RECONSTRUCTION TEST REPORT AND DATA PROCESSING COOKBOOKS

OPTICON FP7-2 Report (JRA.4 DELIVERABLE)

Authors:

Joel Sanchez-Bermudez (Max-Planck-Institut fuer Astronomie -MPIA-, Heidelberg, Germany)

Eric Thiébaut (Centre de Recherche Astrophysique de Lyon, Lyon, France)

Gilles Duvert (Observatoire de Grenoble and IPAG, Grenoble, France)

Guillaume Mella (Observatoire de Grenoble and IPAG, Grenoble, France)

John Young (Univ. of Cambridge, Cambridge, United Kingdom)

J-Uwe Pott (Max-Planck-Institut fuer Astronomie -MPIA-, Heidelberg, Germany)

Nuno Gomes (Universidade do Porto - Faculdade de Engenharia, Departamento de Engenharia Física -CENTRA-, Porto, Portugal)

Paulo J. V. Garcia (Universidade do Porto - Faculdade de Engenharia, Departamento de Engenharia Física -CENTRA-, Porto, Portugal)

December, 2016
Heidelberg, Germany

ABSTRACT

Interferometers provide sparse measurements of the Fourier transform of the brightness distribution for a given source in the plane of the sky. Therefore, images must be reconstructed from these observables. Image reconstruction from interferometric data is, in principle, a Fourier inversion problem. Nevertheless, there are some details to take into account, particularly for the case of data obtained with optical interferometers. For example, (a) the sparseness of the $u-v$ coverage and; (b) the poor calibration of the amplitude of the visibilities due to the strong non-modeled variations produced by the atmospheric turbulence. During the last decade, several programs have been developed in the community to reconstruct images from optical interferometric data. Nevertheless, their proper use requires understanding their main characteristics and settings. Therefore, the user needs training and extensive documentation to maximize their scientific use. Here, we present comprehensive user guidelines for generating simulated interferometric observables from images, and reconstructing images from real or simulated observables using some of the most important and widely-used reconstruction software packages. The different chapters include a collection of examples and commands that the user should follow in order to properly recover images from interferometric data. We also described the properties of a newly developed Graphical User Interface which will allow several alternative algorithms to be used and their results compared.

CONTENTS

LIST OF FIGURES

1

INTRODUCTION

Except perhaps for the most simple objects, interferometric data are hard to interpret directly and image reconstruction is required to analyze the observations. Restoring a correct image from optical interferometric data is challenging because of the sparsity of the measurements and of missing information. Several image reconstruction algorithms have been developed to cope with optical interferometric data and most of them are freely available. These algorithms are however not *black boxes* which could produce reliable images without any human interaction. In fact, for fundamental reasons, image reconstruction from optical interferometric data *cannot* be implemented by a black-box method and user input is required to wisely choose among a number of settings. In a nutshell, because of the sparsity of the data (which is must worse than in radio-interferometry for instance), an infinite number of different images may explain the observations as well and additional constraints are needed to select a unique image among all of these. The choice of such constraints must be carried out with great attention as it has a determining influence on the resulting image. A good image reconstruction is therefore not just the matter of having a good algorithm but also of using it correctly.

We nevertheless believe that understanding the principles of image reconstruction and getting accustomed to the consequences of the different settings of a given algorithm is not difficult. Especially nowadays, as a good number of image reconstruction algorithms have evolved[1] to be easy to play with. To make things even easier and interactive, we developed a graphical user interface (GUI, see Chapter 4) which hides the complexity of some methods and provides a unified interface for different algorithms. Finally, the lack of a single black-box software, which can be thought as a drawback, is really a strength as it is an incitement to try different methods and to play with their parameters. Critical analysis of the results not only helps to find the right combinations of

[1] for some of them, this evolution is a consequence of this JRA

algorithms and parameters but provides a unique insight about the actual contents of the data.

The objective of these cookbooks are to introduce the reader to the use of a number of algorithms. We hope that more algorithms will be part of the interface in a near future. The following sections provides a simple summary to recall the principles of image reconstruction from interferometric data and list the ingredients of an image reconstruction method. These sections also introduce the notation used in this report.

Understanding the principles of image reconstruction is recommended for the proper use of a given algorithm. Fortunately, most, if not all, image reconstruction methods follow similar approaches which are described in details in a companion document (Young and Thiébaut, 2015) as well as in comprehensive reviews (Baron, 2016; Thiébaut, 2009; Thiébaut and Giovannelli, 2010).

Since direct inversion of the data is neither possible nor recommended, image reconstruction algorithms proceed by iteratively solving an *inverse problem* where the model of the data given the object is compared to the actual data in order to determine how to vary the image to better fit the data while respecting given constraints such as positivity, regularity, *etc.* These constraints are needed to avoid over-fitting of the data (and thus explaining the noise as well as the significant signal) and to compensate for the sparsity of the data which leads to an under-determined problem.

In practice, an image is represented by a discrete set of real values, say $\boldsymbol{x} \in \mathbb{R}^n$. Most commonly, these values are those of the pixels in the sought image, consequently n can be fairly large, in particular much larger than the number of measurements. In order to quantitatively judge whether an image of the object of interest is in accordance with the measurements, some numerical criterion, say $f_{\text{data}} : \mathbb{R}^n \mapsto \mathbb{R}$, must be devised. By convention, the smaller $f_{\text{data}}(\boldsymbol{x})$ the closer are the model complex visibilities computed from the image $\boldsymbol{x}$ to the actual data, so $f_{\text{data}}(\boldsymbol{x})$ can be thought as a *distance* between the model and the data. In general and in order to account for the statistics of the noise, $f_{\text{data}}(\boldsymbol{x})$ is expressed from the likelihood of the data given the model image. To be more specific and assuming for the sake of simplicity that the complex visibilities have been measured, the distance between the model and the data could be given by:

$$f_{\text{data}}(\boldsymbol{x}) = \sum_k w_k \left| (\mathbf{H}\cdot\boldsymbol{x})_k - V_k \right|^2, \tag{1.1}$$

where $\mathbf{H}$ is a linear operator which computes the Fourier transform of the image at the spatial frequencies sampled by the observations, $(\mathbf{H}\cdot\boldsymbol{x})_k$ is the model of V_k the k-th measured complex visibility and $w_k > 0$ is a weight which depends on the accuracy of the measurement. As optical interferometers do not directly provide complex visibilities[2], actual data criteria have different expressions than (1.1) which may differ between algorithms (see for instance Meimon et al., 2005, for a convex approximation of the co-log-likelihood of interferometric data). Nevertheless, the simple criterion gives the idea of

[2] to get rid of turbulence effects, the powerspectrum, phases closures and differential phases are usually measured

the kind of data distance.

Since the data are corrupted by noise, an exact match of the measurements and of the model is unexpected. In fact, any image is acceptable provided that it yields model complex visibilities that differ from actual data by amounts consistent with the noise level. Using the metric $f_{\text{data}}(\boldsymbol{x})$, an image should be assumed to be compatible with the data whenever $f_{\text{data}}(\boldsymbol{x})$ is below some threshold, say:

$$f_{\text{data}}(\boldsymbol{x}) \leq \eta\,. \tag{1.2}$$

Clearly, for given sparse data, there are many, possibly an infinite number of, different images which satisfy this *discrepancy principle* and are compatible with the data[3].

To reduce the number of possibilities, the sought image can be restricted to belong to the set $\Omega \in \mathbb{R}^n$ of nonnegative and normalized images:

$$\Omega = \left\{\boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \boldsymbol{x} \geq \mathbb{0},\ \mathbb{1}^{\mathrm{t}} \cdot \boldsymbol{x} = 1\right\}, \tag{1.3}$$

where $\mathbb{0}$ and $\mathbb{1}$ are images whose pixels are all equal to zero and one respectively; thus $\boldsymbol{x} \geq \mathbb{0}$ taken componentwise expresses the nonnegativity of the pixel values while $\mathbb{1}^{\mathrm{t}} \cdot \boldsymbol{x} = \sum_i x_i$ is the sum of pixel values. These strict constraints (in particular the nonnegativity) are really helpful for image reconstruction from interferometric data but are yet insufficient to select a single image out of all the ones which are compatible with data, *i.e.* for which inequality (1.2) holds.

To have a unique solution, the image reconstruction problem has to be further constrained. Acknowledging that it is not possible to trustfully recover an image with many features from a limited amount of data, it is natural to impose that the image be *as simple as possible* (or as regular as possible) while being compatible with the observations. The most flexible way to account for such requirements is to introduce a criterion $f_{\text{prior}} : \mathbb{R}^n \mapsto \mathbb{R}$ such that the smaller $f_{\text{prior}}(\boldsymbol{x})$ the most simple or regular is the image $\boldsymbol{x}$. Then the image reconstruction amounts to "*finding the most simple image which is compatible with the observations*"; formally this writes:

$$\min_{\boldsymbol{x}\in\Omega} f_{\text{prior}}(\boldsymbol{x}) \quad \text{s.t.} \quad f_{\text{data}}(\boldsymbol{x}) \leq \eta\,. \tag{1.4}$$

The usual way to solve the constrained problem (1.4) is to use the associated Lagrangian:

$$\mathscr{L}(\boldsymbol{x}, \alpha) = f_{\text{prior}}(\boldsymbol{x}) + \alpha\, f_{\text{data}}(\boldsymbol{x})\,, \tag{1.5}$$

with $\alpha \geq 0$ the Lagrange multiplier associated with the constraint $f_{\text{data}}(\boldsymbol{x}) \leq \eta$. Technically, the multiplier α must be nonnegative because the constraint is an inequality (Nocedal and Wright, 2006). If $\alpha = 0$, the constraint has no incidence (it is said to be *inactive*) which, in our case, means that the data are completely ignored to determine the sought

[3] in the case of an ideal interferometer which provides the complex visibilities, it is sufficient that the model complex visibilities approximately match the measurements at the observed spatial frequencies, there are no constraints for all other frequencies

image. Obviously, we want to have $\alpha > 0$. As a consequence, the bound of the inequality should be exactly reached and $f_{\text{data}}(\boldsymbol{x}) = \eta$.

To summarize, the solution $\widehat{\boldsymbol{x}}$ of the image reconstruction is given by:

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\Omega}{\arg\min}\left\{\mathscr{L}(\boldsymbol{x},\alpha) = f_{\text{prior}}(\boldsymbol{x}) + \alpha\, f_{\text{data}}(\boldsymbol{x})\right\} \tag{1.6}$$

where $\alpha > 0$ is chosen such that $f_{\text{data}}(\widehat{\boldsymbol{x}}) = \eta$. Since $\alpha > 0$, taking $\mu = 1/\alpha$ yields the following equivalent formulation:

$$\widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\Omega}{\arg\min}\left\{f(\boldsymbol{x},\mu) = f_{\text{data}}(\boldsymbol{x}) + \mu\, f_{\text{prior}}(\boldsymbol{x})\right\}, \tag{1.7}$$

where $\mu > 0$ can be tuned so that the constraints hold.

Depending on the considered algorithm, one of these equivalent formulations is assumed and the tuning parameter (equivalently η, α or μ) is explicitly required or automatically tuned by the method.

2

IMAGE RECONSTRUCTION INTERFEROMETRIC DATA SETS

2.1. INTRODUCTION

In this chapter different tools and data sets, useful for interferometric image reconstruction are presented. In section 2.2 we start by introducing the publicly available software ASPRO2. This software allows the generation of synthetic interferometric data sets tailored to different instruments/observatories. The images can be generated from a few built-in functions or by importing a file. The interferometric data sets generated by ASPRO2 are very precise as they include, e.g., shadowing in the uv-space generations, as well as noise models of the different instruments. Then, we present the Jean-Marie Mariotti Center for Expertise in Interferometry (JMMC) and the International Astronomical Union (IAU) Interferometry Imaging Beauty Contests (section 2.3). Finally, in section 2.4, a library allowing the generation of synthetic images (which can then be uploaded to ASPRO2) is presented. This library complements the possibilities of ASPRO2 built-in functions.

2.2. GENERATING SYNTHETIC DATA SETS WITH ASPRO2

ASPRO2 (A Software to PRepare Observations) is a program developed at the Jean-Marie Mariotti Center for Expertise in Interferometry to allow users in the community to prepare interferometric observations by simulating the response of different interferometric arrays. ASPRO2 uses a simple Graphical User Interface (GUI) in which the user can introduce his/her observing constraints, for example:

- Users can search for their targets. The tool is connected to the SIMBAD database, so that the user can look for a desired target just by specifying a name.

- Users can define the observational setup. ASPRO2 allows to select an observing date, the instrument and interferometric configuration. Once the configuration is selected, users are able specify the integration time and the number of position hours ($u-v$ coverage) required.

With a given setup, ASPRO2 generates the amplitude and phase of the visibilities, the powerspectrum (squared visibilities) and the argument of the bispectrum (i.e., closure phases). In fact, the user is able to export the generated interferometric observables into standard OIFITS files that can be used for modeling and image reconstruction. For a complete information of the ASPRO2 capabilities please see: Duchene et al. (2004); Duvert et al. (2002); Mella and Duvert (2004).

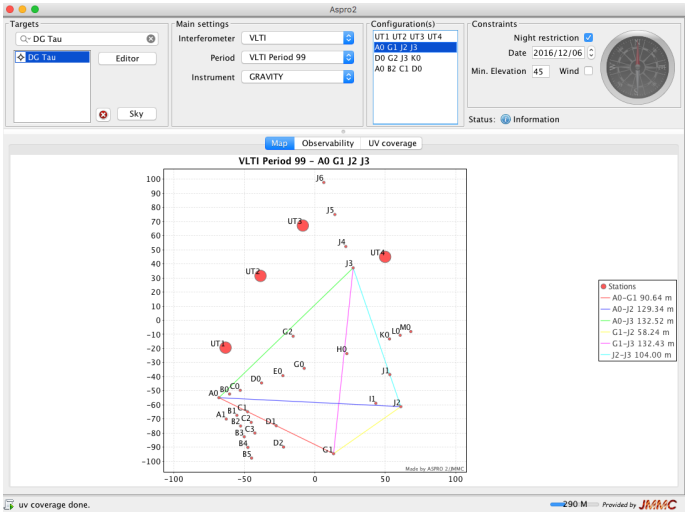

Figure 2.1: The figure displays the main ASPRO2 window. In the upper-left corner the user can specify the name of the target(s) to be analyzed. The upper-middle selection fields allow to define the instrumental configuration and observational ESO period. The upper-right panel serves to select the date of the observations and minimum elevation of the target. The lower section displays can show the interferometric array and baselines, the observability of the source over the night or the $u-v$ coverage.

2.2.1. MODELS FOR IMAGE RECONSTRUCTION

Inside the Editor menu of the ASPRO2 interface, users can corroborate and edit some of the main properties of the target, like the magnitude in different observing bands, the proper motions, coordinates, spectral type, etc. However, they can also define either a geometrical model or a user-defined model of the object's morphology. Figure 2.2 displays a screenshot of the target editor in ASPRO2. Highlighted in red, there is the subpanel in which the user can define a model for the selected target.

GEOMETRICAL MODEL

Inside the Model panel, the user can select over different components to simulate the target's morphology. Each one of them is composed by a given number of parameters that the user may adjust in order to cover his/her needs. For example, in Figure 2.3 we show the model of a source that is composed by two components: a central source (Gaussian) surrounded by an elongated ring. Both of them centered in the middle of the pixel grid. The Gaussian has a full-width-at-half-maximum of 3 mas. The ring has a minor internal diameter of 30 mas and a width of 8 mas. The elongation ratio is of 1.5 and both components have 50% of the flux in them. For more information about how to build models with the pre-defined geometrical functions, the user can consult the ASPRO2 user manual[1] accessible thought the Help panel in the Software. Once the target model is defined, the user can check the $u-v$ coverage and the expected interferometric observables in the ASPRO2 main window. Figure 2.5 shows the interferometric observables obtained with the aforementioned model for UT observations with GRAVITY in low-resolution mode (see Figure 2.4).

USER-DEFINED MODEL

As well as the geometrical model, ASPRO2 also allows the user to upload a pre-defined model. The model can be either (i) a monochromatic image or (ii) a set of images in a data cube. The images should be included in FITS files. The user can enable this option inside the Model menu, by selecting the option `User Model`. In order to read correctly the image(s) inside ASPRO2, the user must corroborate that the FITS header contains the keywords displayed in Figure 2.6. Notice that the CDELT1 should be negative it, by default, if the simulated image East was defined to the left. In the case the user uploads a data cube, CDELT3 defines the increment in wavelength for each one of the frames in the cube, CRVAL3 defines the wavelength at which the simulated bandpass begins and CUNIT3 sets the units in which CDELT3 and CRVAL3 are defined.

Once the data cube is upload, the user can check that everything is correct in the display panel inside the Model editor menu. If a data cube is uploaded, ASPRO2 animates the morphological changes of the source across the simulated band. Figure 2.7 displays an example of the ASPRO2 plot with the disk model used for the Beauty Contest 2016 was added. After the image was uploaded, similar as with the geometrical model, the user can adjust the desired $u-v$ coverage and the interferometric observables. Finally, if the simulation fulfills the desired requirements, the user can save it in a standard OIFITS file. To do this, the user has to select the File menu and then the `Export to OIFits file(s)` option.

[1] Also available here: http://www.jmmc.fr/twiki/bin/view/Jmmc/Software/JmmcAspro2

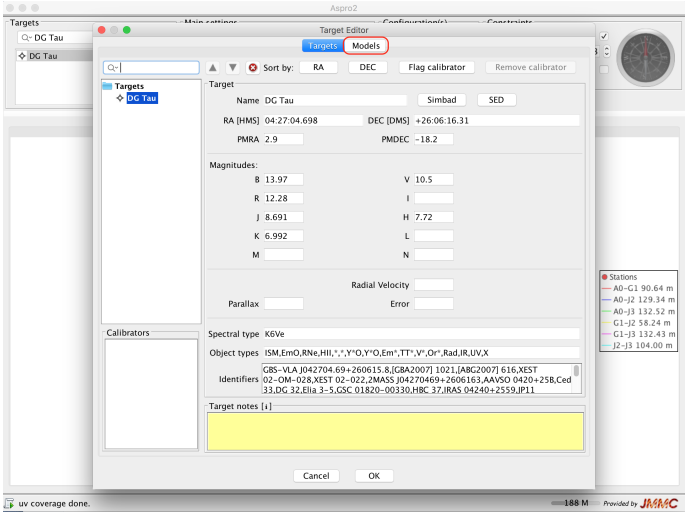

Figure 2.2: The figure displays the target editor menu inside ASPRO2.

