have an acceptable model is to plot the convergence profile of the goodness of fit estimate (by default, the maximum likelihood estimate). You can do this using `visualize.convergence()`

```
import visualize
visualize.convergence()
```

This routine produces *convergence.png*, a plot of the difference between max(lnprob) and lnprob as a function of iteration. This turns out to be a convenient way to visualize the convergence, but it does mean that the best-fit model is shown at the bottom of the plot.

For CDFS_M0, I needed about 450,000 iterations to reach convergence. The worst model fit has max(lnprob) - lnprob ~ 2e4. Both values are fairly typical for a complex system like CDFS_M0.



Plot the Best-fit Model

The second step is to plot the best-fit model and make sure that the model accurately reproduces the data. You can do this using `visualize.bestFit()`.

Some Preliminaries

Caution: You must run `visualize.bestFit()` from inside a CASA terminal OR you must install MIRIAD and add the following line to config.yaml:

```
UseMiriad: True
```

Note: To run `visualize.bestFit()` from inside CASA, follow these steps

1. Install casa-python. This makes it easy to install custom python packages in CASA using pip.
2. Install `pyyaml` and `astropy` into your CASA python environment.
   - `casa-pip install pyyaml`
   - `casa-pip install astropy`
3. Inspect $HOME/.casa/init.py and ensure that it contains a link to the directory where `pyyaml` and `astropy` were installed. In my case, the file already had the following:

   import site site.addsitedir("/Users/rbussman/.casa/lib/python2.7/site-packages")

   So, I had to add the following lines:

   site.addsitedir("/Users/rbussman/.casa/lib/python/site-packages") site.addsitedir("/Users/rbussman/python/uvmcmcfit")

   This allowed CASA to recognize that `pyyaml` and `uvmcmcfit` were installed. You may have placed `uvmcmcfit` in a different directory, so modify these instructions accordingly.

Note: To install MIRIAD on Mac, try the MIRIAD MacPorts page

Caution: If you use MIRIAD to make images of the best-fit model, you must create a special cshell script called *image.csh*. This file should contain the instructions needed for MIRIAD to invert and deconvolve the simulated visibilities from the best-fit model.

The Simplest Best-fit Plot

Generating a simple plot of the best-fit model should be straightforward:

```python
import visualize
visualize.bestFit()
```

If you run this routine in CASA, you will enter an interactive cleaning session.

Note: See this ALMA tutorial for help on interactive cleaning.

After the cleaning session finishes, two plots will be produced like the ones shown below.

Left panel: ALMA 870um imaging of CDFS_M0 (red contours, starting at +/-3-sigma and increasing by factors of sqrt(2)) overlaid on the best-fit model from `uvmcmcfit` (grayscale). The half-power shape of the source `Source0` in this case, is shown by a magenta ellipse. The shape of the synthesized beam is represented by the hatched black ellipse.

Right panel: Same as left panel, but showing the residual image after subtracting the best-fit model simulated visibilities from the observed visibilities. White and black contours trace positive and negative contours, respectively.

Comparison to Alternative Imaging

You can also compare the best-fit model to an image at an alternative wavelength (e.g., to compare lens properties with an optical or near-IR image of the lens). Do this by adding the following keyword:

```
visualize.bestFit(showOptical=True)
```

You should get the same results as above, but with an additional plot showing the comparison with the alternative image. Below is an example comparing the ALMA 870um imaging and best-fit model with VISTA VIDEO Ks-band imaging.

Additional Options

You can turn off interactive cleaning in CASA:

```
visualize.bestFit(interactive=False)
```

`visualize.bestFit()` produces a large number of intermediate stage files that are automatically deleted as the last step in the program. These can sometimes be useful for debugging. To stop the program from automatically deleting all intermediate files, use the following option:

```
visualize.bestFit(cleanup=False)
```

Plot Several Acceptable Model Fits

It's nice to have visual confirmation that the best-fit model gives an acceptable fit to the data. It's even better to see that a random draw from the posterior probability density function (PDF) also gives an acceptable fit to the data. This is easily done using:

```
import visualize
visualize.goodFits()
```

By default, `visualize.goodFits()` produces plots of 12 model fits randomly drawn from the posterior PDF. You can adjust this with the `Nfits` keyword argument.

