UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Bayesian Image Reconstruction Based on Voronoi Diagrams

TESIS PARA OPTAR AL GRADO DE MAGISTER EN CIENCIAS MENCIÓN COMPUTACIÓN

Guillermo Felipe Cabrera Vives

Profesor Guía:
Dr. Simón Casassus Montero

Miembros de la Comisión:
Dra. Nancy Hitschfeld Kahler
Dr. Javier Ruiz del Solar
Dr. Domingo Mery Quiroz

Santiago de Chile
Enero 2008

Resumen

El objetivo de esta tesis es crear un algoritmo de reconstrucción de imágenes usando diagramas de Voronoi. Presentaremos un algoritmo de reconstrucción Bayesiano basado en diagramas de Voronoi (VIR) para datos interferométricos.

Un interferómetro es un telescopio que consiste en un conjunto de antenas. Estas antenas obtienen *visibilidades*, que son la transformada de Fourier de la imagen original. Las visibilidades se obtienen en solamente algunos puntos del plano de Fourier, por lo que para obtener la imagen asociada a ellas se necesita de algoritmos de reconstrucción.

La aplicación de análisis Bayesiano al problema inverso nos permite obtener la probabilidad a posteriori de una nueva parametrización de imágenes interferométricas. Usaremos un diagrama de Voronoi variable como nuestro modelo en lugar de la usual grilla fija de pixeles. Cuantizando las intensidades, podemos calcular la función de verosimilitud y la probabilidad a priori. Optimizamos el diagrama de Voronoi, incluyendo el número de polígonos como parametros libres. Para esto, usamos distintos algoritmos de optimización, incluyendo el método del gradiente conjugado, algoritmos genéticos y métodos de Monte Carlo.

Aplicamos nuestros algoritmos para deconvolucionar simulaciones de datos interferométricos; esto es, obtener la imagen que se ajuste mejor a estos datos. Para comparar con reconstrucciones hechas usando imágenes de pixeles fijos, así como también los distintos algoritmos de optimización, utilizamos residuos, imágenes restauradas, y valores de χ^2. VIR tiene la ventaja de modelar la imagen con pocos parámetros libres, obteniendo una mejor imagen desde un punto de vista Bayesiano.

Abstract

The objective of this thesis is to create an image reconstruction technique using Voronoi diagrams to represent our image. We present a Bayesian Voronoi image reconstruction technique (VIR) for interferometric data.

An interferometer is a telescope consisting in a set of antennas. These antennas obtain *visibilities*, which are the Fourier transform of the original image. Visibilities are obtained only in some sparse points of the Fourier plane. To obtain the image associated to these visibilities it is necessary to use reconstruction algorithms.

Bayesian analysis applied to the inverse problem allows us to derive the a posteriori probability of a novel parameterization of interferometric images. We use a variable Voronoi diagram as our model in place of the usual fixed pixel grid. A quantization of the intensity field allows us to calculate the likelihood function and a priori probabilities. The Voronoi diagram is optimized including the number of polygons as free parameters. We use different optimization algorithms including the conjugate gradient method, genetic algorithms and Monte Carlo methods.

We apply our algorithms to deconvolve simulated interferometric data; this means, to obtain the image that best fits these data. Residuals, restored images and χ^2 values are used to compare our reconstructions with fixed grid models and to evaluate our different optimization algorithms. VIR has the advantage of modeling the image with few parameters, obtaining a better image from a Bayesian point of view.

Acknowledgments

I would like to thank all my family, including my mother, Ana María, her husband, Alejo, my father, Guillermo, my sisters, Anita, Sole, and Vale, and my brother Javier for all the support they have given me through all my life.

I would also like to thank my girlfriend Mariajose for being with me at all times (good and bad times) helping and supporting me in all I needed, always with a smile in her heart.

I would also like to thank my teacher and friend, Simon, for letting me work with him through all this years, teaching me all I asked always with a great disposition. I also thank Nancy for being such a great devoted teacher, always having time to hear her students.

We are grateful to Tim Pearson for advice on FFTs and the use of MOCKCBI. G.F.C. and S.C. acknowledge support from FONDECYT grant 1060827, and from the Chilean Center for Astrophysics FONDAP 15010003. We are also grateful to Giorgio Krstulovic for advice on Appendix A.

Contents

Chapter 1

Introduction

1.1 Introduction to Image Reconstruction for Interferometric Data

The angular resolution of a telescope, δ, is the smallest angle that two objects can be separated in order to distinguish them as different objects. Using diffraction theory, it can be shown that for a single circular telescope of diameter D, and radiation of wavelength λ, this value is $\delta \propto \frac{\lambda}{D}$. This means that for large wavelengths (small frequencies), a large telescope is needed to obtain good resolution. Practical limits exist on the telescope size.

Radioastronomy studies low frequency celestial sources. For this purpose, arrays of telescopes (or antennas), called *interferometers*, are used. When placed at a distance D these antennas get the same resolution of a single dish telescope of radius D.

Interferometers do not obtain the sky image directly. Instead, they acquire *visibilities*, which are the Fourier transform of the image. Each pair of antennas forms a vector $\vec{k} = (u, v)$ in the Fourier plane. The visibility $V(\vec{k})$ for a pair of antennas with a baseline $\vec{k}$ is

$$V(\vec{k}) = \int_{-\infty}^{+\infty} A(\vec{x}) I(\vec{x}) e^{2\pi i \vec{k}\vec{x}} \frac{dx\,dy}{\sqrt{1 - x^2 - y^2}} \;, \tag{1.1}$$

where $\vec{x} = (x, y)$ are the coordinates of the object to study, $A(\vec{x})$ is the interferometer primary beam, $I(\vec{x})$ is the object intensity at $\vec{x}$, and $\vec{k}$ is the vector formed by each pair of antennas, as in Figure 1.1.