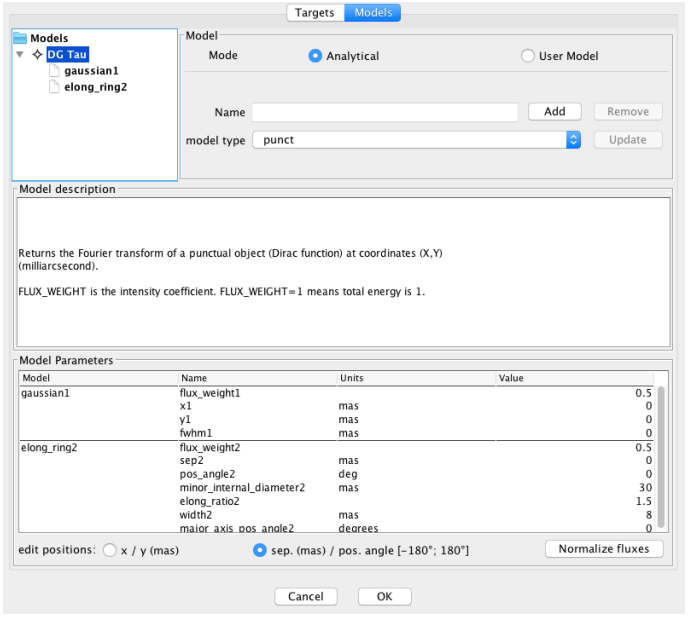

Figure 2.3: The figure displays the geometrical model menu of ASPRO2

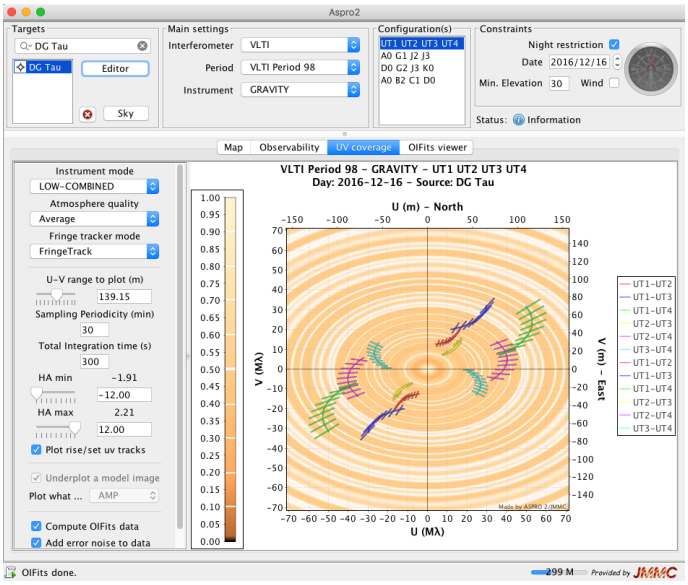

Figure 2.4: The figure displays the $u-v$ coverage display as well as the instrument setup panel of ASPRO2.

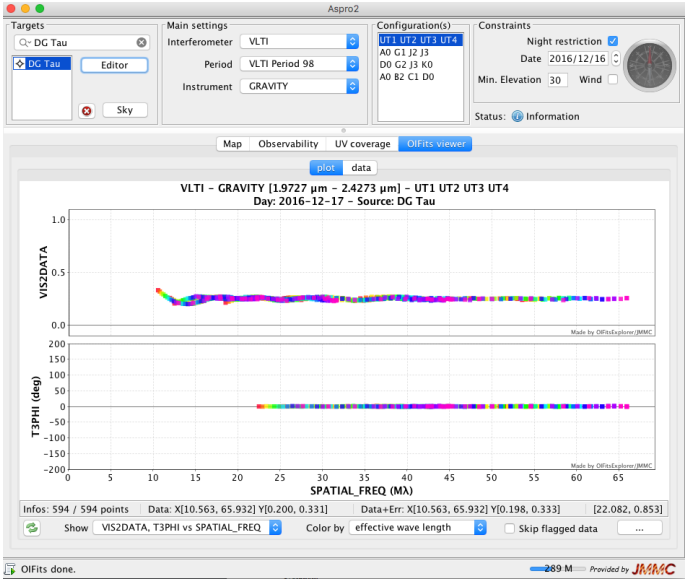

Figure 2.5: The figure displays the ASPRO2 plots with the simulated interferometric observables.

```
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                  -64 / array data type
NAXIS   =                    3 / number of array dimensions
NAXIS1  =                  420
NAXIS2  =                  420
NAXIS3  =                  250
CRPIX1  =                210.0
CRPIX2  =                210.0
CDELT1  =              -0.0001
CUNIT1  = 'ARCSEC   '
CDELT2  =               0.0001
CUNIT2  = 'ARCSEC   '
CRVAL1  =                  0.0
CRVAL2  =                  0.0
CRPIX3  =                    1
CDELT3  = 2.40963855421682E-09
CRVAL3  =              1.9E-06
CUNIT3  = 'METER    '
END
```

Figure 2.6: The figure displays the header keywords required to successfully upload a FITS data cube (with a user model) to ASPRO2.

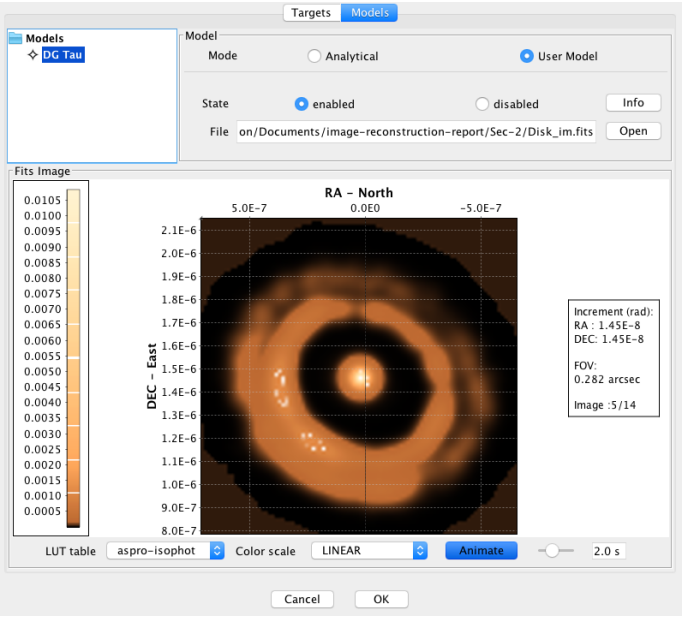

Figure 2.7: The figure displays the ASPRO2 panel with after a user-defined model has been uploaded using a FITS file.

2.3. THE JMMC AND BEAUTY IMAGE LIBRARIES

The page http://apps.jmmc.fr/oidata/ keeps several optical interferometric datasets, to be shared with the community. The main goal is make available a repository of real or simulated reduced data, in order to practice various tools, such as image reconstruction packages as the ones described in this cookbook. Included are the data used in past IAU Interferometry Imaging Beauty Contests (from 2004 to 2010). The maintainers of the repository are continuously accepting new entries and you should directly visit the aforementioned link for a list of available objects and details about them.

2.4. A YORICK SYNTHETIC IMAGE LIBRARY

In addition to the `Aspro2` built-in functions to generate astronomical source brightness distributions, there is a repository available at GitHub, named `YSIL` (Yorick Synthetic Image Library), with `Yorick` code that allows the creation of basic objects. This library can be used to generate astronomical synthetic sources, such as stellar clusters, young stellar objects and stellar photospheres (e.g., see Gomes et al., 2017, for some examples). The code can be found at https://github.com/NunonuN/ysil. It is open source under the GNU General Public License version 3, and it is still under development.

2.4.1. INSTALLATION

In order to use `YSIL`, it is necessary to install `Yorick` (see chapter 6, page 52 for details). Download `YSIL` to any desirable folder and include `ysil.i` in `Yorick`, as in, for example

```
include, "ysil.i";
```

2.4.2. EXAMPLES

Some examples of simple structures or objects created with `YSIL` are illustrated in fig. 2.8.

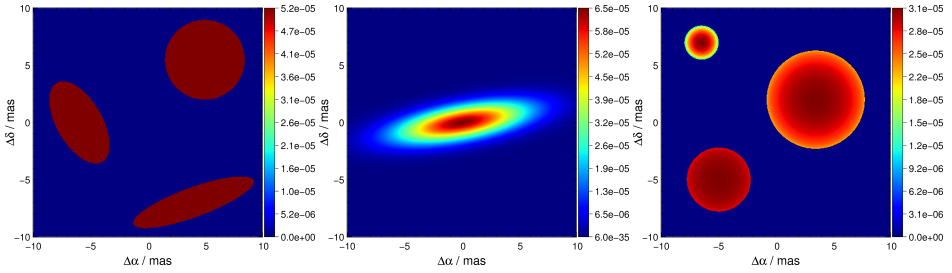

Figure 2.8: Three uniform discs (left), a disc with Gaussian profile (centre) and three stellar photospheres (right) created with the `YSIL` package, in a FOV of 20 mas.

The uniform discs were created with the following command:

```
x= span(-10, 10, 300)(, -:1:300);
discs= ysil_uniform_disc(x, [[2, 4], [3.5, 3.5], [5.6, 1.3]], [[-6, 0], [5, 5.5],
  [4, -7]], angs= [30, 0, -160]);
```


The Gaussian disc was obtained with

```
x= span(-5, 5, 500)(, -:1:500);
disc= ysil_gaussian_disc(x, [4., 2.], angs= 10);
```

and the stellar photospheres with

```
x= span(-5, 5, 400)(, -:1:400);
stars= ysil_limb_darkened_disc(x, [2.8, 4.3, 1.5], [[-5, -5], [3.5, 2], [-6.5,
 7]], u= [0.1, 0.3, 0.7], norm= TRUE);
```

3

IMAGE RECONSTRUCTION WITH SQUEEZE

3.1. INTRODUCTION

SQUEEZE is an open source (GPL v3) image reconstruction software developed by Fabien Baron[1] at Georgia State University. It uses a Monte-Carlo Markov Chain (MCMC) approach as main minimization engine. Two type of MCMC algorithms are included: Simulated Annealing and Parallel Tempering, both of them with Metropolis-Hasting moves. SQUEEZE includes a full polychromatic imaging, performing a simultaneous model-fitting to: (i) visibilities (amplitudes and phases), (ii) powerspectra, and (iii) bispectra, or combination of them. The code has the flexibility to consider the visibility phases as differential ones. It works either with `OIFITS V1.0` or `OIFITS V2.0` tables. The official description of the algorithm can be found in Baron et al. (2010), while the source code and its documentation are reachable at the following link: `https://github.com/fabienbaron/squeeze`.

Disclaimer

The current cookbook has been written from a user perspective. The main aim of the current document is to provide an guide of the main characteristics of SQUEEZE following practical examples. This document does not attempt to replace the official description and software documentation. If SQUEEZE is used or modified, the corresponding original papers and documentation should be always cited according to the established copyright.

[1] e-mail: baron@chara.gsu.edu

3.2. ALGORITHM DESCRIPTION

Due to the sparseness of the $u-v$ coverage, the poor calibration of the complex visibility amplitude and the lack of complete phase information, image reconstruction from interferometric data is an "ill-posed" inversion problem (Thiébaut, 2009, 2013; Thiébaut and Giovannelli, 2010). Therefore, to look for the best image that fits the data, "a-priori" information should be included on the reconstruction process. From a Bayesian inference approach, the best-fit image, $\boldsymbol{x}_{\rm ML}$, corresponds to the object's brightness distribution with the largest posterior probability $\Pr(\boldsymbol{x}|\boldsymbol{d},\boldsymbol{m})$:

$$\boldsymbol{x}_{\rm ML} = \underset{\boldsymbol{x}}{\rm argmax} \Pr(\boldsymbol{x}|\boldsymbol{d},\boldsymbol{m}) \tag{3.1}$$

where $\Pr(\boldsymbol{x}|\boldsymbol{d},\boldsymbol{m})$ is:

$$\Pr(\boldsymbol{x}|\boldsymbol{d},\boldsymbol{m}) \propto \Pr(\boldsymbol{x}|\boldsymbol{m})\Pr(\boldsymbol{d}|\boldsymbol{x},\boldsymbol{m}) \tag{3.2}$$

In the aforementioned expression, $\boldsymbol{d}$ corresponds to the data and $\boldsymbol{m}$ to the prior model. $\Pr(\boldsymbol{x}|\boldsymbol{m})$ is the probability of the image parameters given an "a priori" assumption of the true brightness distribution. $\Pr(\boldsymbol{x}|\boldsymbol{m})$ is assumed to be proportional to $e^{-\mu R(\boldsymbol{x})}$, where μ is the weight (hyperparameter) of the regularization function $R(\boldsymbol{x})$. $\Pr(\boldsymbol{d}|\boldsymbol{x},\boldsymbol{m})$ represents the likelihood of the data given the model image (i.e. the standard χ^2) and it is proportional to $e^{-1/2\chi^2(\boldsymbol{x})}$. Taking the negative log-probabilities from Eq. 3.1, we have:

$$\begin{aligned}
\boldsymbol{x}_{\rm ML} &= \underset{\boldsymbol{x}}{\rm argmin}\left[-\ln\left[\Pr(\boldsymbol{d}|\boldsymbol{x},\boldsymbol{m}) + \Pr(\boldsymbol{x}|\boldsymbol{m})\right]\right] \\
\boldsymbol{x}_{\rm ML} &= \underset{\boldsymbol{x}}{\rm argmin}\left[-\ln\left[e^{-1/2\chi^2(\boldsymbol{x})} + e^{-\mu R(\boldsymbol{x})}\right]\right] \\
\boldsymbol{x}_{\rm ML} &= \underset{\boldsymbol{x}}{\rm argmin}\left[1/2\chi^2(\boldsymbol{x}) + \mu R(\boldsymbol{x})\right]
\end{aligned} \tag{3.3}$$

SQUEEZE aims at finding $\boldsymbol{x}_{\rm ML}$ (as described in Eq. 3.3) using, as central engine, a Monte Carlo Marchov Chain minimization algorithm. The MCMC algorithm uses a Simulated Annealing (SA) implementation to look for the global minimum. SA reproduces the crystallization process of metals due to cooling or annealing. It simulates the motion of the particles at different energetic states. When the particles are in a a high-energy state, their individual motions do not vary so much from the rest of them. Nevertheless, when the system becomes cooler, the motion of the particles tend to minimize the energetic balance. The same behavior is reproduced using SA, for the first iterations, the search of the optimal values suggests drastic random modifications to the reconstructed image, but progressively keeps only those that improve the reconstructed solution. As described in Baron et al. (2010), the acceptance probability of a given image in the MCMC is based on the Boltzman/Metropolis formula:

$$p(j, j+1) = \min[1, J] \tag{3.4}$$

or the Glauber formula:

$$p(j, j+1) = \min\left[1, \frac{J}{J+1}\right] \tag{3.5}$$

where J is defined as:

$$J = \frac{R(\boldsymbol{x})_{j+1}}{R(\boldsymbol{x})_j}\left(\frac{\chi^2(\boldsymbol{x})_{j+1}}{\chi^2(\boldsymbol{x})_j}\right)^{T_j} \tag{3.6}$$

in the previous expression, T_j corresponds to the temperature at the j-th iteration. One of the main parameters critical for the algorithm's performance is the adjustment of the temperature. If this one decreases to fast, it is highly probable to be stuck in a local minima, on the other hand, if the temperature decreases too slow, the algorithm may require a lot of time to converge. In SQUEEZE, for the first iteration, the temperature is settled as:

$$T_0 = \chi^2_{r,0}(\boldsymbol{x})\frac{\gamma}{\mathrm{SF}} \tag{3.7}$$

where γ is the decay coefficient (convergence rate) and SF corresponds to an scaling factor. The decay coefficient is a user defined parameter that can be settled through the `-ct` option. The default value is 4.0. The temperature schedule for successive iterations changes as follows:

$$T_{j+1} = T_j + \frac{1}{\tau_j}(\chi^2_{r,j}(\boldsymbol{x}) - \gamma T_j)\left(1 - \frac{\chi^2_t(\boldsymbol{x})}{\chi^2_{r,j}(\boldsymbol{x})}\right) \tag{3.8}$$

where τ_j is the decay time of the temperature and $\chi^2_t(\boldsymbol{x})$ is the objective χ^2. This last parameter can be defined by the user with the `-fc` option. The $\chi^2_{r,j}(\boldsymbol{x})$ corresponds to the χ^2 of the reconstructed image at the j-th iteration (see also Ireland et al., 2006). SQUEEZE has two implemented type of steps: the first type transfers randomly the flux of an element to another; the second type interchanges the flux of two randomly selected elements. The transfer of flux to another element depends on the χ^2 value. When this is too large, the code favors flux transfer within pixels all over the pixel grid. Nevertheless, when the algorithm has begun to converge, the code favors flux transfer only among adjacent pixels or just the interchange of flux between two of them. For more information about the SQUEEZE algorithm please see Baron et al. (2010).

3.2.1. REGULARIZATION FUNCTIONS:

SQUEEZE allows the individual use or combinations of the following different regularization functions $R(\boldsymbol{x})$:

- **Entropy (user option: `-en`):** This regularizer ensures that the information contained in the image is the minimum one (see e.g., Baron and Young, 2008; Skilling and Bryan, 1984). The function implemented in SQUEEZE computes the sum of

the natural logarithm of the Gamma function for each one of the non-zero pixels in the image:

$$R(\boldsymbol{x}) = \sum \log_e |\, \Gamma(\boldsymbol{x}) \,|, \;\; \forall x \neq 0 \tag{3.9}$$

- **Dark Energy (user option: `-de`):** This regularizer counts all the zero elements in the image. The function implemented in SQUEEZE also add a value to the edges of the image. The general formula of this regularizer is the following one:

$$R(\boldsymbol{x}) = \sum \boldsymbol{x}, \;\; \forall x = 0 \tag{3.10}$$

- **Uniform disk (user option: `-ud`):** This regularizer corresponds to the squared sum of the Lp-norm, with p=0.5, of the local gradient. The SQUEEZE implementation ignore the image borders, and the general formula is described in Eq. 3.11.

$$\begin{aligned} || \nabla x ||_{l,s} &= \sqrt{(x_{l+1,s} - x_{l,s})^2 + (x_{l,s+1} - x_{l,s})^2} \\ R(\boldsymbol{x}) &= \left(\sum \sqrt{|| \nabla x ||_{l,s}}\right)^2 \end{aligned} \tag{3.11}$$

- **Total Variation (user option: `-tv`):** This regularization function quantifies the spatial gradient distribution of the images instead of the intensity distribution (Strong and Chan, 2003). For brightness distributions that tend to be piece-wise smooth, the gradient field is sparse, having large values where the image shows discontinuities. The SQUEEZE implementation ignore the image borders, and the general formula is described in Eq. 3.12.

$$\begin{aligned} || \nabla x ||_{l,s} &= \sqrt{(x_{l+1,s} - x_{l,s})^2 + (x_{l,s+1} - x_{l,s})^2} \\ R(\boldsymbol{x}) &= \sum || \nabla x ||_{l,s} \end{aligned} \tag{3.12}$$

- **L0 sparsity norm (user option: `-l0`):** This regularizer accounts for the total number of non-zero pixels in the image. If an image is reconstructed subject to this function, it will be forced to be the sparsest solution. In contrast with other software SQUEEZE can optimize non-convex functions like the L0-norm. The mathematical expression of this regularizer is the following one:

$$\begin{aligned} \mathrm{L0} &= 1, \;\; \forall x > 0 \\ R(\boldsymbol{x}) &= \sum \mathrm{L0} \end{aligned} \tag{3.13}$$

- **L2 norm (user option: `-l2`):** This function is part of the Lp-norm family of relations that smooth the reconstructed image by accounting for the total of the squared value of the pixels in a given grid (Tikhonov and Arsenin, 1977). The mathematical expression of this regularizer is the following one:

$$R(\boldsymbol{x}) = \sum x^2 \tag{3.14}$$

- **Transpectral L2 regularization (user option: `-ts`):** This regularization function is important when a polychromatic reconstruction is done. The function computes the L2-norm of the pixels across the different spectral channels in the data sets. The mathematical expression of this regularizer is the following one:

$$R(\boldsymbol{x}) = \sum_n \sqrt{\sum_l \sum_s (x_{l,s}^{\lambda 0})^2 + (x_{l,s}^{\lambda 1})^2 + (x_{l,s}^{\lambda 2})^2 ... (x_{l,s}^{\lambda n})^2} \tag{3.15}$$

- **Field of view (user option: `-fv`):** As a default parameter, SQUEEZE moves the image to the centroid position of the frame.
- **Prior image (user option: `-p`):** SQUEEZE also can use a pre-defined model image as regularizer. If this option is selected, the code computes the sum of the negative logarithm of the non-zero pixels in the prior image, $\boldsymbol{L}$:

$$R(\boldsymbol{x}) = \sum_n -\log \boldsymbol{L} \tag{3.16}$$

- **Special Regularizers (user options: `-l0CDF53, -l1CDF53, -l0CDF97, -l1CDF97, -l0ATROUS, -l1ATROUS`):** SQUEEZE has a series of regularizers based on the theory of compressed sensing. The code is able to retrieve the L1- or the L0-norm of the decomposition coefficients of the image on a selected compression basis. The code has implemented the CDF 5/3, CDF 9/7 and "a-trois" wavelet transformations.

