Département Sciences de la matière
École Normale Supérieure de Lyon
Université Claude Bernard Lyon I

Yohann Faure
May–July 2019
M1 SdM

Development of programming tools for

Protoplanetary Disk Fitting

RXJ1615.3-3255

Abstract: *In the context of an ever-increasing angular resolution in astronomy, the need for good fitting process and algorithm keeps increasing, especially when dealing with object such as protoplanetary disks. This internship report is focused on the development of different tools and techniques related to the fitting of experimental data gathered by* Alma*, applied to the example of* RXJ*1615.3-3255.*

The proposed models for J1615 are obtained by fitting Alma *interferometric data using different methods: Image Plane Fitting, Interferometric Visibilities Fitting and Radiative Transfer Modeling. All the methods I implemented are based on the Monte-Carlo Markov Chain (*MCMC*) optimization process and gradient descent algorithms.*

The main results of this internship consist in a set of adaptive and generalist Python procedures, optimized for highly parallelized computation and human understanding. Several models for the J1615 emission profile are given as an example and as a result of these codes.

Keywords: Astrophysics, RXJ1615.3-3255, Alma, Model Fitting, Python.

Internship supervised by
Myriam Benisty, Astronome-Adjointe à l'Observatoire de Grenoble, Unité Mixte Internationale.
Professor **Laura Perez**, Universidad de Chile.

Acknowledgements

This Chilean adventure would not have been possible without certain people that I would like to thank.

I want to express my deepest appreciation to my two advisors, Myriam and Laura, for their helpful advice and relentless support all through the internship, as well as for the valuable knowledge they passed on to me. I hope we will meet and work together again.

I would like to extend my sincere thanks to Marion and Nicolás, who faced with me the computer mess associated with the development of all these tools, and helped me solve them with infinite patience; and to Léonard, one of the best flatmates I've ever had.

Thanks also to all the Calán staff, all the PhD students, all the professors and researchers. You have been a fantastic team to work and live with, and I sincerely hope we will meet again.

I finally gratefully acknowledge the effort of the internships evaluation committee, as well as Guillaume Laibe's commitment and help throughout the past year.

Gracias a tod@s !

Contents

1 Introduction and Summary

Astrophysics, as the branch of physics concerned with the physical nature and behavior of stars and stellar objects, relies heavily on computer science, and is often particularly resource-hungry. The study of exoplanets and protoplanetary disks is by no means an exception to that, as I discovered during my internship.

My main goal was initially set on the study and modeling of a protoplanetary disk around a young star, RXJ1615.3-3255 (short-called J1615). The first images of that disk obtained using SPHERE [1] revealed a fine structure, constituted of multiple intricate levels of rings. The ALMA observations, which I studied during this internship, revealed a similarly complex structure at a different wavelength, as shown in Figure 1. I specifically focused on the data set that led to the band 7 image ($0,8-1,1$mm, on the Left of Figure 1).

The fitting process is based on multiple techniques, using first-order iterative optimization algorithms (gradient descent) as implemented in the Python library SciPy [2, 3, 4], and Markov chain Monte Carlo optimization methods [5] (sections 5,7). The models have to respect some physical constraints, and to be both accurate and realistic. I ended up using compound models, composed of a (Gaussian) core and multiple different rings.

Such methods often require a long running time, and optimization was one of my main objectives. By using more powerful and CPU efficient methods, as well as GPU parallelization and multiple-nodes clustering, I achieved a good complexity reduction as well as a significant time optimization in codes previously used by the research team I worked with. The major gains stem from the replacement of the Multiprocessing module [6] by the MPI4PY implementation of MPI [7, 8, 9], ideal for large clustering computation, and by the use of CUDA acceleration with the Python module Galario [10] (section 9).

This report presents the J1615 object, the different imaging tools, the fitting and optimization process, and the side projects I developed during my internship at the Calán observatory.

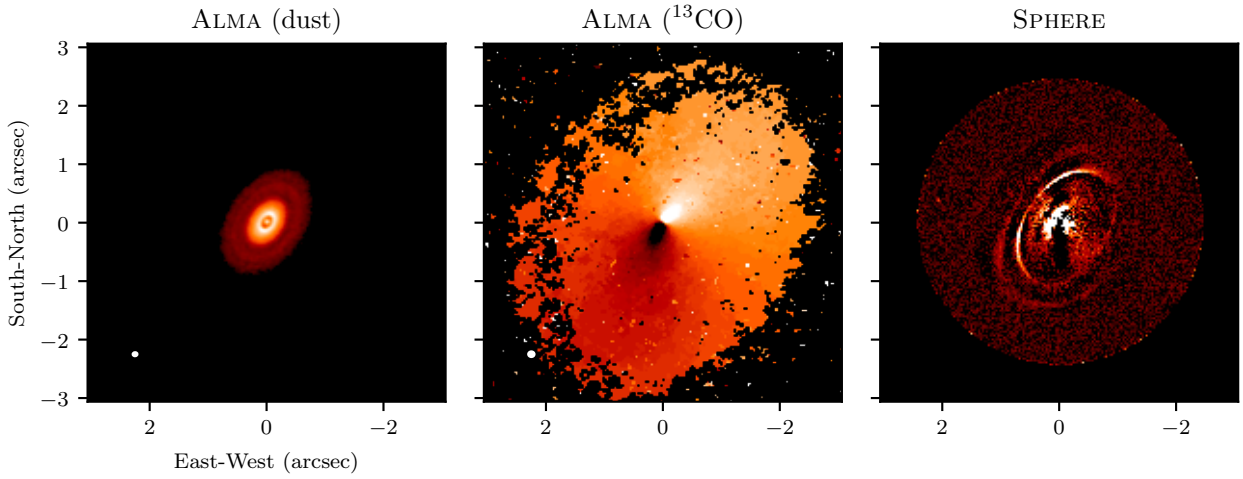

Figure 1: J1615 images obtained with different methods, the ALMA resolution beam is in the bottom left corner. *Left*: ALMA, band 7 image, associated with dust and pebbles. *Middle*: Alma millimetre-wavelength ^{13}CO observation, first moment. Right: SPHERE $1,593-1,667\mu$m image [1]. Both Left and Middle were reconstructed from visibilities using CASA [11].

2 Context of this internship

2.1 Why Chile?

Chile is one of the leading countries in astronomy and astrophysics, especially as it has one of the best observation spots in the world. With an average rainfall of 15mm per year, altiplanos 2500m above sea level, and quasi non-existent light pollution, the Atacama desert is one of the best places to watch the sky. For that reason, it hosts most of the biggest telescopes in the world, such as the Very Large Telescope (VLT), the Atacama Cosmology Telescope (ACT), or the Atacama Large Millimeter Array (ALMA).

Being such a great astronomical observation spot, Chile hosts many astrophysics laboratories, such as the Europe Southern Observatory (VLT), one of the organizations in charge of the good operation of ALMA.

2.2 Cerro Calán

The Cerro Calán national observatory (see Figure 2), founded in 1852, is one of the oldest observatories in South America. Located on the top of a hill near Santiago, it is now a research facility, hosting international meetings every Thursday, and offering a workplace to astronomers from all around the world. As the home of the Unité Mixte Internationale (UMI), it was my workplace for this internship.

Figure 2: One of the five domes of the Cerro, 853m above sea level.

2.3 ALMA and my data set

The Atacama Large (sub)Millimeter Array (ALMA) is a giant radio-interferometer, built in the Atacama desert, 5000 meters above sea level. It is composed of a set of 66 antennas of different size, capable of independently pointing in every direction of the observable sky, and therefore creating a giant interferometer.

Since October 3, 2011 (ALMA's "birthday"), this giant telescope has permitted the (re)discovery of many stellar objects. Its brightest achievements are the high definition imaging of HL Tau's protoplanetary disk [12], highly detailed observations of Einstein's rings, and the observation of spectra related to organic molecules such as sugars and alcohols in a large part of the sky [13].

Figure 3: One of the 66 ALMA's radio-antennas, 5000m above sea level, by Iztok Bončina (ESO).

The data I studied during this internship comes from a compilation of ALMA observations conducted by Laura Perez and Myriam Benisty, since July 2014.

2.4 Hardware and Software used

I had to use different software in this internship. The main programming language I used was Python 3, with the scientific libraries SciPy and NumPy, Astropy, Emcee, Multiprocessing, MPI4PY and Galario [2, 3, 4, 5, 6, 9, 10, 14]. I also used the CASA software to generate images from ALMA data [11] and MCFOST for radiative transfer simulation [15].

The machines I had to use to run my computation were my own computer (Intel i7-4710HQ processor, 16GB RAM, NVIDIA GTX850M GPU), and a local computation server (dual Intel Xeon E5620, 16GB RAM, no dedicated GPU) named Alma. I was also granted an access to up to 120 CPU cores on the Leftraru cluster (Intel Xeon Gold 6152 and E5-2660), with up to two NVIDIA Tesla V100 GPUs, and up to 2TB of RAM.

3 Interferometry

In this section, I will briefly introduce the theoretical concepts associated with astronomical interferometry.

3.1 Basic principles

Let's consider two identical sine waves, with same amplitude and frequency. The superposition of those two waves depends only on the phase shift between them. For example $\sin(\omega t) + \sin(\omega t + \pi) = 0$, while $\sin(\omega t) + \sin(\omega t + 0) = 2sin(\omega t)$, as shown in Figure 4. The first is said to be *in anti-phase*, the interference is destructive, there is no resultant wave. The second is said to be *in phase*, the interference is constructive, the resulting amplitude is twice as big as the amplitude of the two incident waves.