Due to the fact that there are only a finite number of pairs of antennas, when doing an inverse transform side-lobes surrounding the primary beam appear. These side-lobes are irregularities in the radiation pattern due to the finite extent of the points in

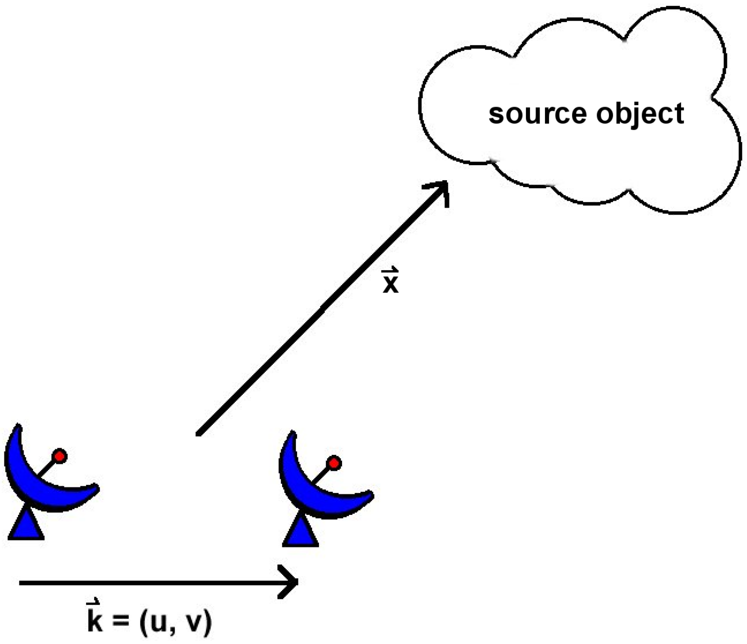

Figure 1.1: Two antennas and their respective baseline.

the (u, v) plane and to gaps in the coverage (Briggs et al., 1999). The inverse transform applied directly on the visibilities is called the *dirty map* (DM).

Astronomical interferometric data result from the addition of instrumental noise to the convolution of the sky image and the instrumental response. Because of incomplete sampling in the (u, v) plane, obtaining sky images from interferometric data is an instance of the inverse problem, and involves reconstruction algorithms.

1.2 Existing Methods

Different reconstruction algorithms have been created in order to obtain the image that best fits the interferometric data. We will briefly describe the most popular ones used in the astronomy community.

The CLEAN method consists of modeling the side-lobe disturbances and subtract them iteratively from the dirty map (Högbom, 1974). The CLEAN method works well for low noise and simple sources. But if the source has many complex features, or if the data is too noisy, CLEAN will do only a few iterations returning a noisy image (Högbom, 1974). Another shortcoming is that CLEAN involves some ad-hoc parameters (the loop gain, stopping criteria, clean beam) that bias the final reconstruction, in the sense that CLEAN can give many different reconstructions for the same dataset.

The maximum entropy method (MEM) finds the image that simultaneously best fits the data, within the noise level, and maximizes the entropy S. This is done by minimizing

$$L_{\mathrm{MEM}} = \chi^2 - \lambda S, \tag{1.2}$$

where, for the case of interferometric data, χ^2 can be calculated as

$$\chi^2 = \sum_{k=1}^{N_{\mathrm{Vis}}} \frac{||V_k^{\mathrm{obs}} - V_k^{\mathrm{mod}}||^2}{\sigma_k^2}, \tag{1.3}$$

where the sum runs over all the N_{Vis} visibilities, the symbol $||z||$ stands for the modulus of the complex number z and σ_k is the root mean square (rms) noise of the corresponding visibility. λ is a control parameter and the entropy S varies for different implementations (e.g. Narayan & Nityananda, 1986). The entropy is used as a regularizing term in a degenerate inverse problem, when there are more free parameters than data. Different formulations for S appear in the literature. Some examples are $\sum_i \ln(I_i)$, $\sum_i I_i \ln(I_i)$, $\sum_i \ln(p_i)$, $\sum_i p_i \ln(p_i)$, where I_i is the specific intensity value at pixel i and $p_i = I_i / \sum_i I_i$ (see Piña & Puetter, 1993, and references therein).

Cornwell & Evans (1985) used MEM in the AIPS VM task. Their method makes some approximations that diagonalize the Hessian matrix required to optimize their merit function. They used an entropy of the form $S = -\sum_i I_i \log(I_i/m_i)$, where the sum extends over all the pixels i, $\{I_i\}_{i=1}^n$ is the model image and $\{m_i\}_{i=1}^n$ is a prior image. However, the neglect of the side-lobe contribution to the Hessian may lead the optimization to local minima that still bear instrumental artifacts. Casassus et al. (2006) implemented a MEM algorithm based on the conjugate gradient method, without the use of the Cornwell and Evans approximation. They used an entropy of the form $S = -\sum_i I_i \log(I_i/M)$, where $\{I_i\}_{i=1}^N$ is the model image and M is a small intensity value, i.e they start with a blank image prior, and M is an intensity value much smaller than the noise.