3.3. INSTALLATION AND CONFIGURATION

SQUEEZE is a stand-alone package that can be retrieved from the following GitHub repository: `https://github.com/fabienbaron/squeeze`. The code is written in C and it is compatible with OpenMP. The code requires `gcc` to be compiled. This compiler is native in most of the Linux distributions. Nevertheless, it has to be installed by the user in OSX. The GitHub repository contains all the necessary instructions to install these compiler. However, the most reliable ways to do it on an OSX system is through the `macports` [2] or the `Homebrew`[3] environments. According to the source repository, once the compiler and dependencies are satisfied, the SQUEEZE installation is relatively simple, a sequence of the following commands has to be executed:

- Download the current version of the code from the host repository or use the following `git` command:

```
git clone https://github.com/fabienbaron/squeeze.git
```

- Go inside the SQUEEZE main directory and update the OIFITSLIB module with the following command:

[2] `https://www.macports.org/`
[3] `http://brew.sh/index_es.html`

```
git submodule update --init
```

- Inside the main SQUEEZE directory use the following `cmake` commands to build the executable:

```
cd squeeze/build
cmake ..
make
```

- Once the executable is created the user can add it to the global path. For a bash shell use: `export PATH=/path/to/SQUEEZE main dir/bin $PATH` For a tsch shell use: `setenv PATH $PATH:/path/to/SQUEEZE main dir/bin`

If everything went well, the user should be able to execute squeeze from a terminal session. A simple test to identify any problem in the installation is to type the following command in the prompt:

```
squeeze -h
```

This command should list all the SQUEEZE options available to the user. These options are divided into six main categories: (i) Main image settings, (ii) Output settings, (iii) Simultaneous model fitting settings, (iv) Oifits import settings, (v) Regularization & init settings and (vi) Convergence settings.

3.4. BASIC USAGE

One of the main advantages of SQUEEZE versus other reconstruction packages is the large number of options and parameters that can be used. These characteristics offer a large flexibility for the reconstruction, but it also requires more training from the user to master all the different options. In this cookbook, some examples of the basic code usage are included, as well as more advanced ones. **Nevertheless, the user has to be aware that image reconstruction is not a straight forward process and may require a deep understanding of the different parameters involved.**

3.4.1. EXAMPLE 1

For the first example, the model of LkHa 101 used for the 2004 Beauty Contest (Lawson et al., 2004) has been selected. The data set was built assuming observations with the six-station Navy Prototype Optical Interferometer (NPOI). Figure 3.1 displays the true image used to simulate the data set as well as the simulated $u-v$ coverage. The data set consisted of 15 baselines tracked at 15 different hour-angles. For this example, the `OIFITS` file only consisted of squared visibilities and closure phases.

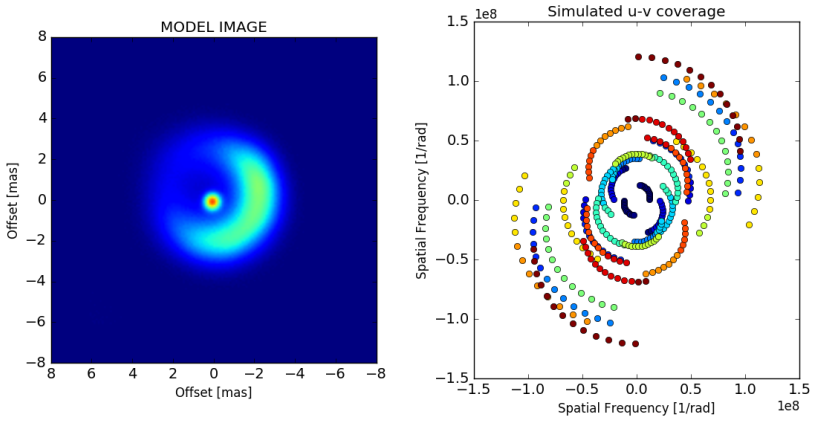

Figure 3.1: The **left:** panel displays the model image of LkHa 101 used to simulate the NPOI visibilities. The **right** panel displays the $u-v$ coverage of the interferometric data. Each different color indicates a different baseline.

3.4.2. RECONSTRUCTION WITHOUT REGULARIZATION:

To begin with a reconstruction, the pixel grid and pixel size should be defined by the user. As a consequence of mapping the interferometric data into a discrete grid of pixels, the user should be aware of avoiding field-of-view aliasing, at the time the field-of-view is chosen large enough to sample all the measured spatial frequencies in the data. As a rule of thumb, the pixel size of the reconstructed image should be of the order of:

$$\Delta\theta \leq \frac{\lambda}{4\mathrm{B}_{\mathrm{max}}} \tag{3.17}$$

where $\mathrm{B}_{\mathrm{max}}$ corresponds to the maximum baseline length in the interferometric array. In this case, the wavelength, λ corresponds to 0.55μm and the maximum baseline length corresponds to 66.4 meters. Therefore, the pixel size of the reconstructed image should be of the order of $\Delta\theta \sim 0.4$ mas. The following SQUEEZE command allows the reconstruction of an image of 128×128 pixels with the derived pixel scale using the data set called `example1.oifits`. In this case, no regularization functions were selected and only one MCMC chain with 500 iterations (default option) is used:

```
squeeze example1.oifits -w 128 -s 0.4
```

Figure 3.2 displays the reconstructed output images from SQUEEZE. Three panels are observed, which corresponds to the mean, mode and median images obtained from the iterations after the MCMC burn-in phase. In this case, the mean image was computed only with the best 35% of the frames (i.e., with the lowest χ^2) in the MCMC chain. If the algorithm does not converge (i.e., it does not pass the burn-in phase in a given number of

iterations), SQUEEZE still returns the aforementioned three images but they might have a large number of artifacts, particularly "image ghosts" due to the misaligned frames at different chain iterations.

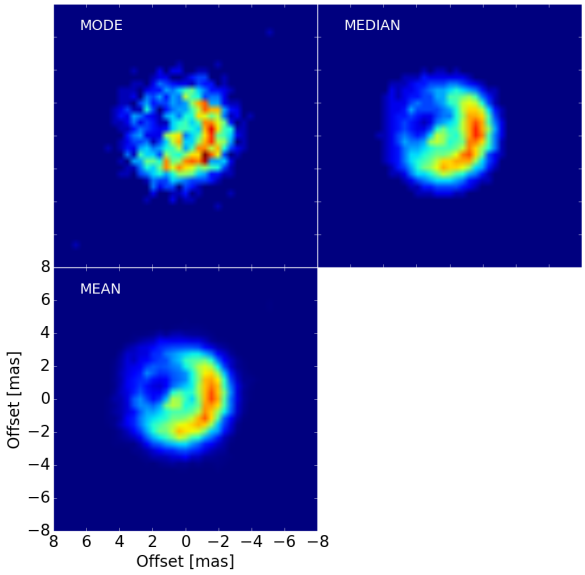

Figure 3.2: The figure displays three panels that correspond to the reconstructed images delivered by SQUEEZE after a given number of iterations for one MCMC chain. The color scheme is the same in all the frames, therefore the user can identify the differences between them.

The SQUEEZE output images are standard `.fits` files and the user could use his/her prefered software to inspect them. Additionally to these files, SQUEEZE can deliver the changes in the posterior probability at each one of the iterations in the chain. To habilitate this option, the user could run the following command as an example:

```
squeeze example1.oifits -w 128 -s 0.4 -fullchain
```

The results of the probability changes are thus saved in a file called `output.fullprobs`. This file contains the following information:

- The number of chains and the number of iterations
- The objective temperature[4]
- The logarithmic likelihood (χ^2), the prior probability and the posterior probability for each one of the iterations in the different chains.

[4] If the user runs several simultaneous chains, the total number of chains should be divided by three and this will be the number of lines that will contain the objective temperature before the posterior probability information.

Figure 3.3 displays the changes of the likelihood (in log space), prior probability and posterior probability for the previous example. Notice that, although the reconstruction did not use a prior function, the code produces a prior probability because by default SQUEEZE uses the **field of view** regularizer to move the image to its centroid position at each one of the iterations in the MCMC. In the previous example, it is clear that the effect of the prior probability is almost negligible once the algorithm has converged. Therefore, the posterior probability is mainly affected by the likelihood (i.e., the difference between the interferometric observables and the model). Visualizing the `.fullprobs` file is particularly interesting to identify whether the algorithm is properly converging or not. Additionally to the `.fullprobs` file, SQUEEZE also generates a file that contains the reconstructed image at each iterations of the MCMC and it is specially useful to make a statistical analysis of the frames selected for the final image. This file is usually named `output_fullchain.fits.gz`.

3.4.3. RECONSTRUCTION WITH REGULARIZATION:

Now that the basic reconstruction with SQUEEZE has been described, it is time to test the reconstruction using some regularization functions. Selecting a regularizer is not an easy task and strongly depends on the "prior" knowledge that the user has of the source's brightness distribution. There would be targets for which the structure is well-known (e.g., a binary) or that could be inferred from a parametric model (e.g., a disk or gaussian envelope), but there would be others more difficult to deduce. SQUEEZE allows to use several regularization functions simultaneously. Nevertheless, the more regularizers are used, the more difficult is to estimate the optimal values of the hyperparameters. To reconstruct the image with a given regularization, the user should check the corresponding option in SQUEEZE. For example, the following commands will serve to reconstruct the image with the associated regularizers:

- Reconstruction with **total variation** and an hyperparameter value, μ_{TV}, of 100:

```
squeeze example1.oifits -w 128 -s 0.4 -tv 10
```

- Reconstruction with **uniform disk** and an hyperparameter value, μ_{UD}, of 10:

```
squeeze example1.oifits -w 128 -s 0.4 -ud 10
```

- Reconstruction with **L0-norm** and an hyperparameter value, μ_{L0}, of 10:

```
squeeze example1.oifits -w 128 -s 0.4 -l0 10
```

Figure 3.4 displays the reconstructed images with the previous parameters and Figure 3.5 displays the posterior probability for each one of the iterations. Notice how, although the probability converged for the three images, the recovered image distribution is quite different among the reconstructions. **Therefore, caution should be taken by the user, when a regularization is selected.**

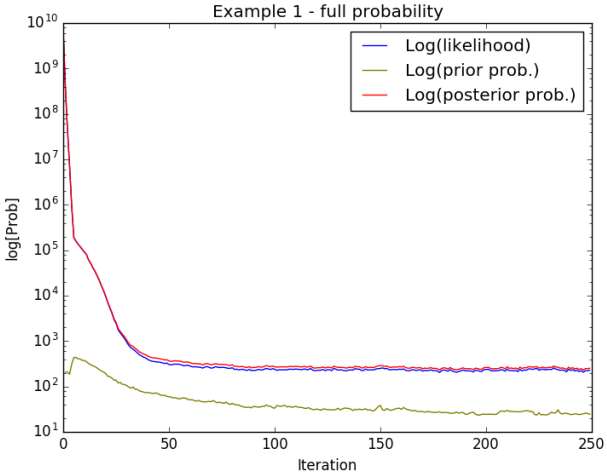

Figure 3.3: The figure displays a plot of the probability (in log space) changes for each one of the iterations in the MCMC. The likelihood, prior probability and posterior probability are displayed with different colors (see label in the image). Notice how after iteration number 50, the value of the posterior probability is quasi-constant around ~250, which corresponds to a total reduced χ^2_r ~1.04, indicating that the code has properly converged.

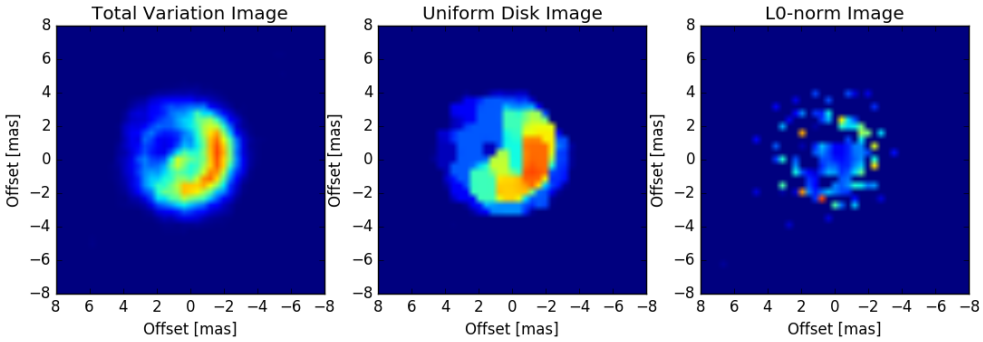

Figure 3.4: The image displays three panels that correspond to the reconstructed images with three different regularizers (see labels in the image). The color scheme is the same among them.

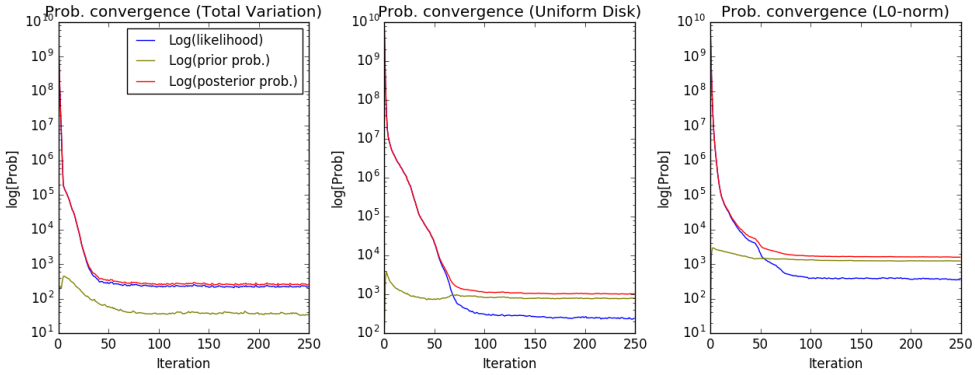

Figure 3.5: The image displays three panels that correspond to the posterior probability convergence with three different regularizers (see labels in the image).

SELECTION OF THE HYPERPARAMETERS μ_i

The hyperparameters μ_i control the trade-off between the χ^2 and the prior information of the brightness distribution encoded in the regularizers $R_i(x)$. Therefore, selecting the appropriate values of μ_i is crucial for the image reconstruction process. One of the most common methods to select the optimum value of a given μ is the L-curve. It computes the image solution for several values of μ, characterizing the response of the prior term versus the χ^2. This relation shows an "L-shaped" curve (Bose et al., 2001; Hansen, 1992). For regions along the curve where the data fidelity term changes rapidly compared with the prior, the reconstructed image is in the over-regularized region. However, if the opposite is observed, the image is in the under-regularized region. The optimum value of μ is thus associated to the elbow of the L-curve.

Therefore, to select the optimum regularization, the user must have to conduct a search over several values of the hyperparameter. Figure 3.6 displays, as an example, the reconstructed image using the uniform disk regularization with different values of μ. Notice the change in the object morphology from the under-regularized ($\mu < 5$) to the over-regularized region ($\mu > 5$). Sometimes the L-curve exhibits more than one elbow. In cases like this, a rule of thumb is to use the optimal value that corresponds to the inflection point with the smallest χ^2. Figure 3.7 displays the L-curve for the reconstructions presented previously.

SQUEEZE WITH SEVERAL CHAINS

One of the advantages of the MCMC implementation in SQUEEZE, is the possibility to perform the reconstructions using several simultaneous chains. This option is particularly important to test the outcome of the reconstruction, even if the code has converged. Analyzing the outcomes of several chains help to identify artifacts in the image and to make statistics about the most probable brightness distribution of the target. The `-chains` option uses a parallel implementation to compute simultaneously the recon-

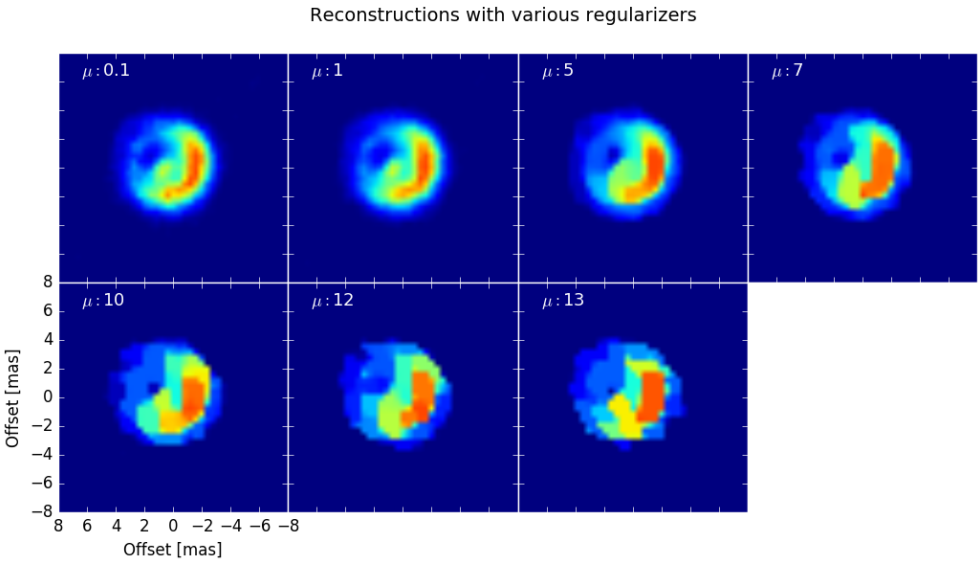

Figure 3.6: The image shows the impact of μ on the image reconstruction process. Seven reconstructed images of LkHa 101 are shown using different hyperparameter values. Notice the changes in the reconstructed image depending on the μ value.

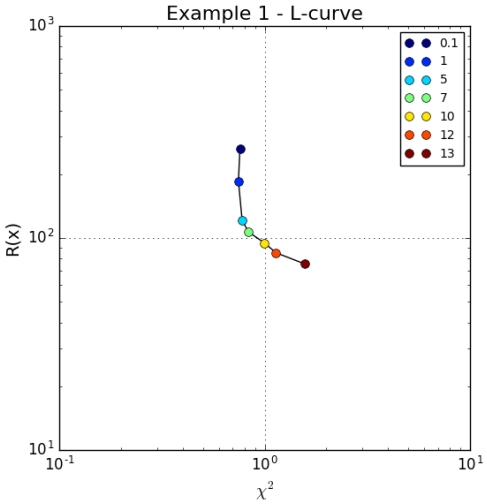

Figure 3.7: L-curve obtained with different values of μ using the Uniform Disk regularizer. The vertical axis displays the value of R(x) and the horizontal axis the χ^2. The optimal value of μ is at the elbow of the curve with the smallest χ^2, in this case it is close to 5.0.

structions. The user should check what is the maximum number of chains that can be set according to the available computer facilities. As an example to habilitate this option with 50 simultaneous chains, the user could run the following command:

```
squeeze example1.oifits -w 128 -s 0.4 -tv 10 -chains 50
```

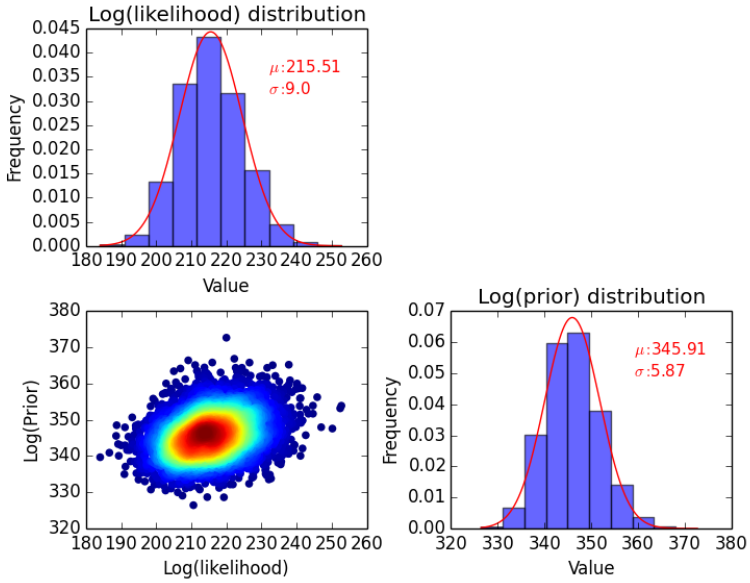

Figure 3.8: The Figure shows individual likelihood and prior distributions, as well as the joined distribution of 50 chains with 100 iterations.