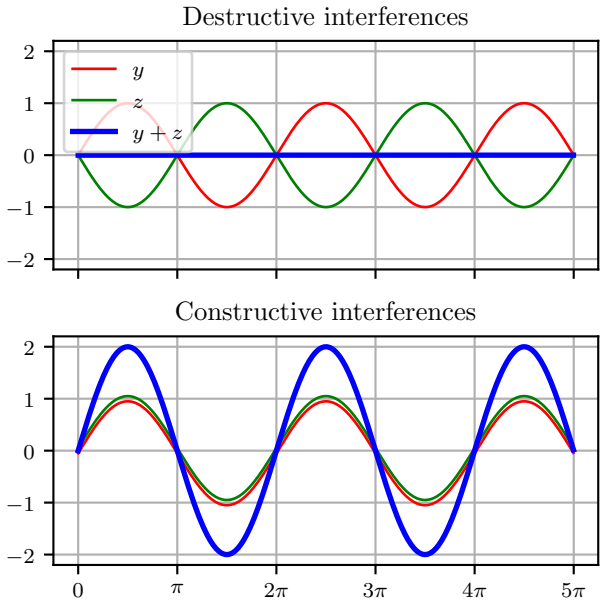

Figure 4: Illustration of destructive and constructive interferences.

In this example, we consider the simplified case of light waves with the same amplitude and frequency, but natural sources emit radiations at multiple wavelengths, and with different amplitudes. When doing interferometry, we need to have relatively close signals in order to measure their interference. This criteria is the *coherence* between the two signals. Coherent waves can be generated by two sources emitting at the exact same frequency, or by splitting light coming from a single source.

Astronomical interferometers usually consist in two or more separate telescopes that combine their signals, as if they were splits in a cosmic Young splits experiment. Assuming that there is no impact of atmospheric fluctuations on the incoming waves (or limiting them by climbing the Andes), the difference in travel time comes mostly from the physical distance between the telescopes, that we know, and from the source itself, that we want to measure.

3.2 Fourier transform and (u, v) plane

In this section I will enunciate the principles behind interferometry. For more details on the math of interferometry and Fourier transform, read this paper from Andreas Glindemann [16].

The main interest of interferometric measurements is that combining many telescopes allows us to have a full map of interferometric complex intensities, called visibilities. Each pair of telescopes gives one point in a plane, described by the position vector between them. These vectors can be described in a 2D base with two coordinates called (u, v), hence the name (u, v) plane. The more antennas there is, the more points in the plane there is, and if n is the number of antennas, the number of points is $\binom{n}{2} = \frac{n(n-1)}{2}$.

Each point in this plane, assuming mathematical perfection or sufficient correction, is actually a point in the Fourier transform of the image of the object emitting the light. The complete mathematical link between the complex visibility ν and the brightness distribution I of the object is a Fourier transform as follows:

$$I(\alpha, \beta) = \iint_{\mathbb{R}^2} \nu(u, v) \exp\left(2i\pi\left(\alpha u + \beta v\right)\right) \mathrm{d}u\mathrm{d}v$$

where (α, β) correspond to the angular coordinates in the sky, and (u, v) the coordinates of the bases used to get the point, describing the spatial frequencies of the brightness distribution.

The visibility function can give information on physical properties of the source. For example, the shape of the source can be deduced directly from the shape of the visibility curve. With an appropriate algorithm, it is then possible to convert the (u, v) plane into an image plane, but many precautions have to be taken, such as weighting the points obtained according to the degree of perturbation they suffered. The reference software for such conversion is CASA, which can also convert simulated images to visibilities, taking ALMA's imperfections into account [11].

3.3 ALMA's resolution

As they recombine their signal, the antennas of an interferometer fake one single dish telescope the size of the largest baseline between telescopes.[1] The largest separation of individual telescopes gives the resolution of the telescope, which allows interferometers to have an incredibly high angular resolution.

Indeed, while the angular resolution $\Delta\theta$ of a single dish telescope is limited by diffraction, and therefore by the size of its mirror, $\Delta\theta \propto \frac{\lambda}{D}$, with λ the wavelength observed and D the diameter of the telescope, interferometers are only limited by the distance you can put between two bases (using the exact same equation). ALMA for example can reach baselines of up to 16km.

ALMA consists in 66 high precision antennas that can be arranged in different configurations over the year. The maximum distance between antennas can vary from 150 meters to 16 kilometers. The variation of baseline allows ALMA to observe with different spatial resolutions. In the most compact configuration, resolutions range from 0.5 arc-second (as) at 950GHz to 4.8as at 110GHz, while in the most extended configuration they range from 20 milli-arc-seconds (mas) at 230GHz to 43mas at 110GHz. These high spatial resolutions are an important step forward to resolve small objects, such as protoplanetary disks.[2]

[1] It is important to note, however, that a single dish telescope of this size would catch countless more photons.

[2] The Event Horizon Telescope, consisting of an array of interferometers all around Earth, including ALMA, allows for baselines as large as the Earth itself, and has a theoretical angular resolution of up to 25µas.

4 Protoplanetary disks

The birth of a stellar system from a dust and gas cloud is a complex process that can take many shapes, amongst which are protoplanetary disks. This sections give some insight in the Science of protoplanetary disks, but doesn't go deep into theoretical considerations. For more details, you can refer to *Protoplanetary Disks and Their Evolution* [17].

4.1 What is a protoplanetary disk?

All across the galaxy, we can observe gas and dust clouds of gigantic radius, that slowly collapse due to their own gravitational pull. When such cloud collapses, it can create a gas ball large enough to start hydrogen fusion in its core. The ball starts shinning and is then called a star.

The conservation of angular momentum and the dramatic collapse of the ensemble makes it rotate fast, creating a favored symmetry plan, around which a disk structure can appear, as shown in the Figure 5. This structure is called a protoplanetary disk.

Those gas and dust disks became clearer with time as the angular resolution in astronomy increased. Stars that used to be seen as point sources with a non-standard light emission revealed much more complex structures, such as stellar systems, binary stars of gas and dust disks. The recent launch of ALMA gave us new insight in the protoplanetary disk structure.

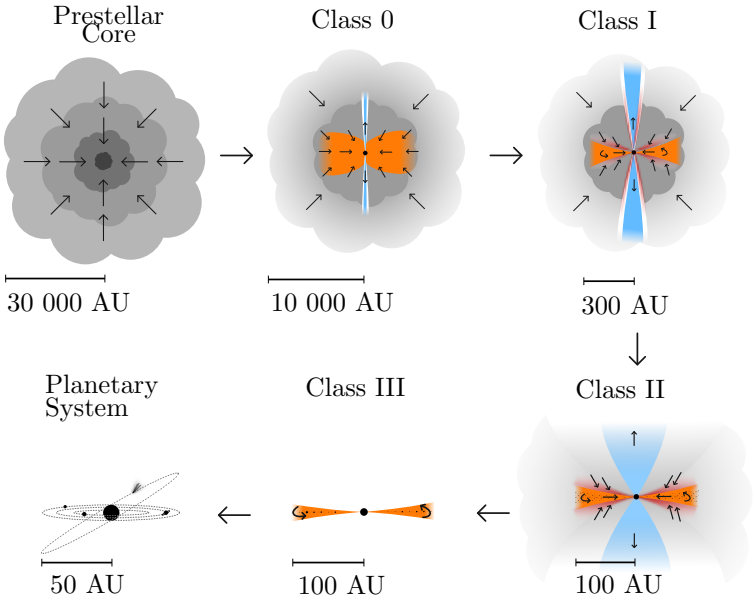

Figure 5: The different steps of stellar formation. Considering as time origin the star formation (Class 0), Class I usually appears in a rather short time (<0.03Myr). Class II, corresponding to the J1615 phase, appears at around 1Myr. Class III appears after 10Myr. Finally the planetary system phase (corresponding to our solar system for example) appears after around 20Myr [18].

4.2 Protoplanetary Disks Physics

4.2.1 Composition

The composition of a disk is similar to that of the interstellar medium, *i.e.* small dust grains and gas. One of the main characteristics of a disk is its gas-to-dust ratio, usually assumed to be roughly the same as interstellar medium, *i.e.* 100.[3] This assumption is more of a consensus, as it is very hard to precisely determine such ratio because of optical thickness and chemical composition variation.

The dust grains are usually considered to be small silicate and graphite grains, while the gas is mainly H_2 (not very useful for the observations as it needs to be very hot to emit light, and therefore very close to the star), with traces of CO and H_2O, easier to track. CO is the second most abundant species after H_2, the CO–H_2 ratio being about 10^{-4}.

[3] Even though recent observations show high variation in this ratio [19].

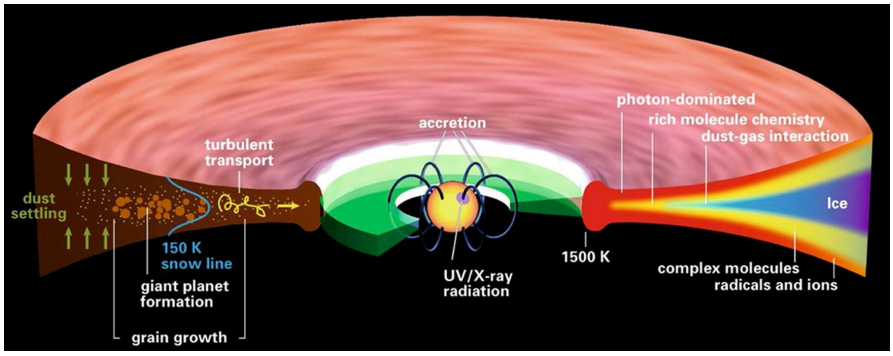

Figure 6: Schematic representation of a protoplanetary disk structure around a 1–5Myr Sun-like star (from T. Henning and D. Semenov [20]).

4.2.2 Light emission

The emission from a star with a protoplanetary disk is due to several sources. First there is the star emission, depending on the star properties. Then there is the disk emission, that can be split into two. The continuum is a black-body radiation due to the disks temperature, emitted by the dust grains. The gas (mainly H_2, ^{13}CO and H_2O) also has specific emission rays. Some minor effects also come into play, such as light scattering. Scattering induces polarization, which is also useful in disk characterization.

The SED, shown in Figure 7, shows J1615 emission in a wide panel of wavelengths.