Bayesian analysis is a powerful tool for image reconstruction techniques. In this application, our goal is to find the most probable image by maximizing its a posteriori probability. For Bayesian methods, the a priori and likelihood distributions are needed. To derive the a priori probability the definition of an intensity quantum is needed. This quantum represents the minimum measurable intensity unit. The intensity in each pixel can be interpreted as a number of quanta $I_i = \sigma_{\mathrm{q}} N_i$, where I_i is the intensity in pixel i, σ_{q} is the quantum size and N_i the number of quanta in pixel i.

Piña & Puetter (1993) used Bayesian analysis in the Pixon algorithm. They use a variable model and maximize $P(I, M|D)$, that is, the probability of the image I and model M given the data D. In their approach the model used to parameterize the image is a set of Gaussians which are used to average a pseudo-image. The pseudo-image starts as a maximum residual likelihood reconstruction and a local Gaussian pixon is assigned to each of its pixels. The number of pixons, and hence the number of free parameters, is reduced in each iteration.

Sutton & Wandelt (2006) have used Bayesian analysis for interferometric data, but using a fixed pixel grid to parameterize the model image. They use Gibbs sampling to determine the posterior density distribution.

The most typical model used in astronomy to represent the sky brightness distribution consists of a pixel grid. A big disadvantage of this grid is that the number of pixels remains fixed as well as their size. Often, uniform pixel grids involve more free parameters than really needed to fit the data.

1.3 Objectives

1.3.1 General Objectives

Explore Bayesian reconstruction with image models based on Voronoi tessellations in place of the usual pixelated image and develop a new tool that returns the image that best fits the interferometric data. This new tool/method is called "Voronoi image reconstruction" (VIR, hereafter).

The advantage of using Voronoi models is that it is possible to use a smaller number of free parameters, as required by Bayesian theory. Our purpose is not optimal CPU efficiency; we search for the optimal image and model from a Bayesian point of view.

1.3.2 Specific Objectives

- Define our merit function to be optimized. This will be done by calculating the a posteriori probability of our parameterization.
- Calculate the derivatives of our merit function in order to use them in the optimization procedure.
- Investigate, use, and compare different optimization algorithms.
- Develop a new tool that allows us to obtain the optimal model image from a Bayesian point of view.

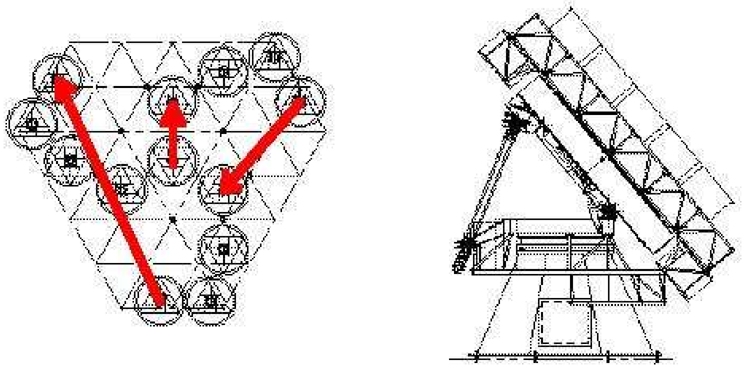

Figure 1.2: Cosmic Background Imager. Three baselines have been drawn as arrows.

1.3.3 Methodology

We used a variable Voronoi tessellation as our model in place of the usual fixed pixel grid. A quantization of the intensity field allowed us to define the a priori probability. Using this a priori probability and the likelihood, we can calculate the a posteriori probability to be maximized to a multiplicative constant.

The calculation of the gradient of our merit function allowed us to use optimization algorithms that require it. We studied conjugate gradient, genetic algorithms, and Monte Carlo methods and developed a new tool that allows us to use and combine them. We made qualitative and quantitative comparisons of reconstructions made with different algorithms.

We used the Cosmic Background Imager (CBI, Padin et al., 2002) to illustrate our method. The CBI is a planar interferometer array with 13 antennas, each 0.9 m in diameter, mounted on a 6 m tracking platform. The CBI in an example configuration of its antennas can be seen in Figure 1.2, where three baselines have been drawn. An example of CBI baselines is shown in Figure 1.3. The radius of the hole at the center of the (u, v) plane is the reciprocal of the minimum distance between two antennas, measured in wavelengths. The side-lobes of the CBI are caused mainly by this central hole in the (u, v) baselines.

1.4 Contents

We briefly summarize the elements of Bayesian theory that determine the probability distributions concerning our problem (Chapter 2). The new model based on Voronoi tessellations is described (Section 2.3), as well as optimization issues involved in our