Figure 3.8 displays the distributions of the likelihood and prior probability for the last 100 iterations of 50 chains. Notice how both correspond to a normal distribution. The joint distribution shows also how the different chains converged to a given value. Therefore, in order to determine what is the "best image" (the most probable one), the user has to analyze the distributions and chain outputs to decide which of them pick to create the final image. For example, in Figure 3.9 the mean "final image" created with the best 10 chains (i.e., with the smallest χ^2) is presented together with an example of the image fit to the observed closure phases and squared visibilities.

Up to now, the reconstructions have been made using using the same initial brightness distribution for all the chains. By default, SQUEEZE uses a delta function centered at the middle of the pixel grid. Nevertheless, it would be particularly important to test the reconstruction with different initial brightness distribution, specially for different chains. To do this, the user could habilitate the following options:

- Initialize the reconstruction with a random image common to all the chains:

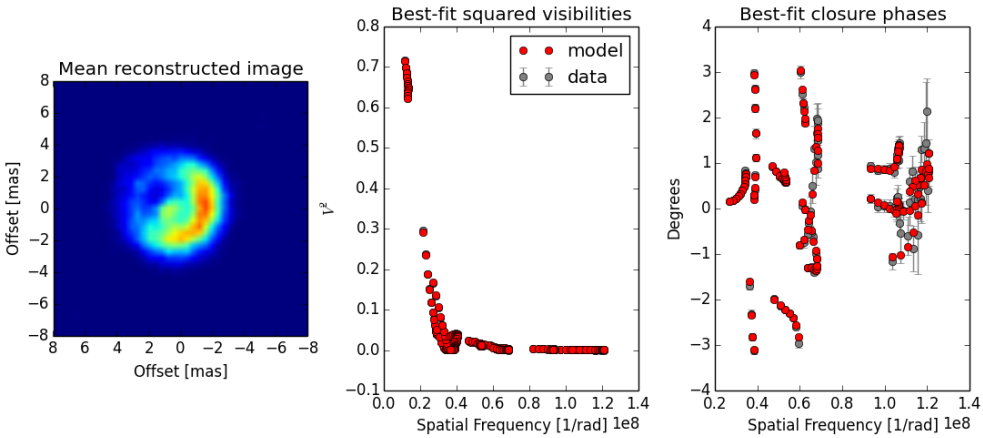

Figure 3.9: **left:** Best reconstructed image. **center:** Best-fit image squared visibilities. **right:** Best-fit image closure phases.

```
squeeze example1.oifits -w 128 -s 0.4 -tv 10 -chains 50 -i random
```

- Initialize the reconstruction with a random image different for all the chains:

```
squeeze example1.oifits -w 128 -s 0.4 -tv 10 -chains 50 -i randomthr
```

- Initialize the reconstruction with a user-defined image `initial_im.fits`:

```
squeeze example1.oifits -w 128 -s 0.4 -tv 10 -chains 50 -i initial_im.fits
```

3.4.4. EXAMPLE 2

For this example, the second data set provided for the 2004 Beauty Contest edition is used. Figure 3.10 displays the $u-v$ coverage, as well as the squared visibilities and closure phases for this data set. According to Lawson et al. (2004), this data set consisted of a central diffuse source together with a secondary compact source. The data used in this example not only contains closure phases and squared visibilities, but also visibility amplitudes, phases and triple amplitudes. In this respect, SQUEEZE allows the user to select which observables fit during the reconstruction. To disable observables from the minimization process, the following options could be selected: **`-novis`, `-novisamp`, `-novisphi`, `-not3`, `-not3amp`, `-not3phi` and `-nov2`**. The user should check the quality of the observables to decide whether use all of them or not. For example, in this case, performing the reconstruction only with closure phases and squared visibilities makes difficult the convergence of SQUEEZE, and the minimum temperature reached is of ~7.0 with a χ^2 in the squared visibilities of ~3700. The user can corroborate this, by running the following command:

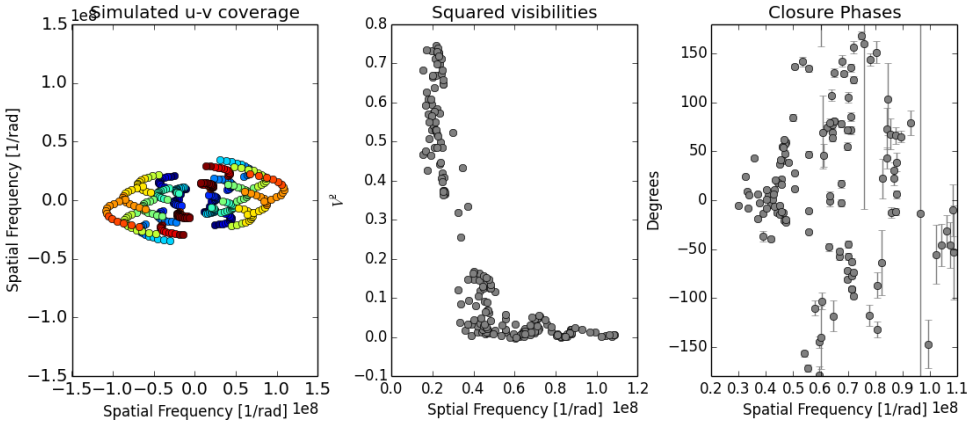

Figure 3.10: **left:** $u-v$ coverage. The colors indicate different baselines. **center:** Squared visibilities. **right:** Closure phases

```
squeeze 2004dataset2.oifits -w 128 -s 0.4 -tv 100 -novisamp -novisphi -not3amp
```

In contrast with the previous attempt, using the visibility amplitude and closure phases the algorithm converges reaching an annealing temperature of 1.0 and a χ^2 of ~3.6. The user can check this reconstruction by running the following command:

```
squeeze 2004dataset2.oifits -w 128 -s 0.4 -tv 100 -nov2 -novisphi -not3amp
```

Figure 3.11 displays the reconstructed images with the previous two commands. Please notice the clear difference in the recovered brightness distribution. The left panel displays the reconstruction that converged properly and the structure of the source (an extended elongated shape + a compact object) is revealed. The right panel displays the non-converged reconstruction.

3.5. ADVANCED USAGE

3.5.1. POLYCHROMATIC RECONSTRUCTION

One of the main goals of the new generation of infrared interferometers is to recover the morphological changes of the astrophysical objects across the bandpass of the observations. For example, this aspect is particularly important for MATISSE (Lopez et al., 2008, 2009), which will have a bandpass as large as $\Delta\lambda$ ~5 μm in the N-band. SQUEEZE allows to perform polychromatic reconstruction by including the differential phase information of the data. The initial setup required to perform this reconstruction is quite similar to the ones used in the previous examples. Nevertheless, this time, the differential phases are included. SQUEEZE has an integrated a transpectral regularizer available

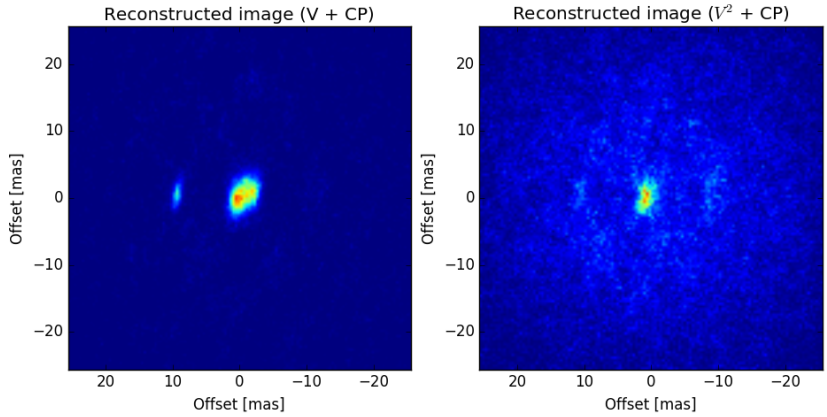

Figure 3.11: **left:** Reconstructed image with visibility amplitudes and closure phases. **right:** Reconstructed image with squared visibilities and closure phases

through the $-ts$ option. Since closure phase is a shift-invariant observable, transpectral regularization is required to ensure spectral continuity across the bandpass used for the reconstructions. This is particularly important to avoid different positions of the recovered object at each one of the frames in the reconstructed data cube.

3.5.2. THE OBJECT

Proto-planetary disks is one of the key science objectives for the second generation of VLTI instruments. Therefore, the object selected to illustrate a polychromatic reconstruction consisted of a simulated accretion disk as observed in the N-band (8-13 μm) with the MATISSE instrument in low-resolution mode. The model includes a central star (surrounded by a large amount of dust) and a disk containing one gap, supposed to be carved by a forming planet (Gonzalez et al., 2015). The brightness distribution across the disk is not uniform with a clear bright spot at a position angle of ~220° East of North and the disk shape is elongated in the East-West direction. The observational setup for this data set includes three simulated nights using three different configurations of the Auxiliary Telescopes. Figure 3.12 displays the simulated $u-v$ coverage, as well as the interferometric observables. Notice how the squared visibilities trace a resolved object with an angular size of ~100 mas. The simulated closure phases are considerably noisier, which increases the difficulty of the reconstruction. The differential phases use the AMBER convention in which the reference channel corresponds to an average of all the channels except the working one. This means that the reference channel varies across the bandpass. The data set used was part of the 2016 Beauty Contest (Sanchez-Bermudez et al., 2016) and can be found here: `www.opticalinterferometry.com`

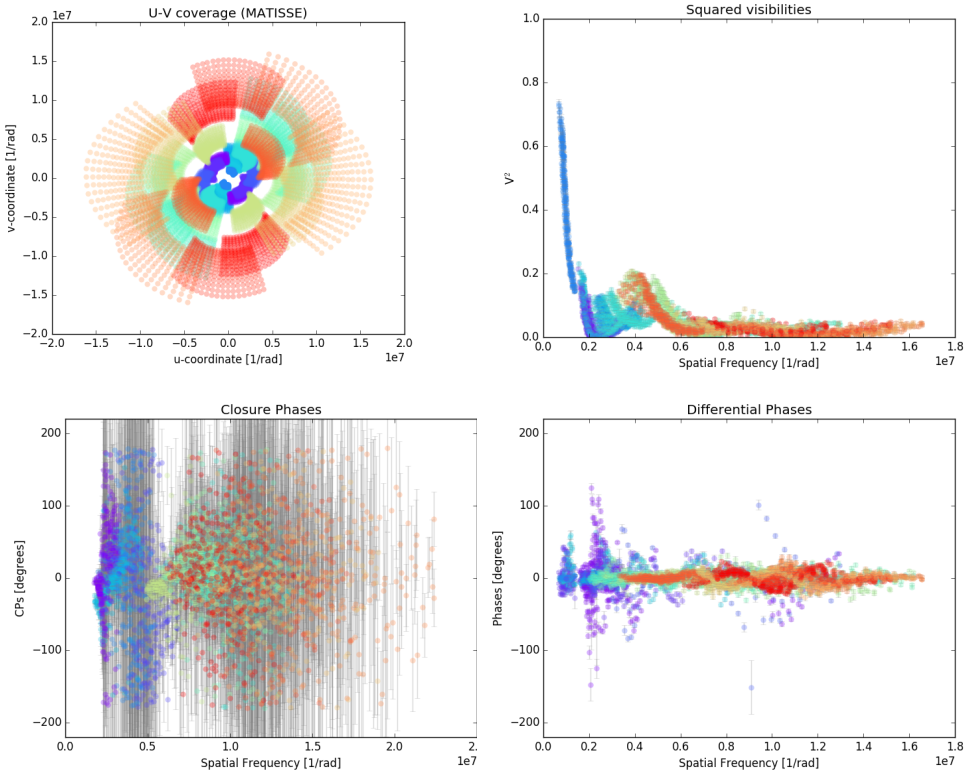

Figure 3.12: The **upper-left** panel shows to the simulated $u-v$ coverage for the MATISSE data. The **upper-right**, **lower-left** and **lower-right** panels display the simulated closure phases, squared visibilities and differential phases respectively. Different colors correspond to different baselines or triangles.

3.5.3. THE RECONSTRUCTION

To begin with this reconstruction, an initial brightness distribution of a Gaussian was used. To include a pre-defined image, in SQUEEZE the user can use the `-i` option. In this case, the data channel with the longest wavelength (from 12.8076 to 13.20 μm) and higher signal to noise was used. For this purpose the `-wavchan` was enabled. With this setup, a first monochromatic reconstruction was performed with the following command:


```
squeeze 2016dataset2.oifits -w 135 -s 3.0 -la 1000 -l0 1 -i initial.fits -novisamp -
    not3amp -novisphi -v2s 2.0 -wavchan 0 1.26e-5 13.10e-6 -n 150 -chains 3
```

In this example, a Laplacian regularization μ_{Lap}=1000 was used. Please notice that the aforementioned value was obtained by using the L-curve described in Sec. 3.4.3. In the command, we have also disabled the use of the visibility amplitudes (not included in the data set), the triple amplitudes (also not included in the data) and the visibility phases. To ensure convergence, the additional option `-v2s` was included. This option multiply the errorbar of the squared visibilities by a factor of two. It is important to highlight, that the user should decide whether this option is necessary for a given reconstructed data set because it relaxes the convergence criteria of the algorithm. Finally, three chains were created for the reconstruction. The Figure 3.13 displays the initial image and the result of the monochromatic reconstruction.

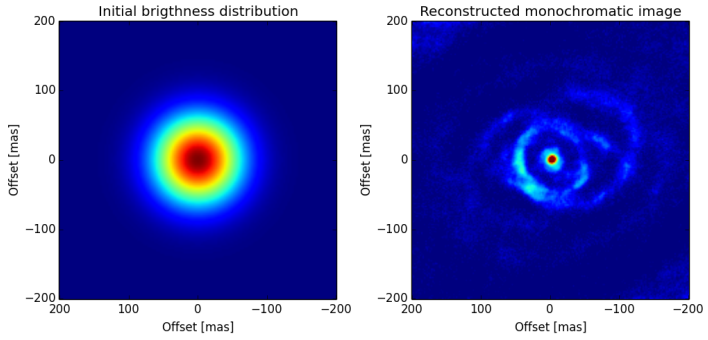

Figure 3.13: **left:** Initial image used for the monochromatic reconstruction. **right:** Best monochromatic reconstructed image. Notice how in the reconstructed image a rim with an asymmetric brightness distribution is observed around a central source.

Once the monochromatic reconstruction was finished. The recovered image was used as starting point to perform a full polychromatic reconstruction. The two regularizers used in the monochromatic reconstruction were included with an additional transpectral regularizer. To enable the reconstruction using all the channels in the data set, the `-wavauto` was used. Contrary to the monochromatic reconstruction, here the

differential phases were included. The complete command for the polychromatic reconstruction is the following:

```
squeeze 2016dataset2.oifits -w 135 -s 3.0 -la 1000 -l0 1 -ts 3 - fv 0 -i
    initial_monochromatic.fits -novisamp -not3amp -novisphi -wavauto -diffvis -n 150
     -chains 3
```

Figure 3.14 displays the recovered images channel by channel, as well as the images of the model used to simulate the data. Notice how the asymmetry of the disk is recovered, together with the changes in size and brightness of the central source. However, the elongation of the disk and the internal gap were not recovered in the reconstruction.

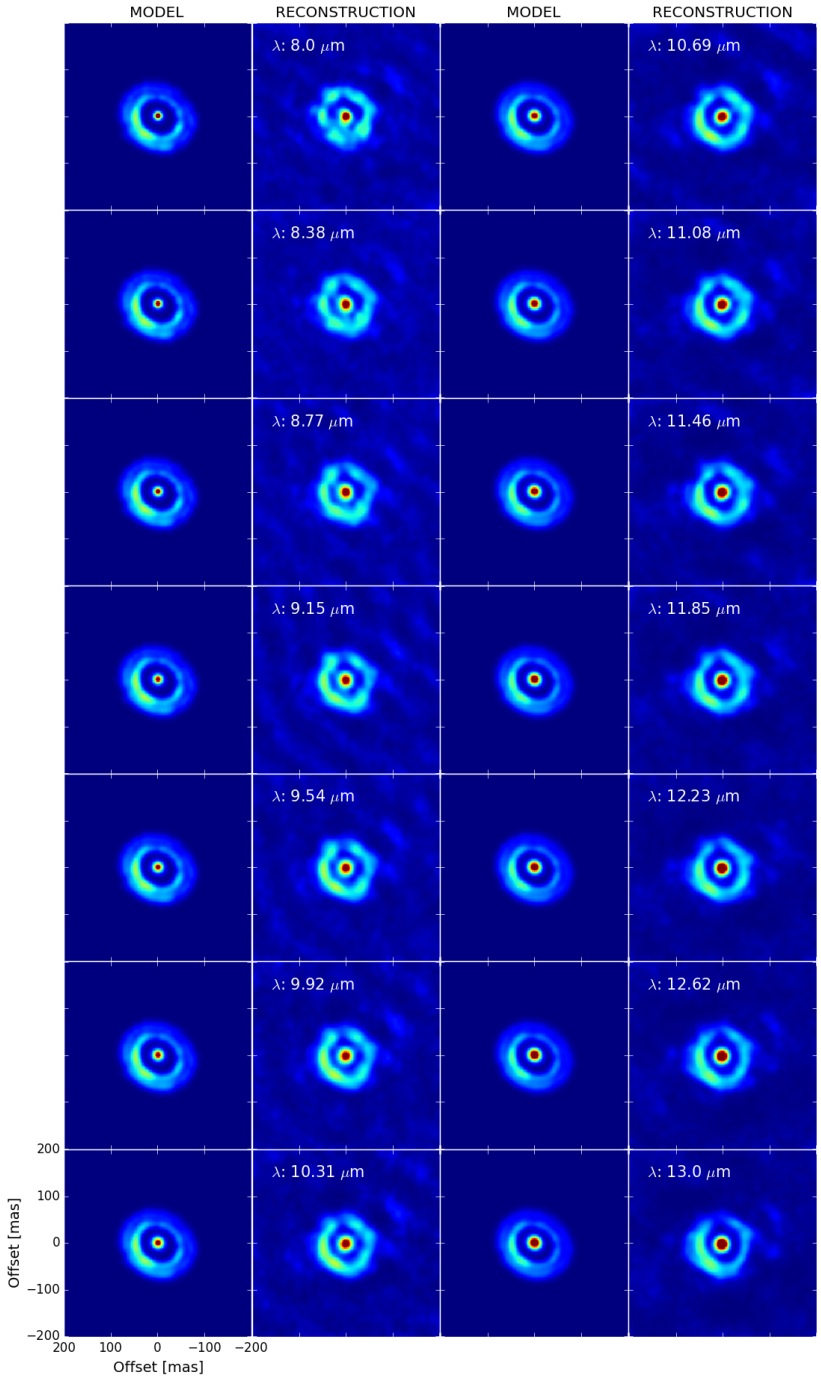

Figure 3.14: Fourteen reconstructed images across the N-band using the SQUEEZE polychromatic reconstruction. The respective channel is labeled in each frame. The model images (from which the observables were obtained) are also displayed.

4

THE IMAGE RECONSTRUCTION GRAPHICAL USER INTERFACE

4.1. INTRODUCTION

This section describes the graphical user interface that is the central part to operate image reconstruction softwares.