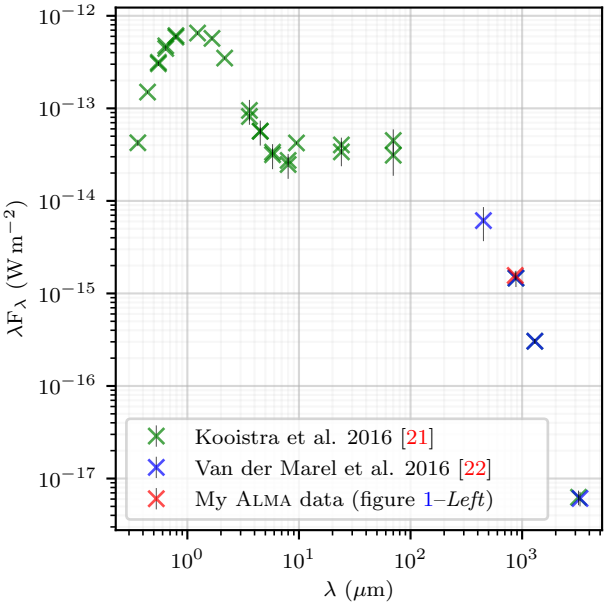

Figure 7: J1615 Spectral Energy Distribution (SED). This plot represents the energy emitted by the disk at each wavelength. See appendix A.1 for the computation of one SED point.

4.2.3 Disk Structure

The protoplanetary disk "geometric" structure is not as simple as just a flat disk (as shown in Figure 6). Under the assumption of an ideal gas disk, the combination of the force equilibrium and the state equation gives its vertical density. Considering an azimuthally symmetric disk in hydrostatic equilibrium, with uniform temperature along the z axis, with r the radius and z the vertical height, we obtain equation 1. Often, H and Σ are given parametric values, described in equation 2. This allows for simpler fitting but is not a physical result.

The characteristic height H is given by $H = c_s/\Omega$, where c_s is the sound speed and Ω the angular rotation speed. $\Sigma(r)$ is the surface density of the disk, with γ the surface density exponent, usually negative. $H(r)$ is the scale height, with β the flaring angle, between 1 and 1,25 [23].

$$\rho(r,z) = \rho(r,0) \times \exp\left(-\frac{z^2}{2H^2}\right) \tag{1}$$

$$H(r) = H_{\text{out}} \times (r/r_{\text{out}})^{\beta} \tag{2}$$

$$\Sigma(r) = \Sigma_{\text{out}} \times (r/r_{\text{out}})^{\gamma}$$

4.3 The need to fit data

When observing a disk (using ALMA for example), complex substructures can appear, such as gaps, local minimum or maximum in the emission, spirals, etc. In order to measure physical values and to verify theories, we need a clean modeling of the disk. The idea behind the modeling choice is to use a relatively simple model that we theoretically created with a physical interpretation, and to fit it with the data.

Observing the residual values (often called the *residuals* of a model), one can then deduce the presence of specific elements in the disk. For example, a gap is often associated with the possible existence of a protoplanet. An asymmetry in the residuals can also be associated with more complex non-resolved structures. Fitting allows for both a precise characterization of the structure and composition of the disk and the discovery of low intensity details that could have a physical meaning.

4.4 J1615

J1615 is a Class II disk located in the constellation of the Scorpius, 600 light-years from earth. According to the Gaia database, it has a right inclination of 244°, a declination of -33°, and a parallax of 6.34mas. Its near-visible light magnitude is of 11.5, which is far above the human eye perception limit. A first look at the Figure 1 shows that the structure of the disk is far more convoluted and complex than the model presented in Figure 6.

The multiple gaps in the structure makes this disk a good candidate for the discovery and observation of exoplanets, as well as a real fitting challenge that I took on the best I could, both in the image plane and the interferometric visibilities plane.

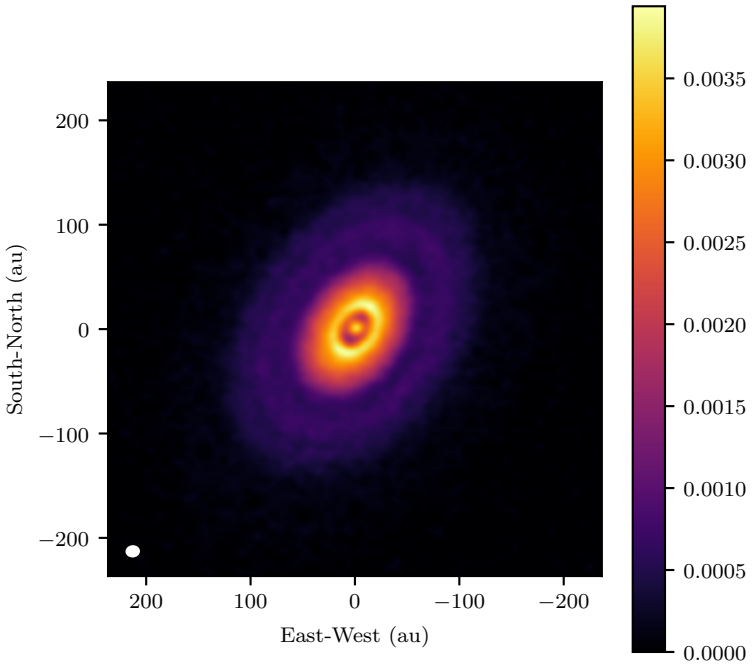

Figure 8: The J1615 Disk image observed by ALMA and generated with CASA, in mJy/beam. The beam size is given by the white ellipse on the bottom left corner.

5 Image plane Fitting

The principle of the image plane fitting is to use a combination of simple mathematical functions to recreate an image as similar to the observed image as possible. By simple functions, I mean functions that are easy to compute and known to approximate images generated by gas and dust distributions around a star, the Gaussian being the most popular choice. This section describes the full image fitting process, the implementation details being given appendix A.3.

5.1 The issues of the image plane

In order to obtain an image such as the one shown in Figure 8, the ALMA interferometric data has to be transformed using for example CASA [11]. This process is a complex computation based on the Fourier transform, that aims at reducing background noise and measure errors. There is therefore inherent loss of information. Yet it is an excellent method for both training and having a first idea of the real disk structure behind the image.

It is also to be noted that when using the image plane, we have to consider the convolution with the observation beam of ALMA. That beam, represented as a white ellipse on Figure 8, corresponds to the smallest thing we can resolve. In order to take in into account, an image generated from a model can be convolved with a Gaussian-like profile, the size of the beam. For this first approach, I decided not to convolve and to fit as if no convolution was made.

5.2 Deprojection

The first step in that fitting is to find, if it is relevant, the inclination and position angle of the disk. The rotation of the disk gives it a reference symmetry plane, which is generally different from the observation plane. To describe the tilt of this plane to the observation plane, we only need 2 angles, the inclination (inc) and the position angle (PA).

To find these angles, the easiest way is to consider the disk as a tilted perfect Gaussian, and to minimize the difference between this tilted ideal Gaussian and the real image. I developed a program that does that for any disk (appendix A.2). The idea behind it is to determine a new set of coordinates for each pixel, in order to determine accurately a radius map.

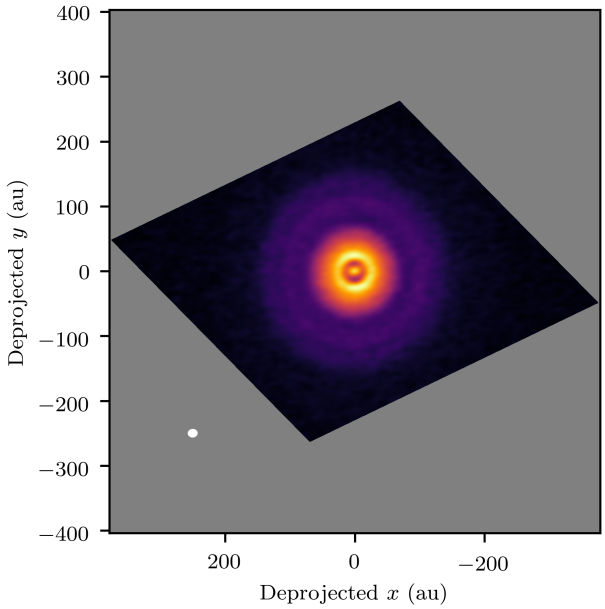

Figure 9: Deprojected J1615 Disk. This deprojection is obtained using (inc,PA)=(45.98°, -34.25°).

The result of the J1615 deprojection is shown in Figure 9. Once the deprojected image is done, it is easier to determine the structure of the disk, as well as the precise radius of every sub-disk. Although this operation is not physically accurate (the disk has thickness), it gives us useful insight for the next step: the features extraction.

5.3 Features extraction

Now that the distance of every pixel to the center is known, it is possible to plot the brightness intensity in functions of radius, as well as the intensity profile along one precise radius. These operations allows to get information on the general radial structure, and on the asymmetry of the disk.

5.3.1 Radial profile

We can see that the structure is quite symmetric, and can therefore as a first approximation be described by only a radial profile. To extract this profile, I decided to use binning on the radius map computed with the deprojection, and to use as error bars the standard deviation of each bin. The result is shown in Figure 10, and can be interpreted as follows. A first model one could use for J1615, based on that observation, would be a Gaussian central dot, followed by a Gaussian ring centered on ∼25au, and two large Gaussian rings at ∼100au and ∼130au. But a slight bump around 60au can already be noticed, and will ultimately have to be tackled.

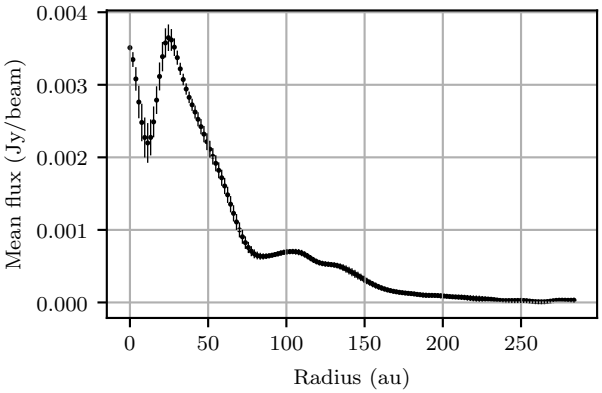

Figure 10: Radial profile of J1615 disk, with a bin size of ∼2au.

5.3.2 Angular profile

The angular profile is a scatter plot of all pixels associated with a radius comprised between two values. It reveals the asymmetries of the disk, as shown in Figure 11, and in this video I made. This profiling shows that there is up to 20% asymmetry in this disk. It might be due to the beaming effect, but can also be a physical result that need to be taken into account.