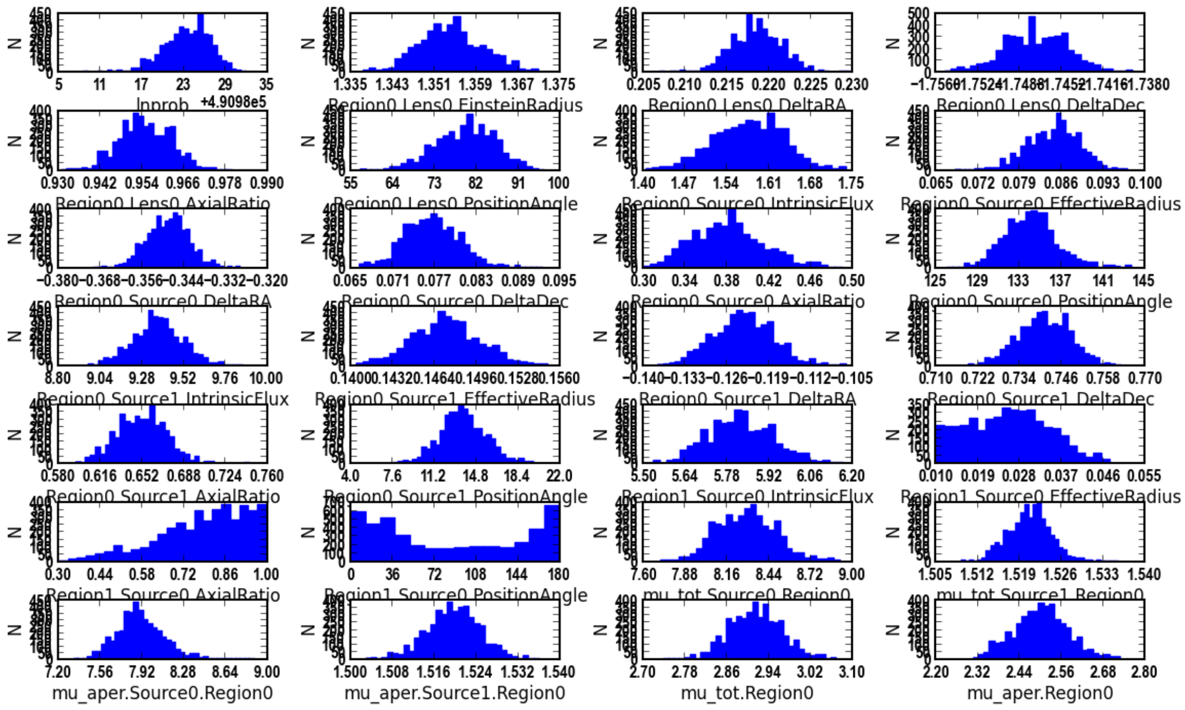
Note: Other than `Nfits`, `visualize.goodFits()` takes the same optional keyword arguments as `visualize.bestFit()`.

Plot the Posterior PDFs

You can examine the posterior probability density functions to get a quantitative look at the results of the model fitting process using `visualize.posteriorPDF()`:

```
import visualize
visualize.posteriorPDF()
```

This will produce a series of histograms showing the posterior probability distribution functions for every parameter in the model.



This routine also prints the average and 1-sigma rms uncertainty on each parameter of the model.

You can also see how the posterior PDF of every parameter in the model changes as a function of iteration using `visualize.evolvePDF()`:

```
visualize.evolvePDF()
```

This function essentially produces a posteriorPDF every `stepsize` iterations. The default is `stepsize = 50000`. You can then view the evolution in the PDF using a viewer application like Preview.

Plot the Covariance Matrix

You might wish to examine how the various parameters are correlated with each other. You can do this by plotting the covariance matrix using `visualize.covariance()`:

```
import visualize
visualize.covariance()
```

This will produce an enormous plot showing correlations between all of the model parameters. The plot is far too big to reproduce here, but you can use Preview to zoom in on any panel of interest.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search