4.1.1. ARCHITECTURE

It is fine by design to separate the graphical user interfaces from the processing. By this way each blocks are specialized to its limited domain and work by cooperation. The strong advantage to put clear interfaces between each of them help to exchange some part or have choice to compare and run multiple algorithms on same input data and parameters. We then decided to provide a common Graphical User Interface (GUI) that can operates many softwares that respect the interface [1].

The execution architecture of the GUI can handle local executions or remote ones. Remote execution is performed over a standardized execution framework (UWS [2]) which avoid installation of local software to the user. Softwares then are hosted on computing servers. Tests have been performed but first version requires the user to install image reconstruction software on his machine.

[1] https://github.com/emmt/OI-Imaging-JRA
[2] http://www.ivoa.net/documents/UWS/index.html

4.2. INSTALLATION AND CONFIGURATION

4.2.1. REQUIREMENTS

The GUI is packaged as a JavaWebStart [3] application then requires a Java runtime environment. The required command is `javaws`.

Java runs on every operating systems with a great compatibility and efficiency. Users who don't have java on their machine yet, just have to install the java package of their distribution or may also visit http://www.java.com. The website provides support in many languages to install the runtime environment.

Image reconstruction softwares must be installed previously and present in the PATH of the GUI. Look at Help Menu / Report Feedback to JMMC ... / Details / System Properties.

Enhancement are planned to help the GUI finding some softwares.

4.3. BASIC USAGE

The work-flow consists of selecting the input data (`OIFITS`), a starting image (`FITS`) and the image reconstruction software with associated parameters settings. Then the user presses the run button so the algorithm runs until reaching the stopping criterion or maximum number of iterations. Complete runs end providing result data : image, `OIFITS` and associated output parameters. An execution log is shown and may help the user for any faulty run.

4.3.1. INPUT DATA

INTERFEROMETRIC DATA

The GUI load interferometric data through `OIFITS` files. The user can load files from its local disk or may use SAMP protocol to load data from the OiDB portal [4]. `OIFITS V2` is not yet supported by the first version of the GUI.

IMAGE DATA

Several fits images can be loaded and displayed by the GUI. The selection is performed by the `INIT_IMG` combo box.

ALGORITHM SETTINGS

A first set of parameters are shown to the user. These parameters come from the interface document.

4.3.2. OUTPUT RESULTS

[3] `https://java.com/en/download/faq/java_webstart.xml`
[4] `http://oidb.jmmc.fr/search`

MODEL OF THE DATA

Result `OIFITS` have new column dedicated to model data. A plotting preset has been implemented to simultaneously display the observed data points and modeled ones. Future enhancement already are planned for the shared component developed by JMMC so that the observed and modeled data are stacked on the same plot.

FINAL IMAGE

As input images, the result image is displayed automatically after each run. The user can compare manually the various result by selecting result set in the dedicated list. Again, some computation ad comparison function are expected in the future versions.

PARAMETERS

A summary of output parameters is display as a basic table. The input data also are displayed to get memory of initial conditions.

EXECUTION LOG

The output of the spawn programs is collected and displayed to the user as free text.

4.4. APPLICATION MAINTENANCE

The current GUI has been based on the jMCS[5] framework provided by JMMC for its softwares. JMMC will provide support, maintain and develop new functions on this application. Current and future version will be reachable at http://www.jmmc.fr/oimaging. Bug reports, feature requests or documentation typo are submitted to the JMMC's project management system. Some issues already have been filed directly from the GUI at first development phases.

[5] `https://github.com/JMMC-OpenDev/jMCS`

5

IMAGE RECONSTRUCTION WITH BSMEM

5.1. INTRODUCTION

The BSMEM (BiSpectrum Maximum Entropy Method) software was first written in 1992 to demonstrate image reconstruction from optical aperture synthesis data. It has been extensively enhanced and tested since then (Baron and Young, 2008). A modified version of BSMEM has been developed as one of several alternative algorithms able to be controlled by the "OImaging" Graphical User Interface presented in chapter 4. It is this new version of BSMEM that is described here.

The BSMEM algorithm applies a fully Bayesian approach to the inverse problem of finding the most probable image given the evidence, making use of the Maximum Entropy approach to maximize the posterior probability of an image. BSMEM is available free-of-charge to the scientific community on submission of the academic license agreement at:

http://www.mrao.cam.ac.uk/research/optical-interferometry/bsmem-software/

BSMEM uses a trust region method with non-linear conjugate gradient steps to minimize the sum of the log(likelihood) (chi-squared) of the data given the image and a regularization term expressed as the Gull-Skilling entropy:

$$f_{\text{prior}}(\mathbf{x}) = \sum_n \left[x_n \log(x_n / \bar{x}_n) - x_n + \bar{x}_n \right] \tag{5.1}$$

with $\bar{\mathbf{x}}$ the *default image*; that is, the one which would be recovered in the absence of any data. The likelihood term used by BSMEM assumes independent Gaussian noise statistics for the amplitude and phase of the measured bispectrum. The optimization engine is

MEMSYS which implements the strategy proposed by Skilling and Bryan (1984) and automatically finds the most likely value of the hyperparameter μ that weights f_{prior} with respect to f_{data}. Because it doesn't attempt to convert the data into complex visibilities, a strength of BSMEM is that it can handle any kind of data sparsity (such as missing closure phases).

The default image (or model image) $\bar{\mathbf{x}}$ is usually chosen to be a Gaussian, a uniform disk, or a delta-function centered in the field of view, which conveniently fixes the location of the reconstructed object (the bispectra and power spectra being invariant to translation). This type of default image also acts as a support constraint by penalizing the presence of flux far from the center of the image. If a low-resolution image of the target object is available, this can be used instead.

5.2. INSTALLATION AND CONFIGURATION

These instructions describe local installation of BSMEM on your computer. Installation under Linux or Mac OS X is supported. OImaging can work with a local installation of BSMEM or by running BSMEM on a remote server; if using the latter option local installation is not needed.

First, follow the online instructions[1] to submit a license agreement (free-of-charge for academic use) for BSMEM and hence obtain the source code. You will receive the source code as a gzipped tar file. Check the README.md in the tarball for more up-to-date installation instructions.

BSMEM has successfully been built on Linux (Ubuntu 14.04, kernel 4.4.0 and CentOS release 6.6, kernel 2.6.32) and OS X (10.9 Mavericks). Any recent Linux distribution or OS X version should work. Building on Solaris is no longer supported.

5.2.1. REQUIREMENTS

The following libraries must be installed on your system:

1. PGPLOT 5.2 (http://www.astro.caltech.edu/~tjp/pgplot/): with at least XTERM, PS, and XSERVE drivers activated in the drivers.list file. On Redhat/Fedora Linux install the pgplot-devel package (from rpmfusion); on Debian/Ubuntu install the pgplot5 package. If using MacPorts on OS X install the pgplot port.
2. CFITSIO 3.x (http://heasarc.gsfc.nasa.gov/fitsio/)
   On Redhat/Fedora install the cfitsio-devel package; on Debian/Ubuntu install the libcfitsio3-dev package. If using MacPorts install the cfitsio port.
3. FFTW 3.x (http://www.fftw.org/)
   On Redhat/Fedora install the fftw-devel package; on Debian/Ubuntu install the libfftw3-dev package. If using MacPorts install the fftw-3 port.
4. NFFT 3.x (http://www-user.tu-chemnitz.de/~potts/nfft/)
   Download the source code and build/install using the standard sequence of com-

[1] http://www.mrao.cam.ac.uk/research/optical-interferometry/bsmem-software/

mands: `./configure`, `make`, `make install`. Alternatively, if using Debian Linux, you can install the libnfft3-dev package.

5. GLib 2.16 or later (https://developer.gnome.org/glib/)
   On Redhat/Fedora install the glib2-devel package; on Debian/Ubuntu install the libglib2.0-dev package. If using MacPorts install the glib2 port.
6. OIFITSlib 2.2.0 or later (https://github.com/jsy1001/oifitslib/)
7. CMake 2.8.6 or later (https://cmake.org/)

You will also require both C and Fortran 77 compilers to build BSMEM. BSMEM has been built successfully using gcc, together with g77 or gfortran.

BUILDING PGPLOT FROM SOURCE

If you need to build PGPLOT from source:

Ensure you have an X11 library installed (e.g. libX11-devel and libXt-devel packages on Redhat/Fedora, libX11-dev and libxt-dev on Debian/Ubuntu). Create the makefile using `makemake` (see `install-unix.txt`), then compile. Check that the C wrapper cpgplot is installed. Finally, remember to set up your `PGPLOT_DIR` environment variable to correctly point to the `pgxwin_server`, `rgb.txt`, and `grfont.dat` files.

5.2.2. RECOMMENDED COMPILERS

On OS X, we recommend the use of gcc 4.5 from MacPorts. Install it and build BSMEM as follows:

- Install MacPorts (https://www.macports.org/)
- Use MacPorts to install gcc45 and gcc_select
- Select this gcc and gfortran:

```
sudo port select --set gcc mp-gcc45
```

- Check that `gcc --version` and `gfortran --version` show version 4.5.4
- Make sure that BSMEM's dependencies are installed (see above)
- Set `PKG_CONFIG_PATH` to include locations of .pc files for libraries not installed through MacPorts, in addition to the MacPorts .pc files directory (normally /opt/local/lib/pkgconfig)
- Build BSMEM using CMake as outlined in the next section, but invoke cmake as `cmake -DCMAKE_C_COMPILER=gcc ..`

5.2.3. BUILDING BSMEM

CMake is now used for building BSMEM. For those not familiar with CMake, instructions can be found at https://cmake.org/runningcmake/. If you are using a Unix-like operating system the following commands should build the software:

```
cd build
cmake ..
```

```
make
```

Finally, use `sudo make install` to install the `bsmem` and `bsmem-ci` executables.

Note that the pkg-config utility is used to locate the GLib and OIFITSLib libraries that bsmem depends on. The search path for pkg-config must be configured to include the locations of the .pc files for those libraries, for example by setting the `PKG_CONFIG_PATH` environment variable.

5.3. BASIC USAGE

The basic workflow consists of specifying the input data, a starting image for the optimization (this typically also serves as the default image for the entropy function), and the algorithm settings, then starting `BSMEM`. The algorithm runs iteratively, until either the stopping criterion has been satisfied, or the specified maximum number of iterations has been reached.

5.3.1. DATA SELECTION

The interferometric data are specified as an `OIFITS` filename plus data selection parameters that define the subset of the file contents to use. An overview of the data selection process is given in chapter 4. `BSMEM` ignores the `USE_VIS` parameter since the current version cannot use complex visibility data. The non-standard `UV_MAX` parameter allows a maximum radius in the $u-v$ plane to be specified (in waves).

The reconstructed image is assumed to be wavelength-independent, so selecting an appropriate wavelength range from spectrally-dispersed data is a trade-off between minimizing the wavelength-variation in the selected data and maximizing the $u-v$ coverage. If in doubt, select a single wavelength channel.

5.3.2. INITIAL/DEFAULT IMAGE

If little is known about the object to be reconstructed, or an uninformative prior is desired, a good choice for the default image is usually a circular Gaussian centered in the field of view. With this kind of prior and adequate $u-v$ sampling, getting `BSMEM` to work usually boils down to choosing the pixel scale, image width, and prior width appropriately. These choices are explored in the examples in section 5.3.7, section 5.3.8 and section 5.4.1.

5.3.3. TOTAL FLUX CONSTRAINT

The image supplied to `BSMEM` specifies the relative pixel values; the total flux of the initial image is specified by the (currently non-standard) `INITFLUX` parameter. The initial flux should be set to a low value (e.g. 0.01). As `BSMEM` iterates, the flux in the image model is increased. The flux should be close to unity at convergence. In case there are no measured data at low spatial frequencies, this is enforced by adding an artificial $V^2 = 1$ data point at zero spatial frequency. The `FLUXERR` parameter specifies the error bar on this

data point and thus the strength of the "unit flux" constraint. Specifying a FLUXERR value that is too small will significantly delay convergence.

5.3.4. STOPPING CRITERION

BSMEM is able to estimate the value of the hyperparameter μ from the data. The AUTO_WGT parameter enables or disables this behaviour. If enabled, BSMEM automatically adjusts RGL_WGT to obtain a reduced χ^2 close to unity (this is the non-Bayesian "historic chi-squared = N method", which was found to be the most robust of the methods supported by MEMSYS).

If AUTO_WGT is disabled, then the behaviour of BSMEM is similar to MiRA. The user must choose an appropriate regularization weight for each imaging problem.

AUTO_RGL should be disabled if there is significant intrinsic variation in the selected data, either with wavelength or time (for time-variable targets), as "good" solutions for such cases should have $\chi^2 > 1$. Note that it is not currently possible to filter the OIFITS data by time within OImaging.

5.3.5. OTHER SETTINGS

The standard algorithm settings are described in chapter 4. BSMEM accepts the following non-standard settings (at the time of writing, only FLUXERR can be set in OImaging):

FLUXERR Error on synthetic zero-baseline squared visibility, see section 5.3.3

INITFLUX Initial image flux, see section 5.3.3

V2A Squared visibility error factor

T3AMPA Triple amplitude error factor

T3PHIA Closure phase error factor

V2B Squared visibility error offset

T3AMPB Triple amplitude error offset

T3PHIB Closure phase error offset

The last six settings can be used to adjust the error bars for the observed data, if you suspect they are systematically incorrect.

5.3.6. OUTPUT PARAMETERS

The following values can be displayed when BSMEM has finished:

NITER The total number of iterations done in the current program run.

CHISQ The reduced χ^2 at the end of the run.

ENTROPY The entropy value at the end of the run.

FLUX The total image flux at the end of the run.

RGL_WGT The weight of the regularization (determined automatically if the AUTO_WGT setting is true).

5.3.7. EXAMPLE 1

For the first example, the model of LkHa 101 used for the 2004 Beauty Contest (Lawson et al., 2004) has been selected (see Chapter 3). The following steps are needed to reconstruct an image from this data set, using BSMEM via the OImaging GUI:

1. Prepare an initial image in FITS format (currently this must be done outside of OImaging). The initial image defines the pixel scale and field of view of the reconstructed image.
2. Load the OIFITS file into OImaging. By default, all of the data are used.
3. Load the initial image into OImaging.
4. Adjust the algorithm settings as necessary. The default settings will work for this data set.
5. Click the "Run" button to launch BSMEM. When BSMEM has finished, the reconstructed image will appear in the "Image" tab of the "Data Visualisation" panel (see fig. 5.1).
6. Examine the "Output parameters" and "Execution log" tabs of the "Data Visualisation" panel to check that the algorithm converged successfully.
7. The "Save image" button may be used to save the reconstructed image.

The initial image used to obtain the reconstruction in fig. 5.1 was a circular Gaussian with a FWHM of 12 mas, centered in the FOV. The same image was used as the default image for the regularization term (this is the default behavior). The image scale was chosen as 0.25 milliarcseconds per pixel. This is equivalent to 6.8 pixels per fastest fringe $\lambda_{\min}/B_{\max}$ (BSMEM requires a minimum of 6).

When BSMEM starts, the image model has a low flux (given by INITFLUX, default 0.01). As BSMEM iterates, the flux in the image model is increased. fig. 5.2 shows the squared visibilities corresponding to the image model after 8 iterations, when the model flux is 0.71. At convergence after 29 iterations, the model flux is 1.004 giving a good fit to the shortest-baseline V^2 data.

The FLUXERR parameter specifies the error bar on a synthetic $V^2 = 1$ data point which helps enforce normalization of the image to unit flux. Reducing this value from the default 0.1 causes BSMEM to add more flux to the image in later iterations. If there are few real data points at low spatial frequencies, there is little information on where to put this extra flux, leading to spurious results. Hence FLUXERR should be varied with caution.

Use of a centrally-peaked default image is recommended, as this constrains the location of the reconstructed object (squared visibility and triple product data being invariant to translation). In this example, a circular Gaussian of FWHM 12 mas was cho-

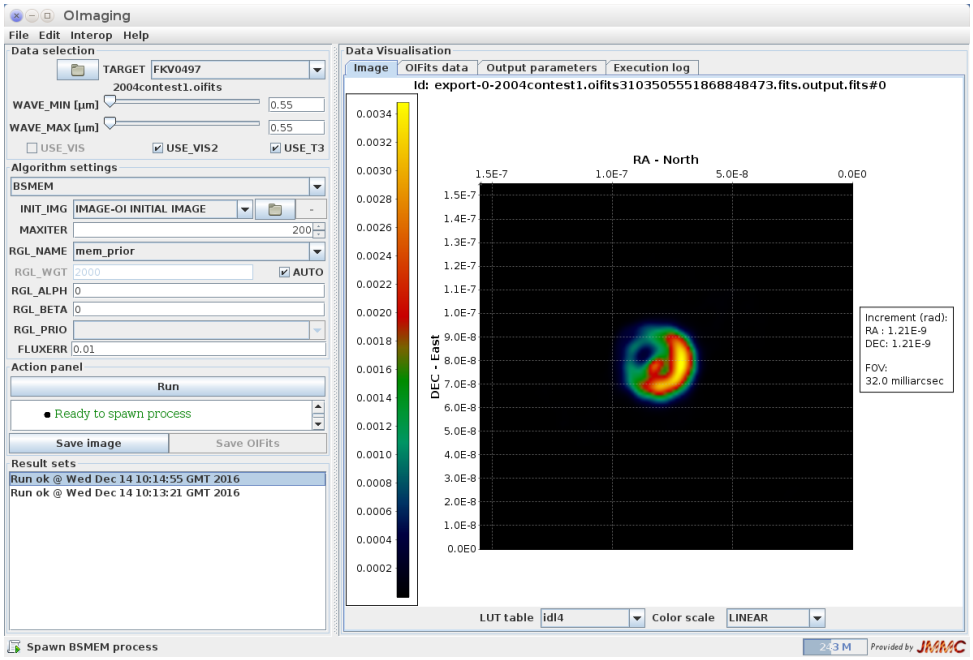

Figure 5.1: Screenshot of `OImaging` showing input parameters on the left and the final reconstructed image on the right. Note that the color scale is different from fig. 3.1. The algorithm settings are all at their default values except for `FLUXERR`, which was changed from 0.1 to 0.01.

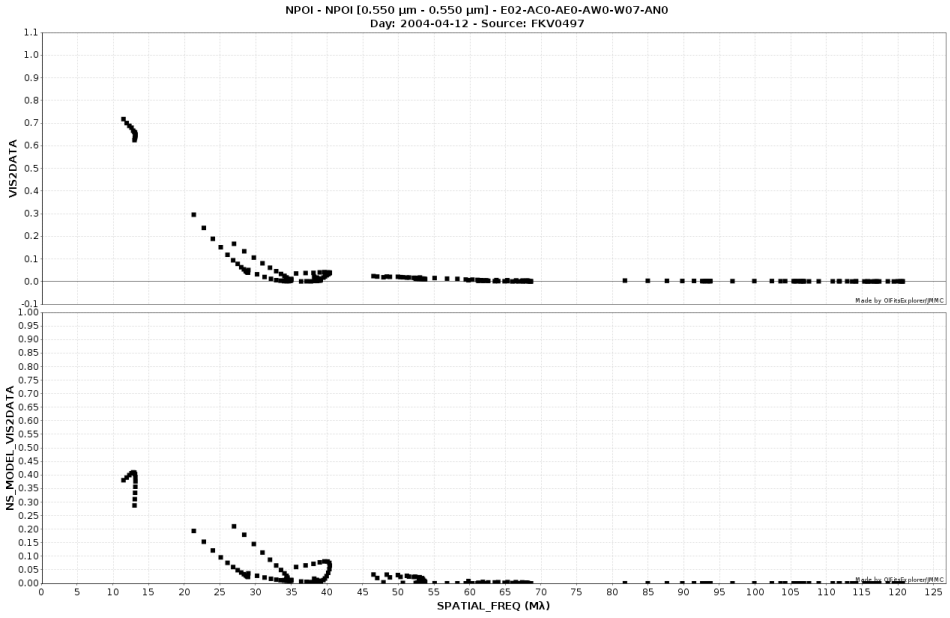

Figure 5.2: Plot of the input squared visibility data (top) and BSMEM's model of the data (bottom) after 8 iterations.

sen. Similar results are obtained with a Gaussian FWHM between 9 and 16 mas. Using a smaller width allows the reconstruction to converge faster, but increases the risk of missing real flux located far from the FOV center. The suggested approach is to start with a broad default image, infer the true size of the object from the result, then repeat the reconstruction using an appropriately-sized default image. Alternatively, the object size could be determined by fitting a simple model to the short baseline visibilities.