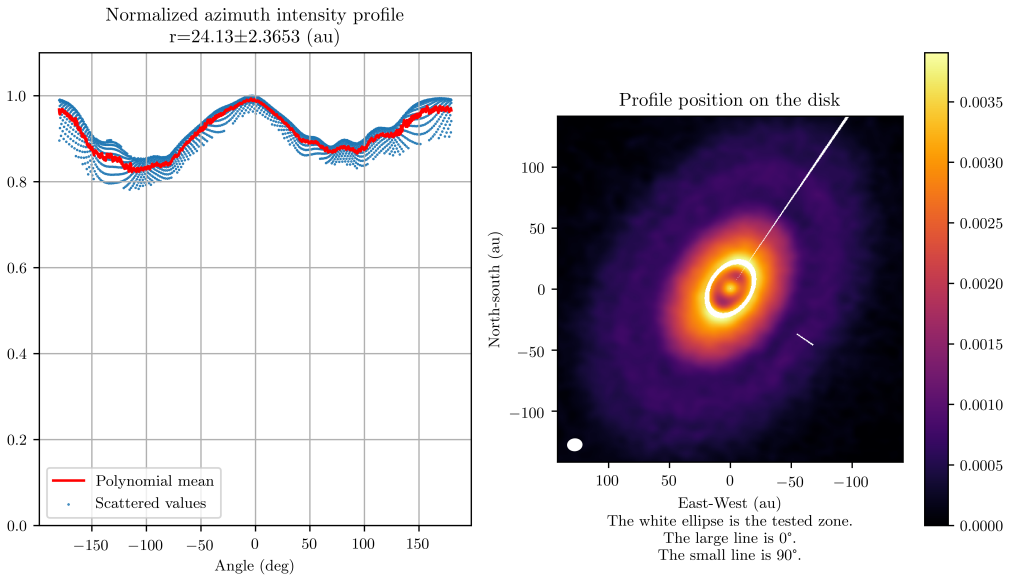

Figure 11: A Screenshot of the video I made showing the angular profiles at different radii.

5.4 Model choice

Considering that profiles, the first model I chose was the sum of a central Gaussian, a large central Gaussian ring (∼20au), and two external Gaussian rings (∼80au), all concentric and all perfectly circular. I'll refer to these structures as G0 for the center and GR1, GR2 and GR3 for the three rings.

After the first attempt at an optimization (more details in section 6), I had to opt for different functions and to add degrees of freedom. I mainly replaced the first ring by a different more complex one based on the literature [24], and gave freedom to the center, the inclination and the position angle of all rings, for a total of 37 degrees of freedom.

6 Optimization

The optimization of the fitting takes three steps: model definition, classical optimization and Monte Carlo optimization.

6.1 Model definition

The implementation of the model is based on Astropy's custom models definition. Using the keyword `custom_model`, one can turn a "normal" function into a function generator, as explained in appendix A.3, and then sum them.

The goal of an implementation is both having a clear and clean code, and an effective code, as it will be called a very high number of time (more than 500 000). For that reason I had to implement myself the functions instead of using the already existing models available in Astropy. (More details about optimization are available in section 9.)

Once the model is defined, one can generate an image based on that model, by calling `Model_J1615(x,y)`, where `(x, y)` is a mesh generated to correspond to the tilt of the disk.

6.2 Classical optimization

To optimize the model I made, I have to define a cost function, quantifying the difference between the real image and the generated image. The first choice was a trivial absolute value difference of the two images, but I soon had to refine it to be able to use boundaries for each parameter of the model. The detailed implementation is available on GitHub.

The process I call *classical optimization* is the use of a Gradient Descent algorithm to minimize the difference between the model and the data. The program optimizes the parameters of the model (angles, center and width of all rings) following the `scipy.optimize.minimize` implementation of Gradient Descent. That process gives a starting point for the Emcee optimization [3, 5].

6.3 Emcee

Emcee is an effective and powerful implementation of the Monte Carlo Markov chain minimization algorithm. This algorithm is based on a semi-random gradient descent, using a set of independent walkers, moving in the parameter space.

A walker is a parameters set, walking in the parameter space as the algorithm goes on. It goes by steps, each step being a semi-random modification of every walker, toward a minimization of the minimal walker cost. As the number n of walkers must be higher than twice the parameter space dimension, and the number m of steps higher with the complexity of the image, the number of model and cost evaluation, $n \times m$ (one per iteration and per walker), increases rapidly with the complexity of the model used.

On the right, in Figure 12, is represented the evolution of three dimensions in the parameter space. It is to be noted that the walkers evolve as an extended cloud in this space, and can therefore define average and percentiles in the parameters value. For more details on the use of Emcee, please refer to appendix A.4.

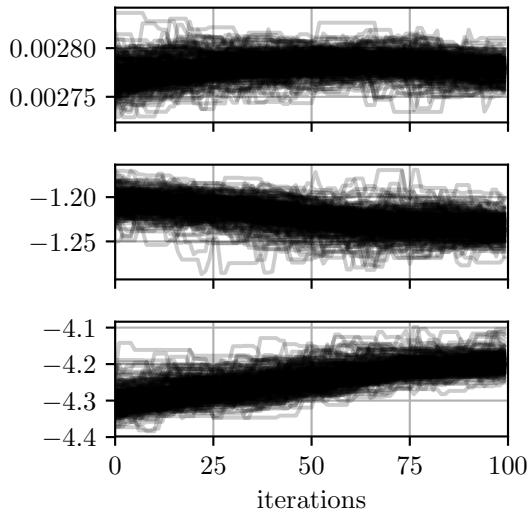

Figure 12: A typical evolution of 300 walkers in three parameters during an Emcee optimization. The parameters shown are the intensity and $(x,\ y)$ coordinates of the central Gaussian G0.

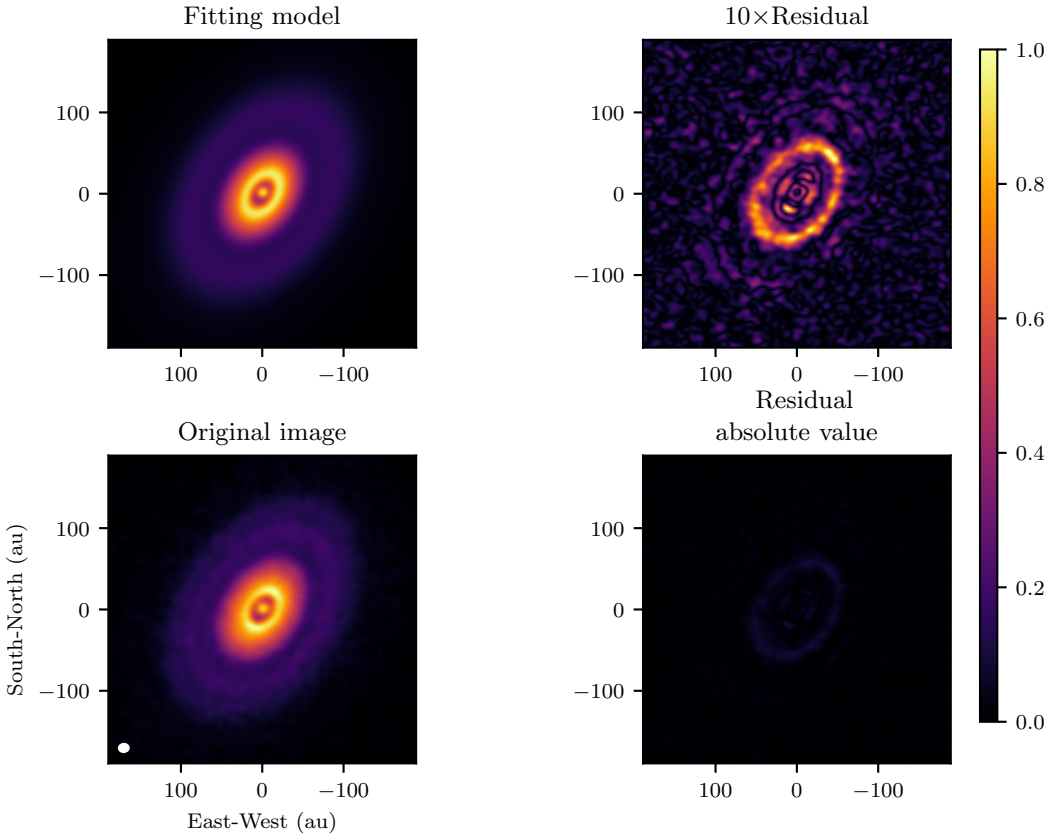

Figure 13: Results of the Emcee optimization, comparing the image with the model, and showing the residual multiplied by 10. The background noise standard deviation is around 2% of the maximum of the image, while the maximum of the residual is around 10%.
The numerical values obtained by emcee are given in Appendix A.5.1.

6.4 Results

The final optimization I made for the image plane had 37 parameters, with 4 rings and a center, all entirely free. I ran it on the local computer Alma, with 300 walkers, in 40 batches of 1000 steps, with trimming of the walkers every 5 batches.[4] As this was done before the optimization process (it actually triggered it), each batch of 1000 steps took approximately 2 hours, giving a total cumulative run-time of 4 days. The definitive convergence value was obtained after 35 batches, and is shown in Figure 13.

The interpretation of this result is complex as the residual is not perfectly symmetric. The residual ring shows that the central disk GR1 is not ideally shaped, and has to be reconsidered. The use of a more flared disk, described by different functions, might be needed. The flower shape of the residuals can also be due to the beam shape and size, and using a direct visibilities fit might get around this issue. There is finally an issue with GR3, as this outer ring ended up fitting a very faint yet quite circular background emission, at a level 3 orders of magnitude smaller than its inner counterpart GR2.

That being said, the residual are lower than 10% of the maximum of the image. It is scientifically significant, as it is around 5 times the background noise standard deviation of the image, but it is still a good result giving useful insight in preparation of the visibilities fitting.

7 (u, v) plane fitting

As the image plane fitting depends on the image generation, as well as on the beam convolution, it is natural to fit the data using the original data set instead. To do so, we need to consider a (u, v) plane fitting using Galario [10].