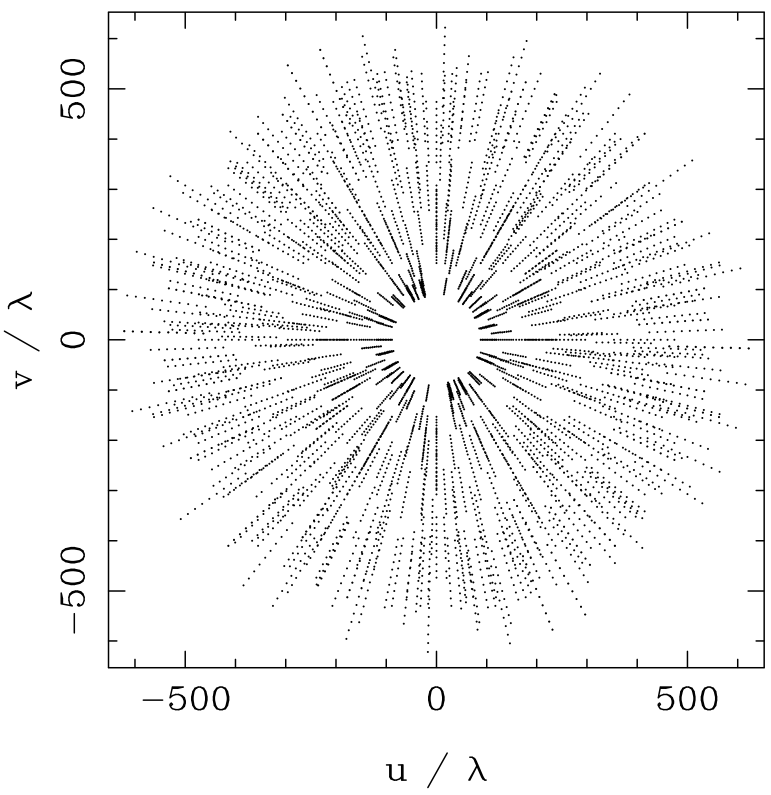

Figure 1.3: Coverage in the (u, v) plane of the CBI in the configuration used for our simulations.

problem (Chapter 3). We show use cases, module and class diagrams and discuss implementation details such as the optimal quantum size and number of Voronoi polygons (Chapter 4), compare reconstructions made with each reconstructor (Chapter 5) and finally summarize our results (Chapter 6).

Chapter 2

Bayesian Theory and Model

An image model is required to parameterize the sky brightness distribution. The most typical model used in astronomy is a rectangular grid of uniform pixels. That configuration of pixels is the model M, and the distribution of brightness in the model is called an image I. We search for the image that represents as accurately as possible the visibility data D. The Bayesian image reconstruction approach, using a fixed model, tries to find the image that maximizes the probability $P(I|D, M)$, i.e. find the most probable image given the data and the model.

Using the Bayes theorem, we obtain

$$P(I|D, M) = \frac{P(D|I, M)P(I|M)}{P(D|M)}. \tag{2.1}$$

Since the data is fixed, $P(D|M)$ is a constant in the problem when the model is not considered as a variable. Thus, the fixed image model optimization problem reduces to

$$\max_I P(I|D, M) = \max_I P(D|I, M)P(I|M). \tag{2.2}$$

The first term, $P(D|I, M)$ is called the likelihood, and measures how well our data represents our image. The second term, $P(I|M)$ is called the image prior, and gives the a priori probability of the image given the model, i.e. how probable is the image given only the model.

In the case of having a variable model, what we would like to find is the image and model that maximize $P(I, M|D)$, i.e. find the most probable image and model given the data. In this case we find

$$\begin{aligned} P(I, M|D) &= P(I|D, M)P(M|D) \\ &= \frac{P(D|I, M)P(I|M)P(M|D)}{P(D|M)} \end{aligned}$$

$$= \frac{P(D|I,M)P(I|M)P(M)}{P(D)}. \tag{2.3}$$

Since the data is fixed, $P(D)$ is constant in our problem. As we cannot privilege one model over another in the absence of image and data, $P(M)$ is the same for all models, so it is not important for our analysis. This way, our optimization problem reduces to

$$\max_{I,M} P(I,M|D) = \max_{I,M} P(D|I,M)P(I|M). \tag{2.4}$$

2.1 Probability Distributions

Our data is a set of $N_{\rm Vis}$ observed visibilities $\{V_1^{\rm obs}, V_2^{\rm obs}, \cdots, V_{N_{\rm Vis}}^{\rm obs}\}$. If we have a certain model M and image I, we obtain model visibilities $\{V_k^{\rm mod}\}$ by simulating the interferometric observations over our image:

$$V_k^{\rm mod} = V^{\rm mod}(u_k, v_k) = \int_{-\infty}^{+\infty} A(x,y)I(x,y)\exp\left[2\pi i(u_k x + v_k y)\right] \frac{dx\,dy}{\sqrt{1-x^2-y^2}}\ , \tag{2.5}$$

where $\{u_k, v_k\}$ are the coordinates of baseline k in the (u,v) plane and A is the primary beam. We thus have a set of $N_{\rm Vis}$ model visibilities. Assuming that each visibility is independent from the others and Gaussian noise, the likelihood is

$$\begin{aligned} P(D|I,M) &= P(\{V_k^{\rm obs}\}_{k=1}^{N_{\rm Vis}} | \{V_k^{\rm mod}(I,M)\}_{k=1}^{N_{\rm Vis}}) = \prod_{k=1}^{N_{\rm Vis}} P(V_k^{\rm obs}|V_k^{\rm mod}) \\ &= \prod_{k=1}^{N_{\rm Vis}} \frac{1}{2\pi\sigma_k^2} e^{-||V_k^{\rm obs}-V_k^{\rm mod}||^2/2\sigma_k^2}. \end{aligned} \tag{2.6}$$

To obtain the image prior, $P(I|M)$, we calculate the statistical weight of a given distribution of counts (as in Piña & Puetter, 1993; Sutton & Wandelt, 2006). Consider a model consisting of n cells. In the case of a traditional image, each pixel would be a cell. There is a number of N quanta falling into these cells. These are intensity quanta of some size $\sigma_{\rm q}$. In the case of a pixelated image, the intensity in each pixel i would be $I_i = \sigma_{\rm q} N_i$, where I_i is the intensity in cell i. Each quantum could fall into any of the n cells, so the total number of possible configuration for the N quanta will be n^N. The probability of the image given the model is the probability of a certain state $\{N_1, N_2, \cdots, N_n\}$ that represents that image, where N_i is the number of quanta in cell i. Consider a given image configuration defined by a particular distribution $\{N_i\}$. The image distribution is not changed in the $N!$ possible redistributions of counts between cells, provided each N_i is constant. The $\prod_i N_i!$ swaps of counts within each cell keep the same image configuration. The model M consists of the Voronoi diagram and the total number of quanta (i.e. n, the position of the generators and N), thus the a priori probability is