5.3.8. EXAMPLE 2

The second example data set is a simulated observation of a proto-planetary disk by VLTI/MATISSE. The synthetic data comprise observations in 14 spectral channels across the N-band (8–13 μm), using three configurations of the Auxiliary Telescopes. The data set used was part of the 2016 Beauty Contest (Sanchez-Bermudez et al., 2016) and can be found here: `www.opticalinterferometry.com`

The model includes a dust-enshrouded central star and a disk containing one gap, supposedly carved by a forming planet (Gonzalez et al., 2015). Further details, including plots of the simulated interferometric data, are given in Chapter 3.

GRAY LIMITATIONS

This example illustrates several issues associated with spectrally-dispersed data. Most importantly, BSMEM can only reconstruct a gray image. That is, BSMEM assumes all of the

observed data are consistent with a single truth image, and attempts to find the "simplest" image that is statistically compatible with the data. In general, the user must choose a subset of the available channels in order to optimize the $u-v$ coverage while minimizing the intrinsic variation between channels that BSMEM cannot solve for. An obvious consequence of the gray image limitation is that differential phase data cannot be used.

The $u-v$ coverage of this example data set is adequate to reconstruct each channel independently, given a suitable initial/default image. However, BSMEM's estimates of the regularization weight (hyperparameter) are not very good for this data set, as evidenced by the surprisingly large and rapid variation in the estimated weight as a function of wavelength. The same array configurations were used to simulate all channels, and the integrated spectrum of the object varies smoothly with wavelength, so *a priori* significant variation would not be expected.

A suitable work-around is to set a fixed regularization weight for all spectral channels. As the weight is no longer automatically chosen to give unit χ^2, its important to check the final χ^2 value is acceptable.

In OImaging, each channel must be selected in turn, and a separate reconstruction carried out. Figure 5.3 shows the settings and result for the longest-wavelength channel. BSMEM achieves convergence for all 14 channels using a regularization weight of 800, yielding χ^2 values between 0.9 and 1.3.

INITIAL IMAGE

The initial/default image (fig. 5.4) is the superposition of two circular Gaussian components centered in the FOV: a 90 mas FWHM component containing 99% of the flux and a 5 mas FWHM component containing 1% of the flux. The larger component provides a support constraint for the disk emission, and the smaller component fixes the location of the central star in the middle of the reconstructed image. Successful reconstructions can be obtained without the second image component, but the object position varies significantly with wavelength. The image scale is 2 mas/pixel.

5.4. ADVANCED USAGE

Sometimes with more challenging data sets, BSMEM won't converge to a good solution when an uninformative initial/default image is used. In such cases, a two-step procedure can often be used to reconstruct an image successfully. The following example illustrates such a method.

5.4.1. EXAMPLE 3

In this section an example two-step procedure is presented, using simulated data from the 2010 interferometric imaging beauty contest (Malbet et al., 2010).

The contest data simulates an observation of a red supergiant star with VLTI/AMBER. To add to the challenge, an unresolved companion star, located 8 stellar radii from the

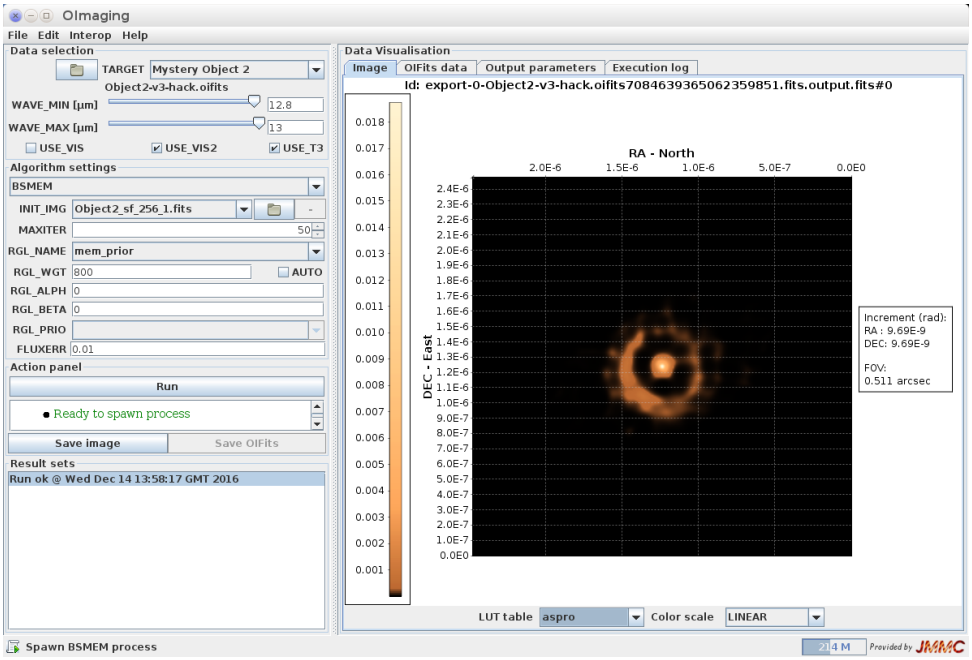

Figure 5.3: Screenshot of `OImaging` showing input parameters on the left and the final reconstructed image on the right. The longest-wavelength channel has been selected by increasing `WAVE_MIN` to 12.8 μm. Automatic calculation of `RGL_WGT` has been disabled for the reason explained in the text.

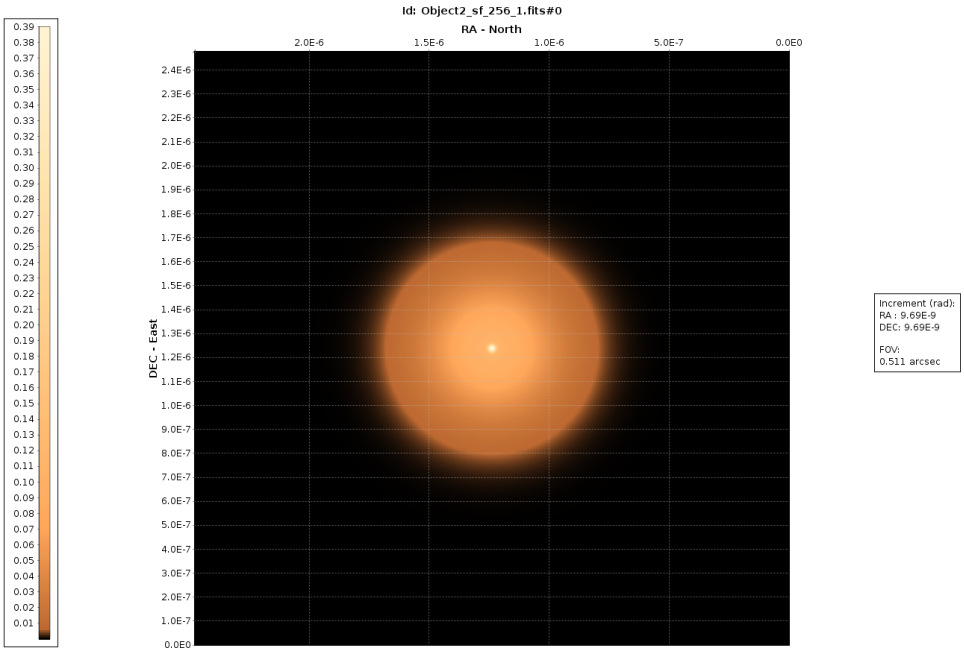

Figure 5.4: The initial/default image for example 2, comprising a 5 mas FWHM Gaussian superimposed on a 90 mas FWHM Gaussian. See text for explanation.

primary, was incorporated into the truth image. Two observations of the target object were simulated: a H-band medium spectral resolution (MRH) observation at resolution $R = 1500$ (512 channels), and a simultaneous H- and K-band low spectral resolution observation (LRHK) at $R = 35$ (20 channels).

The contest invited two kinds of submissions:

1. Three "gray" images derived from different subsets of the simulated spectral channels; and
2. A "spectral" image derived from the MRH data, allowing channel-to-channel differences.

The contest data files can be downloaded from the JMMC archive: `http://jmmc.fr/oidata/#cat_BeautyContest`

BSMEM CONTEST ENTRY

It was found that `BSMEM` was very slow (runtimes upwards of 24 hours on a standard PC) to converge to a well-fitting (reduced $\chi^2 < 5$) solution when given data from multiple spectral channels and an uninformative prior – probably in large part due to the wavelength-dependence of the object.

Thus a two-step approach was used for the `BSMEM` contest entry – first finding an image whose low spatial frequencies were compatible with the data, and then reconstruct-

ing each spectral channel separately using this as the default image. The reconstructed images from each spectral channel were then averaged to generate the three gray images specified by the contest organizers.

In particular, it was found that BSMEM did eventually converge using low-spatial-frequency data (baselines up to 60 m) from the first 8 spectral channels of the low-spectral-resolution data set. The resulting image (fig. 5.5) was convolved with a Gaussian with FWHM equal to the finest fringe spacing and thresholded to remove features judged to be noise. This spatially-filtered image was used as the initial/default image for the subsequent reconstructions of individual spectral channels, which typically converged after 70 to 250 iterations (of order 30 minutes on a standard PC). To generate the gray submissions, the spectral channels were averaged with uniform weighting, and pixels below a specified threshold in the average image were set to zero. The threshold value was chosen to remove most of the obvious circumstellar artefacts, each of which typically appeared in the one spectral channel only.

When following a two-step process, a pixel scale suitable for the full data set must be used for both steps (the initial image for step 2, derived from the result of step 1, defines the pixel scale for the final reconstruction). A scale of 0.4 mas per pixel gives 6.01 samples per fastest fringe at 1.521 μm and is therefore suitable for this example. With this sampling, a 512×512 pixel image is needed to reconstruct both stars.

Since the second-step reconstructions were performed for each spectral channel separately, it was necessary to write a custom script to run BSMEM many times, selecting a different subset of the data for each run.

USING OIMAGING

This section illustrates the use of OImaging to perform a similar two-step procedure for a single channel of the 2010 contest LRHK data (Mystery-Low_HK.oifits).

For the first step, the high spatial frequencies must be removed from the data. This can be done using the oifits-filter program that comes with OIFITSlib. Alternatively, a future OImaging release should allow setting of UV_MAX to achieve the same result. A 60 mas FWHM circular Gaussian is used as the default image. For this data set, MAXITER must be set to a large number as several hundred iterations are needed to converge. The resulting image is shown in fig. 5.5.

The first reconstructed image is exported and processed outside of OImaging. The image is convolved with a Gaussian function, then all pixels below a specified threshold are set to zero. The FWHM of the blurring function is the finest fringe spacing in the filtered data, in this case 5.6 mas. This has the effect of smoothing out all of the stellar surface structure. The threshold value (0.003) is chosen to remove all of the noise in the empty regions of the field.

The blurred and thresholded image is used as the model image for a second BSMEM run on the full-resolution data set. For this second step, each spectral channel must be processed separately. Figure 5.6 shows the result for the first channel.

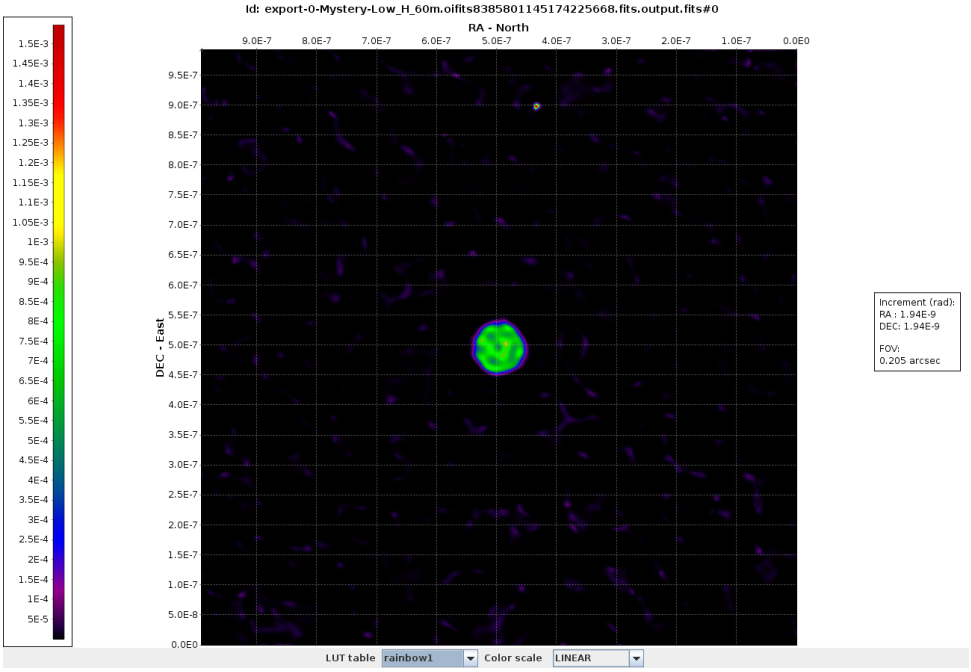

Figure 5.5: Image reconstructed from a low-spatial-frequency subset ($B < 60$ m) of the LRH data, using a 60 mas Gaussian initial/default image. The companion star is visible towards the top center of the field.

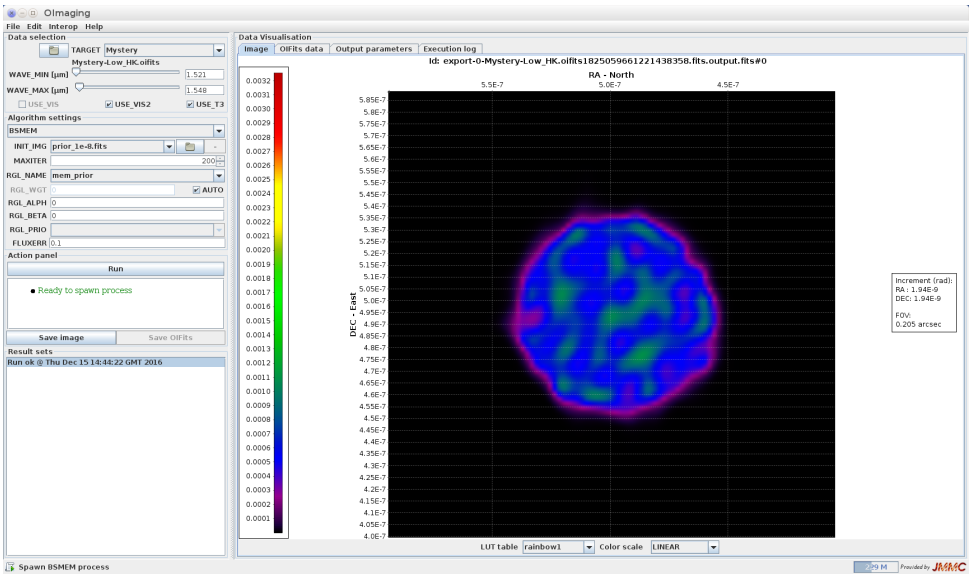

Figure 5.6: Screenshot of `OImaging` showing input parameters on the left and the reconstructed image (after 72 iterations) on the right. The shortest-wavelength channel has been selected. The image has been zoomed to show the supergiant star surface.

6

IMAGE RECONSTRUCTION WITH MiRA

6.1. INTRODUCTION

MiRA, the Multi-aperture Image Reconstruction Algorithm, was developed by Éric Thiébaut,[1] aiming at a versatile package for interferometric image reconstruction, with no restriction on the type of data or regularisation that can be used. The rational is described in detail by Thiébaut (2008, 2013).

MiRA follows a general principle for image reconstruction: it searches for the image x^+ (a grid of square pixels) which minimizes a two-term penalty function with respect to the parameters of the image:

$$x^+ = \underset{x\in\Omega}{\arg\min}\Big\{f(x) = f_{\text{data}}(x|d) + \mu\, f_{\text{prior}}(x)\Big\}. \tag{6.1}$$

The image parameters (e.g., the pixel values) are constrained to the set of possible images Ω that obey to restrictive conditions, such as *normalisation* (the sum of the pixels is equal to 1) and *positiveness* (all pixel intensities are non-negative).

The penalty criterion connects the data and the prior by means of f_{data} and f_{prior}, usually known respectively as the *likelihood* and *regularisation* terms. While the former measures the discrepancy between the actual data d – e.g., squared visibilities V^2, closure phases ϕ_c, visibility amplitudes V and baseline phases ϕ – and their model given the image x, the latter is a penalty which enforces additional priors and is required to avoid artefacts. The regularisation term is mandatory because the data alone cannot unambiguously yield an unique image. The *level of regularisation* μ is a positive *hyperparameter*, adjusted to set the relative weight of the a priori information.

[1] Email: thiebaut@univ-lyon1.fr

The minimisation of eq. (6.1) is done by a non-linear optimisation algorithm. Since MiRA does not perform a global optimisation, a successful restoration when using closure phases (a non-convex problem) relies on the initial image.

MiRA does not try to recover the phases directly, being thus able to handle any type of observable, and it presents no restrictions on the kind of regularisation to be applied to the reconstruction process.

6.2. INSTALLATION AND CONFIGURATION

In order to use MiRA, it is necessary to install Yorick (version 2.1 or superior), OptimPack (from version 1.2 on) and Yeti (version ≥ 6.2.0). It is also advisable to install YNFFT, a Yorick plug-in for the Nonequispaced Fast Fourier Transform, for faster operations. The aforementioned packages can be retrieved from the following links:

- Yorick: http://dhmunro.github.io/yorick-doc/
- OptimPack: https://github.com/emmt/OptimPack
- Yeti: https://github.com/emmt/Yeti
- YNFFT: https://github.com/emmt/ynfft

OptimPack, Yeti and YNFFT can be downloaded either as Git repositories or as archive files. The same principle applies to the MiRA software.

6.2.1. YORICK

INSTALLING FROM A PACKAGE MANAGEMENT PROGRAMME

On Ubuntu based distributions, Yorick can be installed using the apt terminal command, or by means of the synaptic package manager. In case of the former, it is sufficient to write

```
sudo apt install yorick
```

For recent distributions, the previous command will also automatically install rlwrap, which is very handy to recall previous commands inside Yorick environment.

COMPILING AND INSTALLING FROM SOURCE

In case you prefer to compile Yorick from scratch, then execute the following steps:[2]

- Unzip/untar the latest version of Yorick. Example:

  ```
  unzip dhmunro-yorick-y_2_2_04-6-gbc24b71.zip
  ```

  or

  ```
  tar xzvf dhmunro-yorick-y_2_2_04-6-gbc24b71.tar.gz
  ```

[2] This procedure was tested in Linux Ubuntu MATE 16.04, Mint 17.3 Rosa Mate and several Manjaro and xUbuntu (Ubuntu, Kubuntu, Ubuntu Gnome) distributions over the years, both 32 and 64 bits.

- Go to the unpacked directory and type:

```
make config
make
make check
make install
```

This will create a subdirectory **relocate/** in the source tree. The Yorick executable is **relocate/bin/yorick**. You can move the **relocate/** directory wherever you want (the name "relocate" is not important), but any changes in the relative locations of the files therein will prevent Yorick from starting correctly. For example:

```
mv relocate/ /opt/yorick_2.2.04-6__64b
```

You can add the path where the Yorick executable is to your **$PATH** variable, or create a softlink to the Yorick executable from wherever you like, or execute Yorick from a shell script outside its **relocate/** directory.