7.1 (u, v) plane data

When observing with Alma (or any interferometer), each pair of bases used to measure a value is defined by a 2D vector in the observation plane. This vector can be described in a 2-vector base, with coordinates named (u, v). The output data is therefore indexed on that plane, making the visibilities a Fourier transform of the image data.

The visibilities data come with an other indicator: the weight. This indicator is a way for the observer to rate the quality of an observed data point, giving it an error value. It is very useful when fitting a model, because it allows the cost function to consider some points as more important as others. The full data format is shown in Figure 14.

```
#u [m],       v [m],       Re [Jy],     Im [Jy],     weights
-1.5590e+02   2.3435e+02   1.8103e-02   1.3799e-01   2.0006e+02
 9.2907e+00   3.6298e+02  -5.8268e-02   2.8202e-02   2.1695e+02
 9.5235e+01   1.0923e+02   6.3143e-02  -1.6727e-01   1.6719e+02
 ...
```

Figure 14: Data structure of the Alma output. In order the values given are u, v, the real part of the visibility, its imaginary part, and finally its weight.

[4] The trimming process consist in a simple manual elimination of the "lost" walkers, and a change in the boundaries if needed.

$$\chi^2 = \sum_{u,v} \left(\left(\Im\left(Model_{u,v} - Data_{u,v}\right)\right)^2 + \left(\Re\left(Model_{u,v} - Data_{u,v}\right)\right)^2 \right) * w_{u,v}$$

```
def chi2compute(ModelVal,Re,Im,w):
    return(np.sum(((np.imag(ModelVal)-Im)**2.+(np.real(ModelVal)-Re)**2.)*w))
```

Figure 15: The formula used to compute the χ^2 from the data.

7.2 Model fitting

The process is the same as for the image plane fitting, but using the knowledge section 5 gave me, and choosing to work with a perfectly symmetric model, I was able to reduce the number of parameters down to 13, making the optimization process shorter in terms of Emcee iterations. More details about the (heavy) computing requirements are given in section 9.

The main difference with the image plane fitting is that the incomplete (u, v) coverage (that resulted in a beam convolution in the image plane) is perfectly taken into account using Galario [10]. That led the inner Gaussian to become a very point-like source and the inner ring to become thinner. The model choice is still a debate matter at the time I'm writing this line. The fitting result is shown in Figure 16

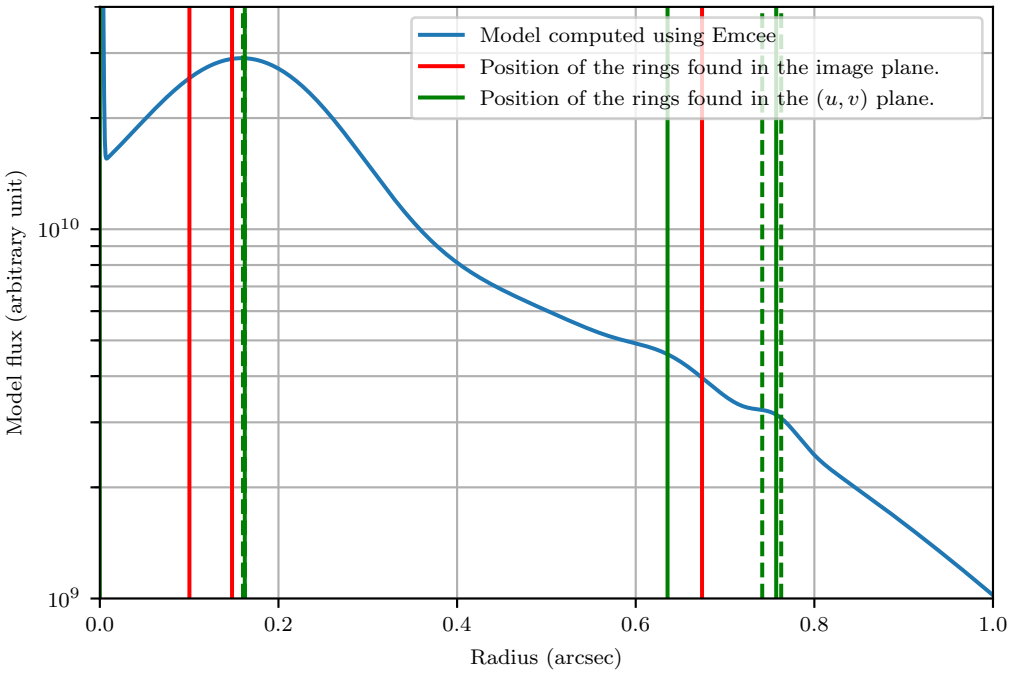

Figure 16: Emcee fitting result, comparing in red the ring positions found using the image plane, and in green the positions found in the (u, v) plane. The dashed lines correspond to the 15% and 85% percentiles of the Emcee optimization, and are mostly hidden behind the solid lines.

The numerical values of the fitting, as well as a discussion about the image plane coherence, are given in Appendix A.5.2.

8 MCFOST

After the model fitting, I decided to use the MCFOST [15] software to find a physical model for the J1615 disk. The motivation behind this project is to shift the approach from a geometrical description to a physical description, based on that geometric properties extracted from the ALMA data.

8.1 Principle

The idea behind MCFOST is to create a dust distribution based on the usual theories applied to protoplanetary disks. Then, based on the properties of the star, the software creates a specified number of photons and generates a temperature distribution in the disk, and propagates the photons. This method is called *radiative transfer*.

The output is two fits files: one containing an actual image of the model, at a specified wavelength, and the other the SED curve of the full stellar system.

8.2 Difficulties

To create such gas and dust distribution, one needs not only the radius of every disk, but also their flaring exponents, characteristic distances and heights, and compositions. With the properties of the star, it sums up to more than 50 free parameters, 30 of them not given by the literature nor the previous studies. That, and the fact that one MCFOST run takes about one minute, makes it impossible to consider an MCMC optimization. It has to be done by hand, at least partially.

8.3 Modeling results

To model J1615 with MCFOST I decided to fit simultaneously the SED curve and the image result, and more specifically the radial profile. To do so I designed a small bash script that makes the convolution with the beam and the light extinction automatic. I also designed a tool that probes the parameters space around a position, to find a probable direction in which the fit might be better.

Combining these tools and a lot of patience, I achieved an almost proper SED fitting, as shown in Figure 17. On that SED, the main difficulty is to reduce the 1µm bump without losing it completely, and without loosing the 3mm value, which is an excellent constraint on the mass of the outer disks.

The details of the model used can be found in the GitHub repository. The MCFOST parameters file used to compute this SED is given in Appendix A.6.

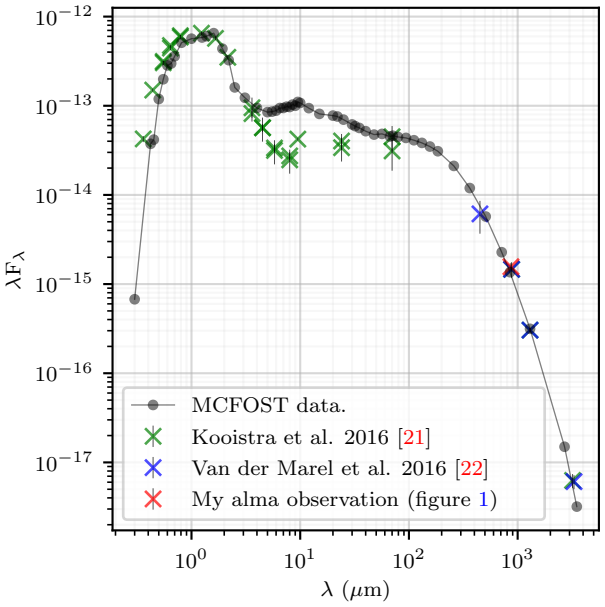

Figure 17: Experimental SED compared to the MCFOST model I fitted.

9 Performance Improvements details

9.1 The computational need

Galario [10] is a tool used to work with visibilities and fit them. The main specificity of this tool is that it can compute visibilities from an image or a radial profile, and can therefore be used to compute the difference between a model and interferometric data. To do so, it needs a lot of computing power, as the data set I used is 640MB large on disk and composed of millions of points. Each point needs to be computed every time the model is changed, and optimizing a fitting consists of many evaluations of the model.

Running Galario on my laptop's CPU was very slow (around 20 seconds for a full evaluation, using one core), and the memory usage was too high to consider the emcee default multiprocessing computation. On the Alma computer, it was not much better, the CPU being ancient compared to mine. The two solutions I came across where to use of a powerful and more versatile multiprocessing method, and to use CUDA graphics card computation. I also had an access to Leftraru, which happened to be more than helpful.

9.2 Python Voodoo magic

This first point I want to make here is that Python is a very rich programming language, that happens to be very easy to learn, but also very easy to learn wrong, as I experienced when I first tried to learn it, back when I was 16. Many tools have already been developed as standalone modules to import, and many shortcuts have been implemented, but some cost time, as I painfully experienced in this internship.

The main shortcut implemented in Python that happens to be computationally very expensive is the type changing routine. Types in Python are very flexible and one can replace almost any float by an int, yet converting full NumPy arrays can be very expensive. When combining that with the fact that some mathematically equivalent formulations are implemented differently, one can end up with a time difference, as shown in Figure 18.

```
>>> t=time()
>>> for i in range(int(1e7)):
...     x=np.random.random()
...     a=np.sqrt(x)
>>> print(time()-t)
9.321187019348145

>>> t=time()
>>> for i in range(int(1e7)):
...     x=np.random.random()
...     a=math.sqrt(x)
>>> print(time()-t)
4.087422132492065

>>> t=time()
>>> x=np.random.random(int(1e7)) ; a=np.sqrt(x)
>>> print(time()-t)
0.11801743507385254
```

Figure 18: Time comparison between different Python methods. Optimizing such things led to a major performance improvement (run-time divided by 2).