$$P(I|M) = P(\{N_i\}|n, N) = \frac{N!}{n^N \prod_i N_i!}. \tag{2.7}$$

As explained above, σ_q is an intensity quantum. It is also possible to describe the number of quanta per cell using a flux quantum σ_i^F, where i is the index of the cell to which we associate the quantum. This flux quantum can be expressed in terms of the intensity quantum as $\sigma_i^F = \sigma_q A_i$, where A_i is the area of cell i. In this case, the number of quanta per cell is $N_i = F_i/\sigma_i^F$, where $F_i = I_i A_i$ is the flux of cell i. This leads to $N_i = I_i/\sigma_q$, which is the same expression for N_i obtained using the intensity quantum σ_q. Using these cell-dependent flux quanta, the probability of a quantum falling into each cell will be $\frac{1}{n}$ for every cell, leaving the a priori probability the same as Eq. 2.7.

2.2 MEM and Natural Entropy

In Bayesian theory, for a fixed model, the image I can be found by optimizing the a posteriori probability:

$$\begin{aligned}
\max_I P(I|D, M) &= \min_I(-\ln P(D|I, M)P(I|M)) \\
&= \min_I \sum_{k=1}^{N_{\text{Vis}}} \frac{||V_k^{\text{obs}} - V_k^{\text{mod}}||^2}{2\sigma_k^2} - \ln\left(\frac{N!}{n^N \prod_i N_i!}\right) \\
&= \min_I \frac{1}{2}\chi^2 - S, \qquad (2.8)
\end{aligned}$$

where we have defined the natural entropy $S = \ln\left(\frac{N!}{n^N \prod_i N_i!}\right)$. Sutton & Wandelt (2006) call the term $\ln(N!/\prod_i N_i!)$ the multiplicity prior. In the limit of large N_i,

$$\begin{aligned}
S &= \ln N! - N\ln n - \sum_i \ln N_i! \\
&\simeq N\ln N - N - N\ln n - \sum_i (N_i \ln N_i - N_i) \\
&= N\ln N - N\ln n - \sum_i N_i \ln N_i \\
&\simeq N\ln\frac{N}{n} - \sum_i N_i \ln N_i, \qquad (2.9)
\end{aligned}$$

and it can be seen that the Bayesian method is very similar to MEM in the sense that we are adjusting the image to the data while maximizing an entropy of the form of Eq. 2.9. VIR uses the natural entropy as a regularizing term.

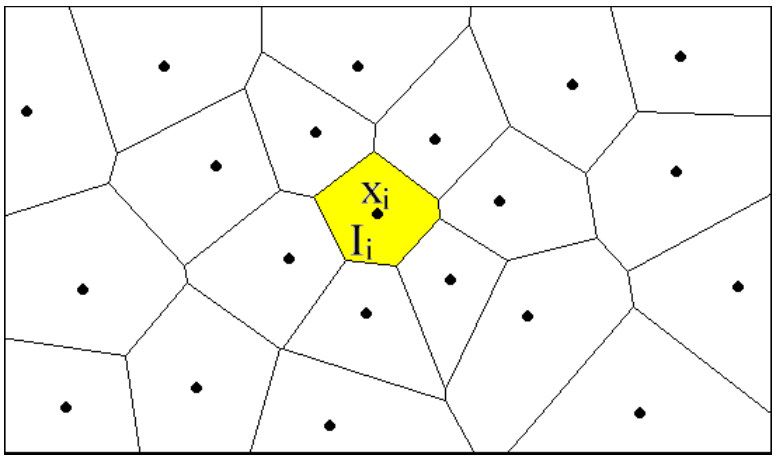

Figure 2.1: Example of a Voronoi diagram.

2.3 A New Image Model based on Voronoi Diagrams

A Voronoi diagram is a division of the Euclidian plane into n regions $\mathcal{V}_i$ defined by n points $\vec{x_i}$ (called sites or generators) such that every coordinate $\vec{x}$ in the space belongs to $\mathcal{V}_i$ if and only if $||\vec{x} - \vec{x_i}|| < ||\vec{x} - \vec{x_j}|| \ \forall \ j \neq i$. The result of the above definition is a set of polygons defined by the generators. Figure 2.1 shows an example of a Voronoi diagram. For further details on Voronoi diagrams see Okabe et al. (1992).

We propose a 2D Voronoi diagram in place of the usual pixelated, uniform grid, image as our model. We associate an intensity I_i to each of these polygons. The advantage of using a Voronoi diagram is that we can use just as many cells (i.e. free parameters) as the data requires. Our optimization parameters will be the position of each generator $\vec{x_i} = (x_i, y_i)$, and the intensity at each cell, I_i.

With our new model M consisting of n generators ($3n$ parameters, x_i, y_i and I_i for each generator), we can vary the number of free parameters as required by the optimization problem. We can see in equation (2.8) that the entropy S increases as the number of cells n decreases.

Chapter 3

Optimization

The optimization problem can be seen as a maximization of the a posteriori probability $\max_{I,M} P(I, M|D)$, or equivalently as a minimization of the more convenient merit function $L = \frac{1}{2}\chi^2 - S$. The conjugate gradient method (CG) is often used for minimization problems where derivatives can be easily calculated. Although it is usually fast in convergence, CG has the problem of converging on local minima depending on the initial condition. Monte Carlo methods (MC) and genetic algorithms (GA) are maximization algorithms that try to find the global maximum of the merit function. MC are used for generating samples that follow a certain density distribution, while GAs are heuristics. GAs are based on evolution of species and search the vector of parameters that maximize certain function. GAs do not have a proof of why they work. The use of other optimization algorithms is postponed to future work.