USING A RELOCATABLE VERSION

In case you want to use a previously compiled version of Yorick, just uncompress the **relocatable/** directory to a desired folder (e.g., **/opt**) and add ***[Yorick-relocatable-folder]/bin/yorick*** to your **$PATH**, or create a soft link to it.

6.2.2. OPTIMPACK, YETI AND MIRA

In order to install any of these packages, just follow the instructions on the corresponding GitHub repositories (see urls in page 52).

6.3. BASIC USAGE

After choosing the regularisation, MiRA needs the following inputs:

(i) The data, in the OIFITS standard format, either table versions 1 (Pauls et al., 2005) or 2 (Duvert et al., 2015).
(ii) An optional estimate for the image – important when only power-spectra and bispectra are present.
(iii) The size of the pixel, $\delta\theta$.
(iv) The hyper-parameter μ.
(v) The maximum number of iterations (Gomes et al., 2017).

The initial image is not important if the data is composed of complex visibilities, with both visibility amplitudes and baseline phases, as the problem becomes complex and MiRA yields an unique solution. However, complex visibilities are still scarce and in face of the more common power-spectra and closure phases, the initial image plays an important role in the success of the reconstruction process. The choice of the lateral size

$\Omega = N \cdot \delta\theta$ of the square image of $N \times N$ pixels is fundamental because it provides a strict constraint on the support of the object and strongly affects the restoration of the image (Gomes et al., 2017). On the one hand, since we are mapping the interferometric data into a discrete grid of pixels, the dimensions of the pixel grid need to be chosen large enough in order to sample all the measured spatial frequencies. On the other hand, N cannot be too large, so as to make the computation practical. According to the Nyquist-Shannon criterion, the pixel size shall sample the maximum angular resolution, but it also shall account for some level of super-resolution introduced by the image reconstruction algorithm. As a rule of thumb, $\delta\theta$ shall be of the order of

$$\delta\theta \lesssim \frac{\lambda}{4\,B_{\max}}, \tag{6.2}$$

where $B_{\max}$ is the maximum projected baseline length.

`MiRA` can be used both in the command line interface or inside the `Yorick` environment. The following instructions are highly inspired in the documentation available in the `MiRA` package (for more details, consult the corresponding GitHub repository, see page 52).

6

6.3.1. USING MIRA FROM THE COMMAND LINE

The general syntax for using `MiRA` from the command line interface is

```
mira [OPTIONS] INPUT [...] OUTPUT
```

where *[OPTIONS]* are optional settings, **INPUT [...]** are any number of OIFITS input data files and **OUTPUT** is the name of the output FITS file to save the resulting image. Option *-help* can be used for a short description of all options.

DATA SELECTION

`MiRA` reconstructs a grey image from the interferometric data. Although `MiRA` cannot currently handle with multi-wavelength data as a whole, the user can select one wavelength included in a set of observations by means of one or a combination of keywords. For example, the wavelengths of the selected data can be specified as the end points of the spectral range:

```
... -wavemin=MINVAL -wavemax=MAXVAL
```

or as the central wavelength and bandwidth:

```
... -effwave=CENTER -effband=WIDTH
```

The arguments of these options have units of length. For instance,

```
... -wavemin=1.6µm -wavemax=1.8microns
```

is the same as

```
... -effwave=1700nm -effband=200nm
```

If the input data files contain observations for more than one object, the target to consider can be specified as

```
... -target=NAME
```

IMAGE PARAMETERS

An initial image for the reconstruction can be provided as

```
... -initial=FILENAME
```

where **FILENAME** is the name of the FITS file with the image to start with.

If no initial image is provided, the reconstruction starts with a random image and option *-seed=NUMBER* can be used to seed the random generator.

By default, the reconstructed image will have the same pixel size and dimensions as the initial image if provided. Otherwise, the pixel size can be specified with the option *-pixelsize=PIXSIZ*, and the image dimensions can be chosen with *-dim=NUMBER* or *-fov=ANGLE*. *PIXSIZ* and *ANGLE* are in angular units, and *NUMBER* is the number of pixels in each lateral dimension of the image (assumed a square image). For instance,

```
... -pixelsize=0.25mas -fov=100mas
```

or

```
... -pixelsize=250e-6arcsec -dim=400
```

both yield a 400×400 image with a pixel size of 0.25 mas.

6.3.2. FOURIER TRANSFORM

The nonequispaced Fourier transform of the pixels can be computed by different methods: *-xform=exact* uses an exact transform, *-xform=nfft* uses a precise approximation by the NFFT algorithm, while *-xform=fft* uses a built-in algorithm which is less precise. The exact transform can be very slow if the image is large and/or if there are many data. The two others exploit an FFT algorithm and are faster. If you have installed Yorick NFFT plug-in, *-xform=nfft* is certainly the method of choice.

IMAGE CONSTRAINTS

The total flux of the sought image, and the lower and upper bounds for pixel values can be specified with the options *-normalization=SUM*, *-min=LOWER* and *-max=UPPER* respectively. For instance,

```
... -normalization=1 -min=0.0
```

is a must for processing OIFITS data.

REGULARISATION

Because the algorithm has to deal with sparse interferometric data, the user has also to indicate which regularisation must be applied to interpolate missing data. The regularisation is the kind of prior imposed to the reconstructed image. `MiRA` accepts any regularisation, but it has several already implemented in the code (see, for e.g., Thiébaut 2013, and `rgl.i`, inside `MiRA` folder, for details). For all implemented regularisations, the relative strength of the prior (compared to the data) is specified with the option `-mu=`μ, where $\mu \geqslant 0$.

Of the available regularisations, the following are the most relevant:

- **Edge-preserving smoothness**, which is selected with the options

  ```
  -regul=hyperbolic -mu=μ -tau=τ -eta=η
  ```

  where τ is the edge threshold and η the scale of the finite differences to estimate the local gradient of the image. Distinctive scales can be set for different dimensions by providing a list of values to `-eta`, e.g., `-eta=1,1,0.3`. By default, `-eta=1`. Using a very small edge threshold, compared to the norm of the local gradients, mimics the effects of *total variation* (TV) regularisations. Conversely, using a very small edge threshold yields a regularisation comparable to *quadratic smoothness*.
- **Quadratic compactness**, which is selected with the options

  ```
  -regul=compactness -mu=μ -gamma=γ
  ```

  where γ is the full width at half maximum (FWHM) of the prior distribution of light. This parameter has angular units. For instance, `-gamma=15mas`.

CAVEATS

- All units are in SI (i.e., angles are in radians, wavelengths in meters, etc.). A very common error in parameter settings is to use completely out of range values because you assume the wrong units. To make things a easier, some constants are pre-defined by `MiRA` package, and you can use, for instance, `5*MIRA_MICRON` (instead of 5×10^{-6} m), or `3*MIRA_MILLIARCSECOND` (instead of $1.454\,44 \times 10^{-8}$ rad).

- Positivity is a must, you cannot expect a good image reconstruction (at least with any practical uv-coverage) without option **xmin=0** in `mira_solve`. If you use certain regularisations such as *entropy*, you must specify a strictly positive value for *xmin* (choose a very small value, for instance: **1e-50*nrm/npix** where *nrm* is the normalisation level and *npix* the total number of pixels).
- Likewise, **normalization=1** must not be omitted if your data set obeys OIFITS standard (i.e., visibilities are normalised) and has no explicit measurement at frequency (0,0).
- The cost function is highly non-quadratic and may cause difficulties for `OptimPack` to converge. To overcome this, it is sometimes very effective to simply restart `mira_solve` with an initial image provided by a previous reconstruction (possibly after re-centring by `mira_recenter`). For instance,

```
      img1 = mira_solve(db, img0, maxeval=500, verb=1, xmin=0.0,
normalization=1, regul=rgl, mu=1e6);
      img2 = mira_recenter(img1);
      img2 = mira_solve(db, img2, maxeval=500, verb=1, xmin=0.0,
normalization=1, regul=rgl, mu=1e6);
      img2 = mira_recenter(img2);
      img2 = mira_solve(db, img2, maxeval=500, verb=1, xmin=0.0,
normalization=1, regul=rgl, mu=1e6);
      ...
```

- It is usually better to work with a purposely too high regularisation level and then lower the value of *mu* as the image reconstruction converges.

6.3.3. USING MIRA INSIDE YORICK INTERPRETER

The process is usually initiated with the command **mira_new**, such as in

```
db= mira_new("data.oifits");
```

A wavelength included in a set of observations can be selected by means of the keyword *eff_wave*. Example:

```
db= mira_new("data.oifits", eff_wave= 1.617e-6);
```

The image will be restored using data with wavelengths in the range `eff_wave` $\pm 0.5 \times$ `eff_band` (*eff_band* defaults to 0.1 µm if not input). If *eff_wave* is not specified, `MiRA` uses the average wavelength computed from the first block of data.

Then, `MiRA` has to be set-up. The procedure amounts to indicate the dimensions of the restored image, the pixel size and the type of linear transformation for the extrapolation of the observables:

```
mira_config, db, dim= 200, pixelsize= 0.15*MIRA_MILLIARCSECOND, xform= "exact";
```

The keyword *dim*, the number of pixels along the width/height of the restored image, must be indicated in pixels. Alternatively, you can introduce the size of the corresponding FOV, in radians, via the keyword *fov*. MiRA includes several multiples and submultiples of angular units, such as **MIRA_MILLIARCSECOND**, that can be handy when indicating too small or too large values. The *xform* keyword can assume the options **exact** (*exact* Fourier transformsm, the default), **fft** (*Fast* Fourier transforms) or **nfft** (*Nonequispaced Fast* Fourier Transforms).

The regularisation can be indicated with the **rgl_new** command. For example,

```
rgl= rgl_new("compactness");
```

6.3.4. EXAMPLES

For the following examples, which illustrate some applications of the MiRA software, it is assumed that all necessary packages are properly installed (see section 6.2.2, page 53).

EXAMPLE 1 (INSIDE YORICK INTERPRETER)

Launch Yorick and load MiRA (this shall automatically load Yeti plug-in):

```
include, "mira.i";
```

Load the OIFITS data file (**db** will be our MiRA instance for this data file):

```
db = mira_new("data1.oifits");
```

If there are several spectral channels in the data file, you must choose one with keyword *eff_wave* or choose a spectral range with keywords *eff_wave* and *eff_band*, as described before.

Configure the data instance for image reconstruction parameters:

```
mira_config, db, dim=100, pixelsize=0.5*MIRA_MILLIARCSECOND, xform="exact";
```

Choose a suitable regularisation method:

```
rgl = rgl_new("smoothness");
```

Attempt an image reconstruction (from scratch):

```
dim = mira_get_dim(db);
img0 = array(double, dim, dim);
img0(dim/2, dim/2) = 1.0;
img1 = mira_solve(db, img0, maxeval=500, verb=10, xmin=0.0, normalization=1,
regul=rgl, mu=1e6);
```

Continue the reconstruction with a re-centred image:

```
 img1 = mira_solve(db, mira_recenter(img1), maxeval=500, verb=10, xmin=0.0,
normalization=1, regul=rgl, mu=1e6);
 img1 = mira_solve(db, mira_recenter(img1), maxeval=500, verb=10, xmin=0.0,
normalization=1, regul=rgl, mu=1e6);
 img1 = mira_solve(db, mira_recenter(img1), maxeval=500, verb=10, xmin=0.0,
normalization=1, regul=rgl, mu=1e6);
```

The resultant image is illustrated in fig. 6.1.

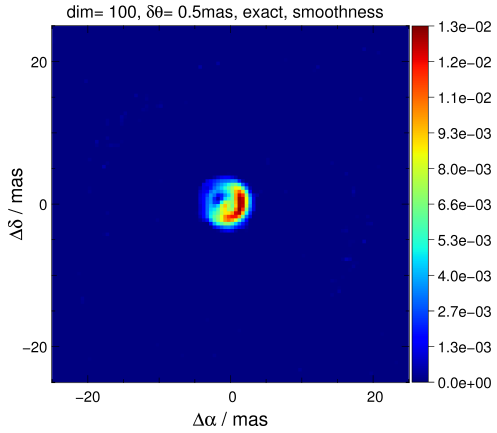

Figure 6.1: Restored image obtained by MiRA using "smooth" support, lateral width of 100 pixels, and a pixel size of 0.5 mas.

Extract the central part of the image and restart the reconstruction with a higher resolution image and a smaller field of view:

```
 cut = (dim + 2)/4;
 scale = 4.0;
 new_img1 = mira_rescale(img1(1+cut:-cut, 1+cut:-cut), scale=scale);
 new_dim = dimsof(new_img1)(2);
 new_pixelsize = mira_get_pixelsize(db)/scale;
 mira_config, db, dim=new_dim, pixelsize=new_pixelsize;
 new_img1 = mira_solve(db, new_img1, maxeval=500, verb=10, xmin=0.0,
normalization=1, regul=rgl, mu=1e6);
 new_img1 = mira_solve(db, mira_recenter(new_img1), maxeval=500, verb=10, xmin
=0.0, normalization=1, regul=rgl, mu=1e7);
 new_img1 = mira_solve(db, mira_recenter(new_img1), maxeval=500, verb=10, xmin
=0.0, normalization=1, regul=rgl, mu=1e8);
 new_img1 = mira_solve(db, mira_recenter(new_img1), maxeval=500, verb=10, xmin
=0.0, normalization=1, regul=rgl, mu=1e9);
```

```
 new_img1 = mira_solve(db, mira_recenter(new_img1), maxeval=500, verb=10, xmin
=0.0, normalization=1, regul=rgl, mu=1e10);
```

Figure 6.2 illustrates the resultant image.

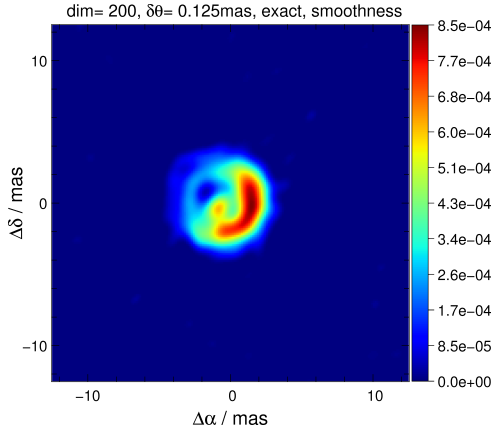

Figure 6.2: Restored image obtained by MiRA using "smooth" support, lateral width of 200 pixels, and a pixel size of 0.125 mas.

Choose a $\ell_2 - \ell_1$ smoothness:

```
 rgl = rgl_new("xsmooth", "cost","cost_l2l1", "threshold",2e-5, "dimlist",dimsof
(new_img1));
 mira_config, db, dim=256, pixelsize=0.1*MIRA_MILLIARCSECOND, xform="fft";
 dim = mira_get_dim(db);
 r = abs(mira_get_x(db), mira_get_x(db)(-,));
 prior = 1.0/(1.0 + (2.0*r/(5.0*MIRA_MILLIARCSECOND))^2);
 prior *= 1.0/sum(prior);
 rgl_config, (rgl = rgl_new("quadratic")), "W", linop_new("diagonal",
1.0/prior);
 img0 = (prior == max(prior)); img0 *= 1.0/sum(img0);
 img1 = mira_solve(db, img0, maxeval=500, verb=1, xmin=0.0, normalization=1,
regul=rgl, mu=1e4);
 img1 = mira_solve(db, img1, maxeval=500, verb=1, xmin=0.0, normalization=1,
regul=rgl, mu=1e4);
 img2 = mira_solve(db, img1, maxeval=500, verb=1, xmin=0.0, normalization=1,
regul=rgl, mu=5e3);
 img2 = mira_solve(db, img2, maxeval=500, verb=1, xmin=0.0, normalization=1,
regul=rgl, mu=5e3);
 img3 = mira_solve(db, img2, maxeval=500, verb=1, xmin=0.0, normalization=1,
regul=rgl, mu=2e3);
 img3 = mira_solve(db, img3, maxeval=500, verb=1, xmin=0.0, normalization=1,
regul=rgl, mu=2e3);
```

```
 img4 = mira_solve(db, img3, maxeval=500, verb=1, xmin=0.0, normalization=1,
regul=rgl, mu=1e3);
 img4 = mira_solve(db, img4, maxeval=500, verb=1, xmin=0.0, normalization=1,
regul=rgl, mu=1e3);
```

6.4. ADVANCED USAGE

When restoring images in the phase closure case, i.e., when the phase information is solely supplied by closure phases, it may happen that the algorithm stalls in local minima, preventing the image to evolve to the optimal or, at least, to a better solution. *Soft thresholding* can be used in order to make `MiRA` jump out of local minima. It consists of taking as input for the next step the previous restored image subtracted by a percentage of its maximum, assuring that no negative values are passed. This can be achieved, for instance, with the following code:

```
...
img1= mira_solve(db, img0, maxeval=500, verb=10, xmin=0.0, ...);
img2= mira_solve(db, max(0.0, img1 - 0.05*max(img1))), maxeval= 500, verb=10, ...);
...
```


6.5. TROUBLESHOOTING

1. When compiling `Yorick` from the source (see section 6.2.1), make sure you have the package **`libx11-dev`** installed, or you will get compilation errors.
2. If you have an Intel graphics card and are using the nouveau or Intel driver, `Yorick` might crash after running `MiRA` or when trying to use a small font height in plots. If that is the case, install the 75 and 100 dpi bitmap font packages. For Debian based machines, type in the terminal:

   ```
   sudo apt install xfonts-75dpi xfonts-100dpi
   ```

   Test that everything is OK. You can do it inside `Yorick` with, for example

   ```
   plt, ''Hello, World!", 0.35, 0.65, height=6;
   ```

   or by executing a `mira_solve()` command.

7

IMAGE RECONSTRUCTION WITH WISARD

7.1. INTRODUCTION

WISARD stands for "Weak-phase Interferometric Sample Alternating Reconstruction Device". Developed around 2005 at ONERA by S.Meimon, the first version of WISARD recovers a monochromatic ("grey") image from power spectrum and closure phase data alone. In contrast to, e.g., MiRA, WISARD first converts the power spectrum and closure phases measured by optical interferometry into equivalent pseudo complex visibilities (so-called "myopic") data akin to what radio astronomy produces. "Equivalent" here meaning 1) that the phases and their variance are computed from, and are compatible with, the measured phase closures and variances and 2) the amplitudes and variances are computed from, and are compatible with, the measured squared visibilities and variances. A subsequent convexification approximation is performed.

From then on, reconstruction proceeds on the same grounds as for other image-reconstruction programs, by minimizing a compound criterion with two terms, one measuring the fit of the reconstruction given the data, the other regularizing the reconstruction based on a variety of priors (positivity, smoothness, etc).

The following text refers mostly to the envisioned use of WISARD through the OImaging GUI, and heavily borrows from the documentation on the "standalone" version available at http://www.mariotti.fr/wisard_page.htm, which contains all relevant information not only on WISARD but also all references to the scientific publications associated with the concepts behind WISARD and the copyrights. In particular, it is expected from the user of WISARD, directly or through OImaging, that he/she obeys the acknowledgments requirements described in http://www.jmmc.fr/wisard/acknowledgement.

7.2. INSTALLATION AND CONFIGURATION

The installer can be downloaded from the JMMC website at http://www.mariotti.fr/wisard_page.htm.

Although WISARD consists in a set of IDL procedures, it is not necessary to buy this expensive software: the alternative, open-source, free IDL clone "GDL" is able to run WISARD. GDL is available on several distributions (Debian, FreeBSD but also MacOS and Windows). See http://gnudatalanguage.sourceforge.net/ for details.

We recommend GDL with a version ≥ 0.9.6. WISARD uses several procedures and functions from the IDLASTRO library (http://idlastro.gsfc.nasa.gov) and C. Markwardt's library (https://www.physics.wisc.edu/~craigm/idl/). In the following, the terms "IDL" and "GDL" are equivalent.