9.3 The use of a super computer

What is called a super computer is (generally) a cluster of many "normal" computers[5] connected and set up so they can work together as one huge computing machine, shared between users all across the world. It is a very expensive installation that requires permanent technical support, and has a power draw sufficient to heat up a whole Manhattan-like skyscraper. For that reasons, its use comes at a cost. The advantages of such computing methods make it worth paying for, especially in astrophysics. Indeed, the task submission system (SLURM in Leftraru) allows a lot more flexibility than the use of a local computer.

The main need I had was to be able to let a computation running overnight (thus making my laptop a no-go), and without any influence from other persons (Alma for example was a shared machine, and every user had the same CPU priority). The fact that Leftraru had many CPU cores (120 for my account), virtually unlimited RAM (2TB), and V100 GPUs also revealed to be a step up in my program development.

9.4 Multiprocessing

The multiprocessing distribution of the computation consists in sharing the walkers on multiple instances of Galario. It can be done very simply using the Python module Multiprocessing, but that module scales badly with the number of cores, so that after 20 cores, you gain nothing adding more.

The alternative to this module is to use MPI, a more complex and lower level implementation of multithreading. Doing that was a pain, as it required phone calls to the Leftraru staff, so they could install the required C++ libraries.

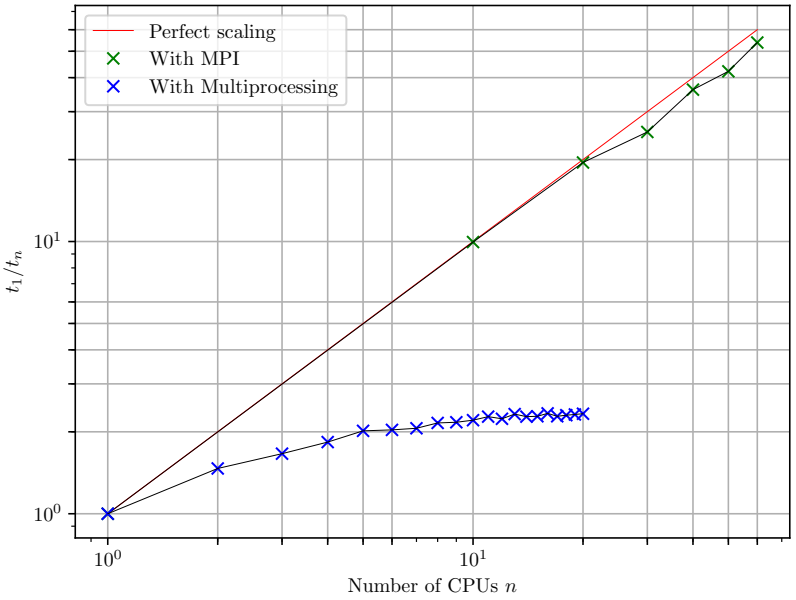

Figure 19: Scaling comparison of MPI and Multiprocessing.

It ended up not working on Leftraru due to a dependency problem that I could do nothing about. On my PC though, it worked fine and made the Galario process capable of running with 4 cores without memory error. For the image plane fitting, I could use Alma's 16 cores without any problem. The scaling of the run time is shown in Figure 19.

9.5 CUDA

CUDA is an NVIDIA parallel computing platform. It allows using a GPU in general purpose computing, taking advantage of the very high number of cores available in a GPU. To use a GPU with Python, you need to design a program specifically, and Galario happens to be optimized for the use of CUDA. The run-time was divided by about 5 using CUDA on Leftraru with my data set.

[5] If a computer with 44 cores in a 4000$ CPU, more than a terabyte of RAM and two state of the art GPUs can be considered as "normal".

Conclusions and prospects

The program development and performance improvements I made came with the creation of detailed methods and tutorial on how to implement them. That should allow anyone to apply those performance improvements, to make any fitting process using Galario or Emcee faster. The work I conduced during these three months will be used again and improved, and is already being applied to the fitting of an other disk by a PhD student of the research team. It should lead to a publication mainly focused on J1615, describing and analysing the structure of this disk.

the next steps of the project will be (1) to pursue the radiative transfer modeling (joint analysis of SPHERE and ALMA that trace different dust grain population), (2) to interpret the results with the help of hydrodynamic simulations of planet-disk interactions in order to constrain the properties of the planets likely embedded in the disk of J1615.

This internship was very instructive for me, as it allowed me to discover and to have firsthand experience in astrophysics and research in general, and was both a scientific and a human experience that I deeply enjoyed, and would be glad to repeat.

References

[1] J. de Boer, G. Salter, M. Benisty, A. Vigan, A. Boccaletti, P. Pinilla, C. Ginski, A. Juhasz, A. L. Maire, and S. Messina, "Multiple rings in the transition disk and companion candidates around RX J1615.3-3255. High contrast imaging with VLT/SPHERE," *Astronomy and Astrophysics*, vol. 595, p. A114, Nov 2016.

[2] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[3] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–.

[4] T. E. Oliphant, *A guide to NumPy*, vol. 1. Trelgol Publishing USA, 2006.

[5] D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman, "emcee: The MCMC Hammer," *Publications of the ASP*, vol. 125, p. 306, Mar. 2013.

[6] M. M. McKerns, L. Strand, T. Sullivan, A. Fang, and M. A. G. Aivazis, "Building a framework for predictive science," *CoRR*, vol. abs/1202.1056, 2012.

[7] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using python," *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011.

[8] L. Dalcin, R. Paz, M. Storti, and J. D'Elia, "Mpi for python: Performance improvements and mpi-2 extensions," *Journal of Parallel and Distributed Computing*, vol. 68, no. 5, pp. 655–662, 2008.

[9] L. Dalcin, R. Paz, and M. Storti, "Mpi for python," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 1108–1115, 2005.

[10] M. Tazzari, F. Beaujean, and L. Testi, "GALARIO: a GPU Accelerated Library for Analysing Radio Interferometer Observations," *Mon. Not. Roy. Astron. Soc.*, vol. 476, no. 4, pp. 4527–4542, 2018.

[11] J. P. McMullin, B. Waters, D. Schiebel, W. Young, and K. Golap, "CASA Architecture and Applications," vol. 376, p. 127, Oct 2007.

[12] ALMA Partnership, C. L. Brogan, L. M. Pérez, T. R. Hunter, W. R. F. Dent, A. S. Hales, R. E. Hills, S. Corder, E. B. Fomalont, and C. Vlahakis, "The 2014 ALMA Long Baseline Campaign: First Results from High Angular Resolution Observations toward the HL Tau Region," *Astrophysical Journal, Letters*, vol. 808, p. L3, Jul 2015.

[13] J. K. Jørgensen, C. Favre, S. E. Bisschop, T. L. Bourke, E. F. van Dishoeck, and M. Schmalzl, "Detection of the simplest sugar, glycolaldehyde in a solar–type protostar with ALMA," *The Astrophysical Journal*, vol. 757, p. L4, aug 2012.

[14] The Astropy collaboration, "The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package," *Astronomical Journal*, vol. 156, p. 123, Sept. 2018.

[15] C. Pinte, F. Ménard, G. Duchêne, and P. Bastien, "Monte Carlo radiative transfer in protoplanetary disks," *Astronomy and Astrophysics*, vol. 459, pp. 797–804, Dec 2006.

[16] A. Glindemann, *Optical interferometry*. Cambridge University Press, 2012.

[17] J. P. Williams and L. A. Cieza, "Protoplanetary disks and their evolution," *Annual Review of Astronomy and Astrophysics*, vol. 49, no. 1, pp. 67–117, 2011.

[18] M. V. Persson, "Current view of protostellar evolution (eng)," Aug 2014.

[19] M. Ansdell, J. P. Williams, N. van der Marel, J. M. Carpenter, G. Guidi, M. Hogerheijde, G. S. Mathews, C. F. Manara, A. Miotello, and A. Natta, "ALMA Survey of Lupus Protoplanetary Disks. I. Dust and Gas Masses," *Astrophysical Journal*, vol. 828, p. 46, Sep 2016.

[20] T. Henning and D. Semenov, "Chemistry in protoplanetary disks," *Chemical Reviews*, vol. 113, no. 12, pp. 9016–9042, 2013.

[21] R. Kooistra, I. Kamp, M. Fukagawa, F. Ménard, M. Momose, T. Tsukagoshi, T. Kudo, N. Kusakabe, J. Hashimoto, and L. Abe, "Radial decoupling of small and large dust grains in the transitional disk RX J1615.3-3255," *Astronomy and Astrophysics*, vol. 597, p. A132, Jan 2017.

[22] N. van der Marel, B. W. Verhaar, S. van Terwisga, B. Merín, G. Herczeg, N. F. W. Ligterink, and E. F. van Dishoeck, "The (w)hole survey: An unbiased sample study of transition disk candidates based on Spitzer catalogs," *Astronomy and Astrophysics*, vol. 592, p. A126, Aug 2016.

[23] J. E. Pringle, "Accretion discs in astrophysics," *Annual Review of Astronomy and Astrophysics*, vol. 19, no. 1, pp. 137–160, 1981.

[24] S. Facchini, E. F. van Dishoeck, C. F. Manara, M. Tazzari, L. Maud, P. Cazzoletti, G. Rosotti, N. van der Marel, P. Pinilla, and C. J. Clarke, "High gas-to-dust size ratio indicating efficient radial drift in the mm-faint CX Tauri disk," *Astronomy and Astrophysics*, vol. 626, p. L2, Jun 2019.

This work has made use of data from the European Space Agency (ESA) mission *Gaia* (https://www.cosmos.esa.int/gaia), processed by the *Gaia* Data Processing and Analysis Consortium (DPAC, https://www.cosmos.esa.int/web/gaia/dpac/consortium). Funding for the DPAC has been provided by national institutions, in particular the institutions participating in the *Gaia* Multilateral Agreement.

A Appendix

A.1 SED point calculation

To compute a SED point from an image I used the following algorithm, and the associated functions. The first steps is to open the data. Then I need to determine which pixels are relevant, for that I compute the standard deviation of the background noise and only keep pixels that are at least three times bigger than that value. Then I sum all this pixels and divide them by the size of the observation beam, as the image is in mJy per beam. I finally convert this value to SI, multiplying by the wavelength and converting the units.