3.1 Conjugate Gradient

One of the most used optimization algorithms is the Conjugate Gradient method (CG). This method searches for the location $\vec{x}^*$ of the minimum of an objective function $f(\vec{x})$.

The CG method is motivated by the problem of solving big linear equation systems of the kind

$$A\vec{x} = \vec{b}, \tag{3.1}$$

where $\vec{x}$ is an unknown vector, $\vec{b}$ is a known vector and A is a square symmetric positive defined matrix. This problem is the same of minimizing the quadratic form

$$f(\vec{x}) = \frac{1}{2}\vec{x}^{\mathrm{T}}A\vec{x} - \vec{b}^{\mathrm{T}}\vec{x} + c, \tag{3.2}$$

where c is a constant scalar.

The CG method uses the gradient of $f(\vec{x})$ to generate *conjugate directions* in which the minimum is likely to be. On each iteration, it searches on that direction for the linear minimum and then calculates new conjugate directions from that point. For further details on the CG method, please refer to Press et al. (1992).

The CG method searches parameters space using the gradient of the function to be minimized. For our merit function L, the derivatives are

$$\frac{\partial L}{\partial x} = \frac{1}{2}\frac{\partial \chi^2}{\partial x} - \frac{\partial S}{\partial x}, \tag{3.3}$$

$$\frac{\partial \chi^2}{\partial x} = 2\sum_{k=1}^{N_{\mathrm{Vis}}} \frac{1}{\sigma_k^2}\mathrm{Re}\left((V_k^{\mathrm{mod}} - V_k^{\mathrm{obs}})^* \frac{\partial V_k^{\mathrm{mod}}}{\partial x}\right), \tag{3.4}$$

where x is any of the optimization parameters (x_i, y_i or I_i). The derivatives of the visibilities with respect to the position $\vec{x}_i = (x_i, y_i)$ of the i generator are

$$\frac{\partial V_k^{\mathrm{mod}}}{\partial x_i} = \sum_{j\in J_i}\left[(I_i - I_j)\sum_{l|\mathrm{pixel}\ l\epsilon a_{ij}} A_l \Delta t_l (M_x t_l + b_x) e^{(t_l c_2 + s_0 c_1)}\right], \tag{3.5}$$

$$\frac{\partial V_k^{\mathrm{mod}}}{\partial y_i} = \sum_{j\in J_i}\left[(I_i - I_j)\sum_{l|\mathrm{pixel}\ l\epsilon a_{ij}} A_l \Delta t_l (M_y t_l + b_y) e^{(t_l c_2 + s_0 c_1)}\right], \tag{3.6}$$

where I_i is the intensity in cell i, J_i is a set of the indices of the polygons adjacent to $\mathcal{V}_i$, a_{ij} is the edge which divides polygons $\mathcal{V}_i$ and $\mathcal{V}_j$, l sums over the pixels which intersect a_{ij}, A is the CBI primary beam. For further details see Appendix A.

The derivative of the visibilities with respect to the intensity of each cell I_i is

$$\frac{\partial V_k^{\mathrm{mod}}}{\partial I_i} = \frac{\sin(\pi u_k \Delta x)\sin(\pi v_k \Delta y)}{\pi^2 u_k v_k}\sum_{\mathrm{pixels}\ l\epsilon\mathcal{V}_i} A_l e^{2\pi i(u_k x_l + v_k y_l)}, \tag{3.7}$$

where $\vec{k}_k = (u_k, v_k)$ is the baseline corresponding to the pair of antennas k, Δx and Δy are the pixel width and height, and the sum extends over all the pixels inside $\mathcal{V}_i$.

The entropy only depends of the intensities I_i, so $\frac{\partial S}{\partial x_i} = \frac{\partial S}{\partial y_i} = 0$, then (see Appendix A.2)

$$\frac{\partial S}{\partial I_i} = \frac{1}{\sigma_{\mathrm{q}}}\left(\sum_{k=1}^{N}\frac{1}{k} - \ln n - \sum_{k=1}^{N_i}\frac{1}{k}\right). \tag{3.8}$$

3.2 Genetic Algorithm

Genetic Algorithms (GAs) are methods based on the evolution of species that find the vector of parameters that maximize certain function. As said before, GAs do not have a proof of why they work.

PIKAIA (see PIKAIA (online)) is a particular implementation of a genetic algorithm consisting in a set of points in parameter space (called *population*), which are initialized with random values. On each iteration the following procedure is made:

1. All points in the population are evaluated by the merit function to be maximized.
2. Pairs of points in the set are selected (*parents*) with a probability proportional to their function values and breeded producing two new points (*offspring*). This step is repeated until the number of offspring produced equals the number of individuals in the current population.
3. Replace the old population by the new offspring.

These steps are repeated until some termination criterion is satisfied.

PIKAIA's idea is to simulate the process of natural selection. It involves the breeding of the most qualified (or most probable) beings, which produce offspring with similar characteristics. The less probable points start to disappear on each generation. For the breeding process PIKAIA encodes the parameters to a string-like structure (*chromosome*). The creation of the new offspring involves crossover and mutation of their parents chromosomes. This creates a new pair of chromosomes that afterwards are decoded to obtain the offspring parameters.