After unpacking the software (see below), you need to accept the license, which can be accessed under IDL by typing:

```
dummy=wisard(/copyright)
```

7.2.1. DEPENDENCIES

WISARD needs Éric Thiébaut's "OptimPack" library for IDL (OptimPack_IDL*.so) and its IDL frontend (op_*.pro files) to be installed. For convenience, the OptimPack IDL frontend and several pre-compiled OptimPack libraries for IDL (for Linux, Mac/OS and Solaris, on 32 and 64-bit architectures, and for Windows) are included in the lib/optimpacklib/ directory of the WISARD distribution, with their author's permission. WISARD selects the library automatically so this should be transparent. In case of trouble, refer to the complete documentation on the JMMC website.

7.2.2. UNPACKING THE SOFTWARE

WISARD is distributed in a compressed tar archive. Unpacking it is straightforward and will create a WISARD/ directory under which is the distribution. From the shell prompt, type:

```
tar xzvf Wisard-<version>.tgz
```

7.3. BASIC USAGE

As for other image-reconstruction programs, WISARD will need an input data set (one or more OIFITS file — one file with OImaging), a field-of-view value (in milliseconds of arc), and a choice between several regularisations. A maximum value for the number of iterations or, alternatively, a convergence threshold, can be given to limit the number of iterations. Additionally, an input "guess" image can be used as a starting point. In the absence of a guess image, WISARD uses the *dirty map* (thresholded to positive values,

directly computed as the Fourier transform of the myopic complex visibilities). Other input options are possible, for example to set the minimum number of pixels reconstructed (by default this value is the minimum number needed to fulfill the Shannon-Nyquist criterion).

The choice of the Field of View (FOV in the OImaging interface) is critical for the goodness of the WISARD reconstruction.

WISARD takes all tables in the OIFITS file(s) passed and merges them in the minimal set of tables possible: one table per interferometric observable (V2, closure...), array, object,spectral setup. If the data contains several objects, the one for which image reconstruction should be attempted should be given. Besides there are some constraints on the data (next section).

7.3.1. CONSTRAINTS ON THE INPUT DATA

WISARD quickly evades the problematic of image reconstruction in optical interferometry by using "equivalent" radio-like complex visibilities. By doing so, it poses severe constraints on the observational data:

- WISARD needs to correctly associate triplets and visibility baselines. That is, any closure measurement should be associated with 3 existing square visibilities (one on each baseline associated with the triplet). This imposes that related closures and visibilities share the same value of TIME or MJD, a constraint not imposed by the OIFITS format and not followed by some instruments. It is possible to ask WISARD to allow for small differences in the time-stamps of closures wrt. their related square visibilities by giving a delta time (in seconds) in which two TIME or MJD can be deemed equal. The consequence of this restriction is that WISARD may retain fewer observations than present in the data.
- Due to its matrix internal treatment, and pending a complete rewrite (yet unforeseen), WISARD cannot mix arrays with different numbers of telescopes. WISARD will try to determine the number of telescopes used, based on statistics of the baselines present in the data, and can be corrected if it guesses wrong. In the transformation from closures and squared visibilites to complex visibilities, unknown phases are added, but the number of unknowns decreases rapidly with the number of simultaneous telescopes used: it is in percentage 66% for 3 telescopes, 50% for 4 and 33% for 6. WISARD will perform way better when it can use more simultaneous telescopes, even if doing so it discards data obtained with fewer telescopes, which can happen if, e.g., some baseline was "lost" during the observations. In the more desperate cases, one can ask WISARD to consider the data as only coming from 3 telescopes: all the closures present in the data will be taken into account, but separately, and at the expense of a worse data reconstruction.
- WISARD uses only the OIFITS observables VIS2DATA, VIS2ERR and T3PHI, T3PHIERR. All other observables are ignored.

In summary: inputting WISARD with a set of heterogeneous data observed using various

instruments, or using various numbers of baselines, will likely result in WISARD using only a "homogeneous" subset of all the data and discard all the rest. This selection process is not usually significant with "modern" OIFITS made by recent optical interferometry instruments.

7.3.2. DATA SELECTION

WISARD reconstruct a wavelength-independent ("grey") image. Selecting a range of spectral channels (in the case of multi-channel data) is possible and will permit a piece-wise reconstruction for objects that change shape significatively along the observed spectral band.

7.3.3. CHOICE OF REGULARIZATION

Beside the usual constraint of positivity of the reconstruction, WISARD proposes a choice of regularizations, mathematical expressions of the somewhat fuzzy preconceived idea that the object to be reconstructed has some kind of smoothness (or piece-wise smoothness, or global smoothness apart from some spikes, etc). In WISARD, several regularization terms are available to embody this prior knowledge of the solution.

1. A **smooth solution** is obtained by a quadratic regularization, which uses the object's PSD (Power Spectral Density) as input. This regularization needs the PSD, a 2-D map of size NP_MIN×NP_MIN containing the PSD for the (quadratic) regularization of the reconstruction. Another parameter, MEAN_O, can be passed to this regularization: it is a 2-D map of size NP_MIN×NP_MIN containing the MEAN Object to be used for regularization of the reconstruction.
   Using the OImaging interface, this is triggered by the PSD choice.
2. A **piece-wise smooth prior** is obtained by a linear-quadratic (so-called L1L2) regularization. This prior is called *edge-preserving* as it allows sharp edges in the object if the data is compatible with them, contrarily to a quadratic regularization. Associated parameters are DELTA and SCALE:
   - DELTA is a scalar factor for L1-L2 regularization, used to set the threshold between quadratic (L2) and linear (L1) regularization.
   - SCALE is a scalar factor for L1-L2 regularization, which should be of the order of the RMS object's gradient value. More details are available in the JMMC documentation.

   Using the OImaging interface, this is triggered by the L1L2 choice.
3. A **variant** of this regularization, designed to grant the solution with some smoothness while allowing spikes: this pixel-independent (or white) L1L2 regularization is called *spike-preserving*. Associated parameters are DELTA and SCALE as above, except that SCALE should be here of the order of the average object value.
   Using the OImaging interface, this is triggered by the L1L2WHITE choice.
4. The **Total Variation** (*totvar*) regularization, which tends to avoid having many small variations in the image but does not object to have a small number of large

variations. This preserve extended shapes (making them more uniform than they probably are) and does not remove isolated spikes (e.g., stars). *totvar* is a simple and quite successful regularization for astronomical image restoration.
Using the OImaging interface, this is triggered by the `TOTVAR` choice.

5. The **soft_support** regularization uses an additional image (map) and forces the reconstructed image towards the shape of this *support* (e.g., a star uniform disk). Since the interchange format for image reconstruction algorithms does not implement this case, this regularization will probably be absent from OImaging. Associated parameters are MU_SUPPORT, FHM and MEAN_O. Refer to the JMMC documentation for more information on this regularization.
Using the OImaging interface, this is triggered by the `SOFT_SUPPORT` choice.

7.4. STOPPING CRITERION

The stopping criterion occurs after either NBITER iterations or when the THRESHOLD has been attained:

- NBITER is the maximum number of iterations for the reconstruction, 500 by default. For a better control of the reconstruction, one should rather use `THRESHOLD` below and not lower this value.
- THRESHOLD is the convergence threshold to be used as a stopping criterion for the iterations. It is by default set to the machine precision in simple precision (around 10^{-7}), even if computations are done in double precision. For a (rather) quick-look result, set to a smaller value, e.g., 10^{-6}.

7.5. OUTPUT PARAMETERS

`WISARD` in its standalone version outputs at the end of a minimization cycle the reconstructed image and an IDL structure, AUX_OUTPUT, described in the file wisard.pro of the distribution and containing all relevant information. The `wisardgui` procedure used in conjunction with the OImaging GUI reads and writes `OIFITS` files according to the interoperability norm described in the Interface to Image Reconstruction document (https://github.com/emmt/OI-Imaging-JRA/blob/master/doc/interface/OI-Interface.pdf)

7.5.1. GALLERY

The following figures are examples of what can be expected from `WISARD` depending from the regularization used (and the peculiarities of the object-to-be reconstructed).

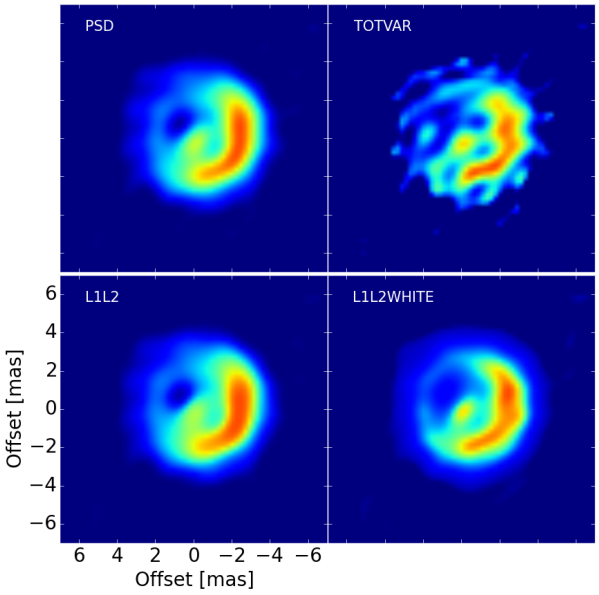

Figure 7.1: Four reconstructed images of the same object as in previous chapters, stopping criteria is 10^{-6}, from left to right and top to bottom: PSD, TOTVAR, L1L2 and L1L2WHITE. L1L2WHITE give the best results in this case (smooth object wit a spike at the center), and TOTVAR the worst.

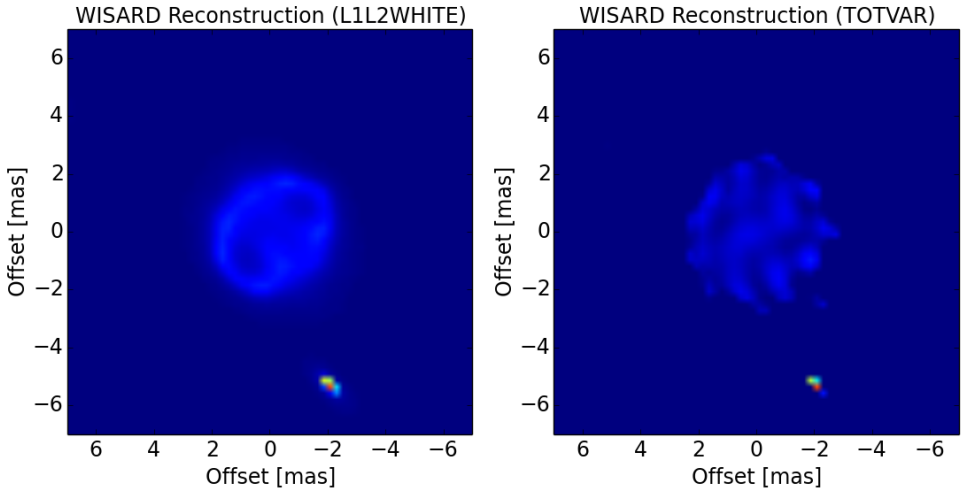

Figure 7.2: Two reconstructed images of a binary: a bright unresolved source near to the large uniform disk of a supergiant. The scaling of the image is logarithmic as to show the disk of the supergiant star. Left: TOTVAR, right L1L2WHITE. TOTVAR is closer in this case to the flatness of the supergiant surface and the compactness of the unresolved star.

8

CONCLUSIONS

- Restoring milliarcsecond resolution interferometric images in the infrared will represent a key facility for the new generation of optical interferometers. During the last decade, the community has gain experience by developing a number of different softwares and innitiatives (e.g., the Interferometric Imaging Beauty Contest). The packages, here described, are the result of an extensive collaborative effort and represent a reliable basis to properly recover images from interferometric data.
- The current understanding of the image reconstruction problem together with today's development of software allow the user to perform both mono-chromatic and poly-chromatic image reconstructions of simulated and real interferometric data. However, algorithms to perform full poly-chromatic reconstructions are still under development and future improvements of the existing software should move towards this field.
- Testing the image capabilities of the different imaging algorithms have been essential to have a full description of them. This agrees with the main goal of this report, which consists in providing a simple an homogeneous view of image reconstruction in optical interferometry to the community, by having complete cookbooks of the different packages as well as a dedicated GUI to use them.
- The better we understand the requirements to achieve a science-grade images from interferometric observations, the easier will be to provide tools and procedures to the community to make more accessible the use of the current techniques. This is a task that should be addresses in the coming years as part of an effort to broaden and engage the field with more members of the international community.

REFERENCES

Baron, F. (2016). Image reconstruction in optical interferometry: An up-to-date overview. In Boffin, H. M. J., Hussain, G., Berger, J.-P., and Schmidtobreick, L., editors, *Astrophysics and Space Science Library*, volume 439 of *Astrophysics and Space Science Library*, page 75.

Baron, F., Monnier, J. D., and Kloppenborg, B. (2010). A novel image reconstruction software for optical/infrared interferometry. *Optical and Infrared Interferometry II, SPIE*, 7734:77342I.

Baron, F. and Young, J. S. (2008). Image reconstruction at Cambridge University. *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 7013:3.

Bose, N. K., Lertrattanapanich, S., and Koo, J. (2001). Advances in superresolution using the L-curve. *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2:433–436.

Duchene, G., Berger, J.-P., Duvert, G., Zins, G., and Mella, G. (2004). ASPRO-VLTI: a JMMC software dedicated to the preparation of VLTI observations. *Proc. SPIE*, 5491:611–616.

Duvert, G., Berio, P., and Malbet, F. (2002). ASPRO: a software to prepare observations with optical interferometers. *Proc. SPIE*, 4844:295–299.

Duvert, G., Young, J., and Hummel, C. (2015). OIFITS 2: the 2nd version of the Data Exchange Standard for Optical (Visible/IR) Interferometry. *ArXiv e-prints*.

Gomes, N., Garcia, P. J. V., and Thiébaut, É. (2017). Assessing the quality of restored images in optical long-baseline interferometry. *Monthly Notices of the Royal Astronomical Society*, 465(4):3823–3839.

Gonzalez, J.-F., Laibe, G., Maddison, S. T., Pinte, C., and Ménard, F. (2015). "ALMA images of discs: are all gaps carved by planets?". *Montly Notices of the Royal Astronomical Society*, 454:L36–L40.

Hansen, P. C. (1992). Analysis of discrete ill-possed problems by means of the L-curve. *SIAM Review*, 34:561–580.

Ireland, M. J., Monnier, J. D., and Thureau, N. (2006). Monte-Carlo imaging for optical interferometry. *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 6268:1.

Lawson, P. R., Cotton, W. D., Hummel, C. A., Monnier, J. D., Zhao, M., Young, J. S., Thorsteinsson, H., Meimon, S. C., Mugnier, L., Le Besnerais, G., Thiebaut, E., and Tuthill, P. G. (2004). "The 2004 Optical/IR Interferometry Imaging Beauty Contest". *Bulletin of the American Astronomical Society*, 36:1605.

Lopez, B., Antonelli, P., Wolf, S., Lagarde, S., Jaffe, W., Navarro, R., Graser, U., Petrov, R., Weigelt, G., Bresson, Y., Hofmann, K. H., Beckman, U., Henning, T., Laun, W., Leinert, C., Kraus, S., Robbe-Dubois, S., Vakili, F., Richichi, A., Abraham, P., Augereau, J.-C., Behrend, J., Berio, P., Berruyer, N., Chesneau, O., Clausse, J. M., Connot, C., Demyk, K., Danchi, W. C., Dugué, M., Finger, G., Flament, S., Glazenborg, A., Hannenburg, H., Heininger, M., Hugues, Y., Hron, J., Jankov, S., Kerschbaum, F., Kroes, G., Linz, H., Lizon, J.-L., Mathias, P., Mathar, R., Matter, A., Menut, J. L., Meisenheimer, K., Millour, F., Nardetto, N., Neumann, U., Nussbaum, E., Niedzielski, A., Mosoni, L., Olofsson, J., Rabbia, Y., Ratzka, T., Rigal, F., Roussel, A., Schertl, D., Schmider, F.-X., Stecklum, B., Thiebaut, E., Vannier, M., Valat, B., Wagner, K., and Waters, L. B. F. M. (2008). "MATISSE: perspective of imaging in the mid-infrared at the VLTI". *Optical and Infrared Interferometry*, 7013:70132B.

Lopez, B., Lagarde, S., Wolf, S., Jaffe, W., Weigelt, G., Antonelli, P., Abraham, P., Augereau, J.-C., Beckman, U., Behrend, J., Berruyer, N., Bresson, Y., Chesneau, O., Clausse, J. M., Connot, C., Danchi, W. C., Delbo, M., Demyk, K., Domiciano, A., Dugué, M., Glazenborg, A., Graser, U., Hanenburg, H., Henning, T., Heininger, M., Hofmann, K.-H., Hugues, Y., Jankov, S., Kraus, S., Laun, W., Leinert, C., Linz, H., Matter, A., Mathias, P., Meisenheimer, K., Menut, J.-L., Millour, F., Mosoni, L., Neumann, U., Niedzielski, A., Nussbaum, E., Petrov, R., Ratzka, T., Robbe-Dubois, S., Roussel, A., Schertl, D., Schmider, F.-X., Stecklum, B., Thiebaut, E., Vakili, F., Wagner, K., Waters, L. B. F. M., Absil, O., Hron, J., Nardetto, N., Olofsson, J., Valat, B., Vannier, M., Goldman, B., Hönig, S., and Cotton, W. D. (2009). "Matisse". *Astrophysics and Space Science Proceedings*, 9:353.

Malbet, F., Cotton, W., Duvert, G., Lawson, P., Chiavassa, A., Young, J., Baron, F., Buscher, D., Rengaswamy, S., Kloppenborg, B., Vannier, M., and Mugnier, L. (2010). "The 2010 interferometric imaging beauty contest". *SPIE: Optical and Infrared Interferometry II*, 7734:77342N.

Meimon, S., Mugnier, L. M., and le Besnerais, G. (2005). Convex approximation to the likelihood criterion for aperture synthesis imaging. *J. Opt. Soc. America A*, 22:2348–2356.

Mella, G. and Duvert, G. (2004). A new GUI system for ASPRO. *Proc. SPIE*, 5496:582–589.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer Verlag, 2nd edition.

Pauls, T. A., Young, J. S., Cotton, W. D., and Monnier, J. D. (2005). A Data Exchange Standard for Optical (Visible/IR) Interferometry. *PASP*, 117:1255–1262.

Sanchez-Bermudez, J., Thiébaut, E., Hofmann, K.-H., Heininger, M., Schertl, D., Weigelt, G., Millour, F., Schutz, A., Ferrari, A., Vannier, M., Mary, D., and Young, J. (2016). The 2016 interferometric imaging beauty contest. *Proc. SPIE*, 9907:99071D–99071D–18.

Skilling, J. and Bryan, R. K. (1984). Maximum Entropy Image Reconstruction - General Algorithm. *MNRAS*, 211:111.

Strong, D. and Chan, T. (2003). *Inverse Problems*, volume 19.

Thiébaut, E. (2008). "MIRA: an effective imaging algorithm for optical interferometry". *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 7013:1.

Thiébaut, E. (2009). Image reconstruction with optical interferometers. *New A Rev.*, 53:312–328.

Thiébaut, É. (2013). *Principles of Image Reconstruction in Interferometry*, volume 59. EAS Publications Series; eds: Mary, D. and Theys, C. and Aime, C.

Thiébaut, E. and Giovannelli, J.-F. (2010). Image reconstruction in optical interferometry. *IEEE Signal Processing Magazine*, 27:97–109.

Thiébaut, É. (2009). Image reconstruction with optical interferometers. *New Astronomy Reviews*, 53:312–328.

Thiébaut, É. and Giovannelli, J.-F. (2010). Image reconstruction in optical interferometry. *IEEE Signal Process. Mag.*, 27(1):97–109.

Tikhonov, A. N. and Arsenin, V. Y. (1977). Solutions of Ill-Posed Problems. *Mathematics for computation*, 32:1320–1322.

Young, J. and Thiébaut, É. (2015). Unified image reconstruction description. Technical Report Deliverable 4.1 of OPTICON WP4, OPTICON.