Python pseudo-code:

```
def ComputeLimitSignal(image):
    std = image background noise standard deviation
    return(3*std)

def ComputeSEDImage(location):\textbf{Python pseudo-code :}

    header, image = openimage(location)

    lim       = ComputeLimitSignal(image)
    Flux      = sum( pixels in image if pixel > lim)

    beam_area = beam_area from header
    lambda    = wavelength from header
    sed       = Flux * lambda / beam_area
    err       = lim/3  * lambda / beam_area

    sed, err  = sed * conversion, err * conversion * number of pixels
    return(sed,err)
```

Real implementation:

```
def ComputeLimitSignal(image):
    """Computes the background noise of an image,
    and return 3*standard deviation of s\textbf{Python pseudo-code :}
uch noise"""
    l1,l2=image.shape
    i=image[:l1//10,:l2//10]
    return(3*np.std(i.flatten()))

def ComputeSEDImage(location):
    """Computes the SED point of an XP image"""
    header,image=openimage(location)
    # Do not sum the background noise
    lim=ComputeLimitSignal(image)
    FluxTot=np.sum(((image>lim)*image).flatten())
    # Compute the beam area to get the real SED
    beam_area = np.pi/(4*np.log(2)) *
        header['BMAJ']/header['CDELT2']*header['BMIN']/header['CDELT2']
    Flux=FluxTot/beam_area
    sed=Flux*header['RESTFRQ']*1e-26
    err=lim/beam_area*header['RESTFRQ']*1e-26*np.sum(image>lim)/3
    return(sed,err)
```

For this implementation I supposed that the 10% first pixels in the top left corner are representative of the background noise of the image, which will not always be the case. I also assumed the image to be in milli Jansky per beam, which is one of the standard output of the CASA software.

A.2 TiltFinder implementation details

This is a short explanation of the TiltFinder program, used to determine the inc and PA of a protoplanetary disk. The main requirement for the program to work properly is that the disk must be relatively Gaussian. To have a better fitting, I decided to blur the image with a large σ.

The basic principle is to define a cost function between a generated image and the real image, and to minimize it. The method I chose to minimize it is a simple gradient descent from SciPy [3].

Python pseudo-code:

```
##### Opening the image properly
header,image=openimage(location)
##### Resizing it around the interesting part
image=resizedimage(image)
##### Blurring image and evaluating the cost
blur = blurimage(image)
def Function_to_Minimize(params):
    model = gaussianfit(params)
    return(cost(blur,model))
##### Optimize
Optimized_params = minimize(Functions_to_Minimize)
return(params)
```

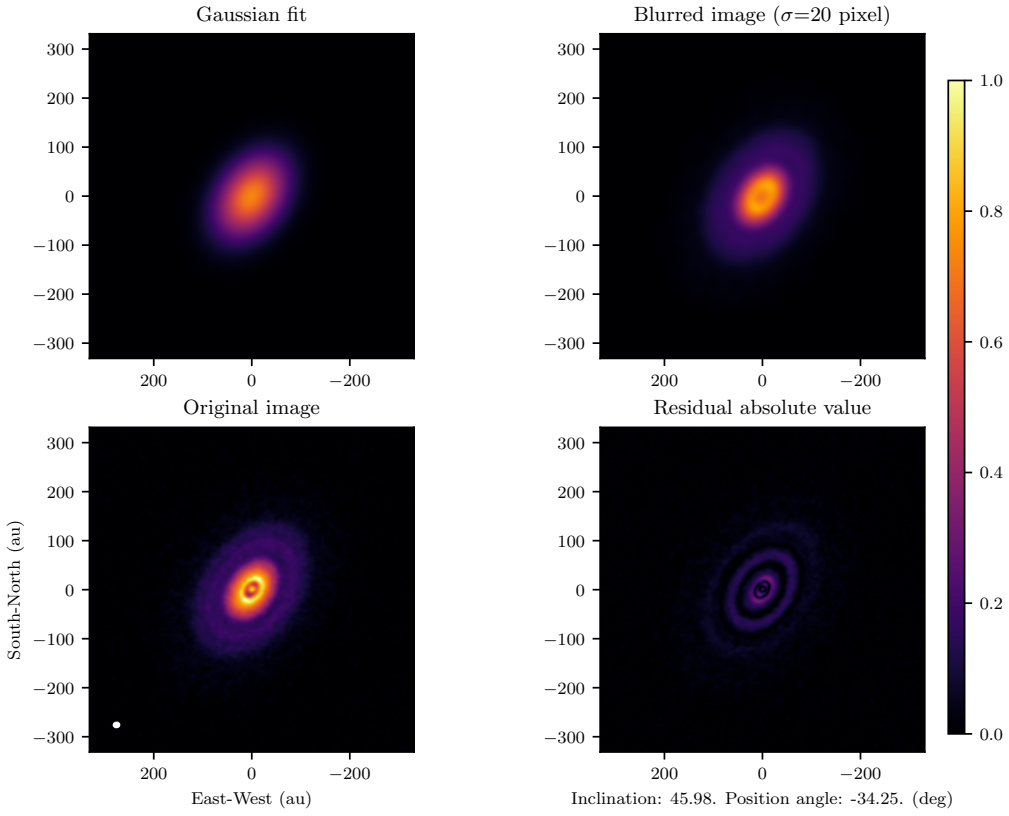

Figure 20: TiltFinder image output, showing a comparison between the original image and the optimized tilted Gaussian.

A.3 Implementing a model with Astropy

It is possible to create function generators using Astropy. By function generator, I mean a tool taking parameters as input and a function as output, based on those parameters. For example, I could use the `GaussianGenerator` tool to generate a Gaussian function, using a syntax such as `Gaussian=GaussianGenerator(amp,sigma)`. Then using `Gaussian(x)` would give the value of a Gaussian of amplitude `amp` and width `sigma`.

Here is a minimalist example, creating a parabolic function with three parameters.

```
>>> from astropy.modeling import models, fitting

>>> x=np.arange(1.,2.,0.0001)

>>> @models.custom_model
... def Model1(x, a=1., b=1., c=1.):
...     return(a*x**2.+b*x+c)
...

>>> function1=Model1(a=4.,b=-14.,c=12.)

>>> function1
<Model1(a=4., b=-14., c=12.)>

>>> function1(x)
array([ 2.00000000e+00,  1.99940004e+00,  1.99880016e+00, ...,
       -5.99640000e-04, -3.99840000e-04, -1.99960000e-04])
```

The actual implementation of my fitting functions are as follow. First a Gaussian ring, then a ring based on Facchini et al. [24], and then a central Gaussian.

```
@models.custom_model
def GaussianRing(xx, yy, amplitude=1., xc=0., yc=0.,
        width=1., a=1., b=1., theta=0.):
    """
    This makes a Gaussian ring centered on (xc,yc), elliptic with semi-axis a and
    b,
    and rotation theta.
    """
    c=math.cos(theta)
    s=math.sin(theta)
    # centering the mesh
    x=xx-xc
    y=yy-yc
    # compute distance to the ellipse
    u=np.sqrt(((x*c+y*s)/a)**2.+((y*c-x*s)/b)**2.) - 1.
    # compute Gaussian
    return( amplitude * np.exp(  ( -.5*(a*b)*(width)**-2. ) * (u**2.) ) )
```

```python
@models.custom_model
def FFRing(xx, yy, i0=1., i1=1., sig0=1., sig1=1.,
        gam=1., xc=0., yc=0., a=1., b=1., theta=0.):
    """
    Facchini Ring, refer to eq 1 in arXiv 1905.09204.
    """
    c=math.cos(theta)
    s=math.sin(theta)
    x=xx-xc
    y=yy-yc
    u=np.sqrt(((x*c+y*s)/a)**2.+((y*c-x*s)/b)**2.)
    f = (i0 * ((u / sig0)**gam)) * np.exp(-(u**2.) / (2. * sig0**2.))
    f += i1 * np.exp(-(u**2.) / (2. * sig1**2.))
    return(f)
```

```python
@models.custom_model
def Gaussian2D(xx, yy, amplitude=1., x_mean=0., y_mean=0.,
        x_stddev=1.,y_stddev=1.,theta=0.):
    """
    Redefining the Gaussian2D Model of astropy module,
    for it to be faster in terms of computation.
    Please refer to the documentation of astropy.modeling.models.Gaussian2D
    """
    c=math.cos(theta)
    s=math.sin(theta)
    x=xx-x_mean
    y=yy-y_mean
    c2=math.cos(2.*theta)
    s2=math.sin(2.*theta)
    csq=c**2.
    ssq=s**2.
    a=.5*( csq * (x_stddev )**-2. + ssq * ( y_stddev )**-2.)
    b=.5*s2*(y_stddev**-2.-x_stddev**-2.)
    c=.5*(ssq*(x_stddev**-2.) + csq*(y_stddev**-2.))
    return(amplitude*np.exp( - (a*x**2.+b*(x*y)+c*y**2.) ))
```

```python
ring1=FFRing()
ring2=GaussianRing()
ring3=GaussianRing()
ring4=GaussianRing()
centralgauss=Gaussian2D()

Model_J1615=centralgauss+ring1+ring2+ring3+ring4
```

A.4 Emcee

Emcee [5] is an implementation of the MCMC minimization algorithm. It is based on the idea of partially randomly probe a multi-dimensional space in order to find the minimum of a function. To do so, it creates an ensemble of walkers, *i.e.* an ensemble of moving points in the space, and moves them following specific rules.

To implement it, we need to define a cost function to minimize, an initial set of positions, and boundaries in the space. The real implementation needs to take other details into considerations, such as the number of cores to use in the process, or the type of multi-threading used. To have a look into my implementation, see the `ModelingEmcee.py` file in my GitHub repository.

Python pseudo–code:

```
p0=initial position
p0_list=[randomize n positions around p0]
def cost(p):
  model=Model_J1615(p)
  return(||model-image||)
def boundaries(p):
  if p is into the boundaries:
    return(0)
  else:
    return(+infinity)
def FullCost(p):
  return(boundaries(p)+cost(p))
emcee_optimize(FullCost,p0_list)
```

The walkers can define a discreet probability distribution. The zones where most walkers are stuck in corresponds to the highest probability, and therefore the lowest value of the cost function. To evaluate the success of an Emcee optimization, the corner plot tool is useful. It represents the correlation between each pair of parameters, plotting the joint probability distribution. It has to look as sharp as possible, and gives us important information about local minimum and redundancy in parameters.