Suppose you have a point $(x_1, x_2, \cdots, x_n)$ in the n-dimension parameter space. PIKAIA's encoding consists in directly fragmenting each parameter x_i into simple decimal integers and then concatenating them into a single string. The number of digits taken into account nd is given by the user. For example, if the parameter space has two dimensions, a point $(x_1, x_2) = (0.123456789, 0.987654321)$ using $nd = 8$ would be encoded into the string "1234567898765432". The decoding process is straightforward, the string should be splitted into two strings of size nd and transform them to their decimal form, $1234567898765432 \rightarrow (0.12345678, 0.98765432) = (x_1, x_2)$. Note that in the whole process a truncation loss has occurred (the original point was $(0.123456789, 0.987654321)$, but after decoding ends as $(0.12345678, 0.98765432)$).

PIKAIA's crossover process consist of splitting each chromosome in a unique random position and then swap their first and second parts in order to obtain two new chromosomes. For example, if we have, 1234567898765432 as one of the parents and 7654321023456789 as the other one, and we randomly select to cut them at site number 10, the splitted chromosomes would be $123456789 \leftrightarrow 8765432$ and $765432102 \leftrightarrow 3456789$. Then, the swap of the splitted parts would be $123456789 \leftrightarrow 3456789$ and $765432102 \leftrightarrow 8765432$, which finally produce the two offspring 1234567893456789 and 7654321028765432. After this, both new chromosomes are decodified in order to obtain the offspring parameters. In literature, this is called one point uniform crossover. "One point" because chromosomes are splitted in only one place, and "uniform" because the place to be splitted is selected by a random uniform distribution.

PIKAIA also involves a mutation process in which each gene (or digit) in the offspring chromosome can be changed with a small probability $P_{\rm m}$. If the gene is to be changed, a new digit is chosen randomly in order to replace the old one. For example, in our chromosome 7654321028765432, digit number two can be changed with a probability $P_{\rm m}$ to another random number, for example, 1, leaving the offspring as 7154321028765432. In literature, this is called one point uniform mutation.

Crossover and mutation can create individuals very similar or very different from their parents. It all depends on the splitting point in the crossover process and in the chromosome to be changed in the mutation process. These selections may be done in one of the leading digits or in one of the least significant digits of the chromosome, making some points to move further from their parents than others.

3.3 Monte Carlo Methods

Monte Carlo (MC) methods are computational techniques that make use of random numbers. They are used to generate random samples $\{\vec{x}^r\}_1^R$ from a given distribution $P(\vec{x})$ or to estimate expectations of functions under this distribution. For further details on Monte Carlo methods see MacKay (2004).

If having a method that generates samples from a given distribution, it is possible to use it to find a maximum of the given distribution. When generating consecutive samples, the most probable samples will be more frequently generated. These samples are the ones that give bigger values for our distribution $P(\vec{x})$. This way, the global maximum can be found by leaving the program running for a large amount of time.

3.3.1 The Problem of Generating Samples from a Given Density

Generating random samples from a a given distribution is not as simple as it seems. We will assume that our objective distribution $P(\vec{x})$ can be evaluated, at least to a multiplicative constant. This means that a function $P^*(\vec{x})$ can be evaluated such that

$$P(\vec{x}) = \frac{P^*(\vec{x})}{Z}, \tag{3.9}$$

where Z is a constant. Generally, we do not know the normalizing constant

$$Z = \int P^*(\vec{x}) d^N x, \tag{3.10}$$

where N is the dimension of $\vec{x}$. But even if we knew it, the problem is still complicated.

A naive way of approaching the problem is to divide each i dimension of the $\vec{x}$ space into M points, $\{x_i^j\}_{j=1}^M$. Then, we can evaluate the function at each of these cells obtaining a set $\{p_i^j\}_{j=1}^M$ and use a standard uniform random number generator to generate a number p between 0 and $\sum_j p_i^j$. We proceed to locate the function value $p_i^{j^*}$ such that $\sum_{j=1}^{j^*} p_i^j < p < \sum_{j=1}^{j^*+1} p_i^j$ and return $x_i^{j^*}$.

The problem with the previous naive algorithm is that when the parameter space has many dimensions, the number of function evaluations increases exponentially with the number of dimensions; the number of function calculations is M^N. For a large number of dimensions this is really expensive to calculate. To solve this problem more efficient MC methods exist.

Markov chain Monte Carlo methods involve a Markov process that generates a set of states $\{x^{(t)}\}_{t=1}^T$. Each one of this states $x^{(t)}$ is generated from a probability density that depends on the previous step $x^{(t-1)}$. As each step is not independent from the previous one, a number of iterations is required in order to generate a pair of independent samples. This is done by a random walk that often takes a lot of steps. There are some efficient MC methods that try to accelerate this random walk.

3.3.2 Metropolis-Hasting

Metropolis-Hasting (MH) uses a proposal density $Q(\vec{x}', \vec{x}^{(t)})$ which depends of the current state $\vec{x}^{(t)}$. The proposal density $Q(\vec{x}', \vec{x}^{(t)})$ may be any simple distribution from where we can generate samples. Assuming that we can evaluate $P^*(\vec{x})$ for every $\vec{x}$, we generate a new state $\vec{x}'$ from the proposal density $Q(\vec{x}', \vec{x}^{(t)})$. To decide if the new state is accepted we calculate a value

$$a = \frac{P^*(\vec{x}')Q(\vec{x}^{(t)}, \vec{x}')}{P^*(\vec{x}^{(t)})Q(\vec{x}', \vec{x}^{(t)})}. \tag{3.11}$$

If $a \geq 1$ then the new state is accepted, otherwise, the new state is accepted with probability a. If the new state is accepted, $\vec{x}^{(t+1)} = \vec{x}'$, if not accepted, $\vec{x}^{(t+1)} = \vec{x}^{(t)}$.