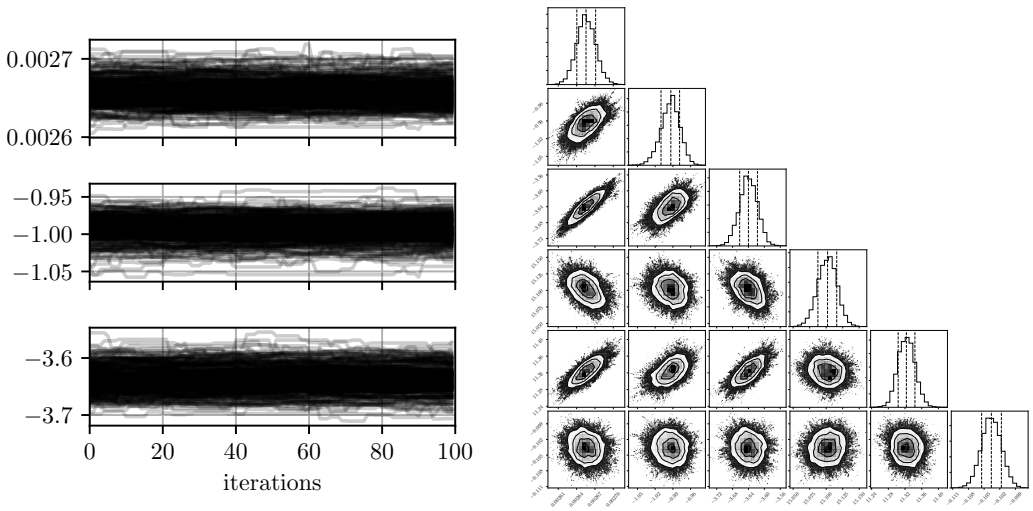

Figure 21: A typical converged Emcee partial output. The parameters are restrained in a very small and stable area, and the corner plot is sharp. These values are extracted from the 3 ad 6 first parameters of my image plane fitting process.

A.5 Emcee numerical results

In this section I will present two table per optimization. One will only be a simpler per ring properties description, while the other will be a full parameters description.

A.5.1 Image plane fitting

The rings I ended up using for this fitting are a central Gaussian G0, a Gaussian ring multiplied by a power law FR1, and three Gaussian rings GR2–4.

Layer	Radius (au)	Amplitude (mJy/beam)	Gaussian width (σ, au)
Central Gaussian G0	–	2.610e-3	6.2291
Power law ring F1	15.814	6.141e-4	–
Gaussian ring GR2	23.340	2.959e-3	9.7004
Gaussian ring GR3	106.25	3.073e-4	31.360
Gaussian ring GR4 (faulty)	186.23	3.925e-5	42.861

Table 1: Summary of the emcee results for the image fitting process using Emcee. σ corresponds to the Gaussian standard deviation. GR4 is not representative as it fits background noise.

A.5.2 (u, v) plane fitting

For this fit, I decided to use 4 Gaussian rings, and to reduce their freedom, especially fixing their center, inclination and position angle.

Layer	Radius (au)	Amplitude (arbitrary unit, log scale)	Gaussian width (au)
Central Gaussian G0	–	11.026	0.7465
Gaussian ring GR1	1.5826	9.9739	68.715
Gaussian ring GR2	25.408	10.278	14.619
Gaussian ring GR3	97.045	8.5007	3.6525
Gaussian ring GR4	116.11	8.4624	7.0975

Table 2: Summary of the Emcee results for the $(u,\ v)$ plane fitting.

The most notable difference is that the outer rings are much sharper in this fitting. That might be due to the fact that we made no beam convolution in the image plane fitting.

Parameter name	15% percentile	Median	85% percentile
Central Gaussian G0			
amplitude_0	2.610e-3	2.615e-3	2.618e-3
xc_0	-1.04728	-1.03924	-1.03570
yc_0	-3.70907	-3.69838	-3.69434
xstd_0	15.05814	15.06351	15.06645
ystd_0	11.26137	11.27178	11.27677
theta_0	-0.10922	-0.10854	-0.10818
Power law ring FR1			
i0_1	5.716e-4	5.735e-4	5.7465e-4
i1_1	7.830e-4	7.882e-4	7.913e-4
sig0_1	1.28183	1.28842	1.28980
sig1_1	0.44824	0.44950	0.45021
gam_1	4.67338	4.67820	4.68007
xc_1	0.33112	0.33457	0.33613
yc_1	0.24303	0.24574	0.24725
a_1	33.07245	33.14110	33.16868
b_1	33.65266	33.72201	33.74980
theta_1	-0.04395	-0.04318	-0.04261
Gaussian ring GR2			
amplitude_2	2.969e-3	2.970e-3	2.971e-3
xc_2	-0.80081	-0.79834	-0.79701
yc_2	-0.46515	-0.46306	-0.46196
width_2	20.49650	20.50571	20.50915
a_2	48.35268	48.35669	48.35961
b_2	50.31715	50.32187	50.32394
theta_2	0.06083	0.06152	0.06179
Gaussian ring GR3			
amplitude_3	6.4956e-4	6.4964e-4	6.4968e-4
xc_3	0.59396	0.60192	0.60592
yc_3	-1.49655	-1.48652	-1.48063
width_3	66.28178	66.29217	66.29768
a_3	221.03183	221.04322	221.05016
b_3	228.43998	228.45219	228.45790
theta_3	0.01641	0.01722	0.01767
Gaussian ring GR4			
amplitude_4	8.293e-5	8.298e-5	8.300e-5
xc_4	-4.86612	-4.81012	-4.78337
yc_4	13.19933	13.26826	13.30691
width_4	90.51086	90.60467	90.64812
a_4	387.42063	387.54946	387.60922
b_4	399.81574	399.84269	399.85641
theta_4	-0.41073	-0.40366	-0.40063

Table 3: Value of every parameter of the J1615 model in image plane fitting. The "_n" refers to the number of the ring

Parameter name	15% percentile	Median	85% percentile
Central Gaussian G0			
f0_0	10.95160	11.02614	11.06120
sigma_0	0.00472	0.00473	0.00474
Gaussian ring GR1			
amplitude_1	9.95791	9.97398	9.98054
width_1	0.42908	0.43577	0.43838
center_1	0.01001	0.01004	0.01007
Gaussian ring GR2			
amplitude_2	10.27771	10.27804	10.27818
width_2	0.09231	0.09271	0.09280
center_2	0.16022	0.16113	0.16129
Gaussian ring GR3			
amplitude_3	8.47161	8.50068	8.51255
width_3	0.02152	0.02316	0.02520
center_3	0.61354	0.61543	0.61736
Gaussian ring GR4			
amplitude_4	8.43635	8.46239	8.47069
width_4	0.04500	0.04501	0.04502
center_4	0.73343	0.73634	0.73876

Table 4: $(u,\ v)$ plane fitting detailed results. The amplitude has a logarithmic scale. The width and center correspond to the standard deviation of the Gaussian and the radius of each ring in arcsec.

A.6 MCFOST results

The bare file used to generate the SED presented in figure 17 is available on GitHub, this is just a basic description of the rings I found.

Layer	Inner diameter	Outer diameter
Central ring CR1	0.412	5.178
Ring R2	17.84	61.88
Ring R3	65.54	94.49
Ring R4	127.2	138.7

Table 5: MCFOST results. These results appear to be coherent with the $(u,\ v)$ plane fitting.

A.7 Other projects

During this internship, I had many minor objectives, tools to develop, codes to analyse, clean and optimize, *etc.* This section briefly describes two of these side projects.

A.7.1 Point and click localisation

When fitting a spiral disk, one of the key information to get is the spirals opening rate. To find it, multiple automatic methods exist, but I wanted a tool that could help me getting that information just by clicking on the image. I developed a tool that allows the user to open a fits file and to point and click on it, recording the coordinates of the pointing. It can be used to determine distances, or spirals angles.

A.7.2 FitsSlider

The main tool used to plot fits images is CASA, but it's a Mac OS X and Red Hat only program. Using Debian, I needed a tool to plot these images in a fast and useful method, with sliders adjusting for the maximum and minimum of the image.

The program I developed is named FitsSlider. It's a very simple Python tool, designed to be a fast observation tool, as shown in Figure 22.

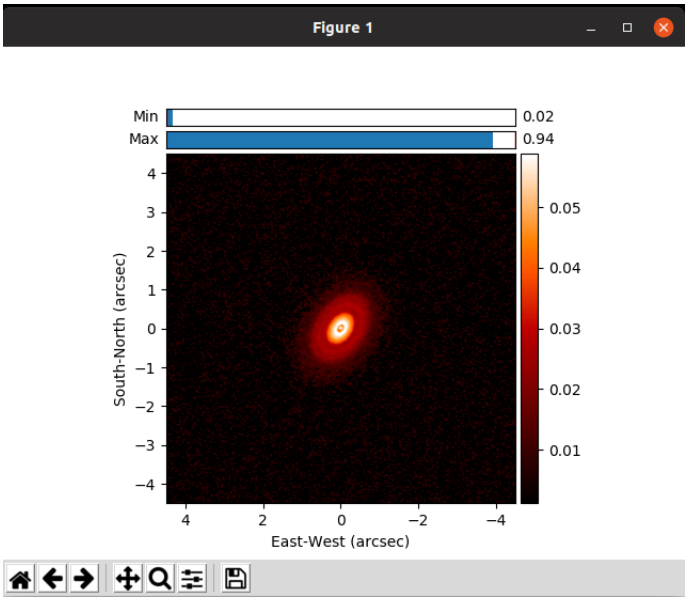

Figure 22: FitsSlider display, called by `FitsSlider J1615.fits --function 'lambda x : x**.5'`.

A.8 GitHub repositories

The code I produced during this internship is available in the following GitHub repositories:

Fitting the image plane: DiskFitting

Fitting the visibilities plane: GalarioFitting

Using MCFOST: mcfost_scripts

Plotting fits images: FitsSlider