If the proposal density is symmetrical, then $Q(\vec{x}^{(t)}, \vec{x}') = Q(\vec{x}', \vec{x}^{(t)})$, and $a = P^*(\vec{x}')/P^*(\vec{x}^{(t)})$. We used a symmetrical Gaussian function as our proposal density. This particular case is sometimes called just the Metropolis method. Figure 3.1 shows an example of a Metropolis method step in one dimension. The proposal density is a Gaussian centered on the current state $\vec{x}^{(t)}$.

Each sequential step $t = 1, \ldots, T$ does not necessarily generate an independent sample from the previous one. In order to generate independent samples $r = 1, \ldots, R$, the method needs to compute a minimum number of steps depending on the parameter

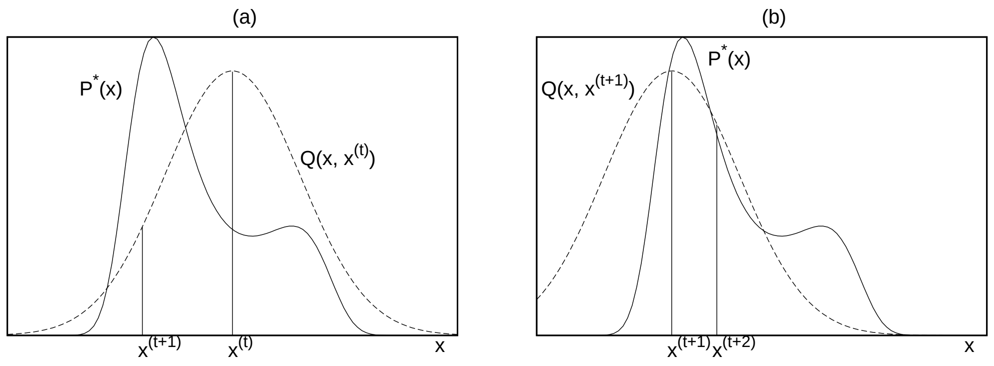

Figure 3.1: Example of a Metropolis-Hasting step using a Gaussian distribution as our proposal density Q. (a) The Gaussian centered at $\vec{x}^{(t)}$ generates $\vec{x}^{(t+1)}$. (b) The new Gaussian is centered at $\vec{x}^{(t+1)}$.

space. It can be shown, that for any positive Q, as $t \to \infty$ the probability distribution of $\vec{x}^{(t)}$ tends to $P(\vec{x})$.

We will consider a proposal distribution with a length scale ϵ (in the case of a Gaussian, ϵ would be its variance) and a probable region with a longest length scale of L (See Figure 3.2). In the case of using a large $\epsilon \gg 1$, the proposed state is likely to fall into very unlikely regions, so few states would be accepted. On the other hand, if we use a proposed density with small ϵ, the method will take many steps to generate two independent samples. It can be shown that after T steps of size ϵ the state is likely to have moved a distance about $\sqrt{T}\epsilon$. Thus, in order to obtain a roughly independent sample, we need to realize $T \simeq (L/\epsilon)^2$ steps. This is assuming that we accept each step. If only a fraction f of the steps is accepted on average, then we should multiply the previous expression by $1/f$.

In the case of having a N-dimensional probability distribution $P(\vec{x})$, the analysis is straightforward. Take a spherical Gaussian with standard deviation of ϵ as proposed distribution Q . Without loss of generality, we can assume that P is a separable distribution aligned with the axes of $\vec{x}$, having a standard deviation σ_i in direction i. Let us assume also that ϵ is adjusted so that approximately all steps are accepted. Under these assumptions, each variable will evolve independently from the rest. Lets suppose that the biggest standard deviation of P is $\sigma^{\max}$, and the smallest $\sigma^{\min}$. As each variable evolves independently, the number of steps needed to get an independent sample will depend of the largest length scale $\sigma^{\max}$. As seen before, this number of steps is $T \simeq (\sigma^{\max}/\epsilon)^2$. As also explained before, if we choose a large ϵ, most of the states will be rejected. On the other hand if we choose a small ϵ we will need a lot of steps to generate an independent sample. Thus, we need to use the biggest ϵ that assures us not to fall into unlikely regions. This is controlled by the smallest length

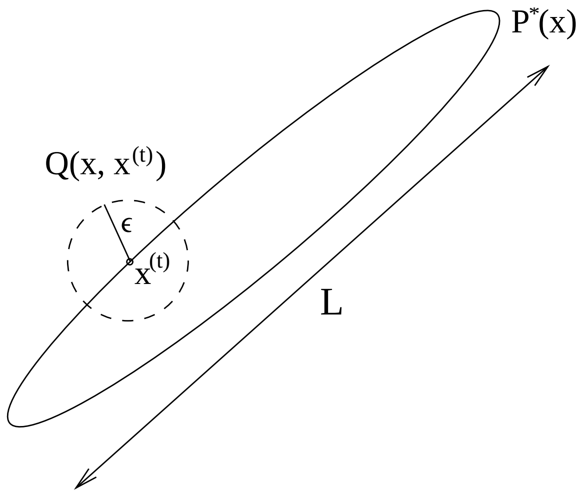

Figure 3.2: MH method using a spherical proposed density Q of variance ϵ and a density P^* of length scale L.

scale, so we should choose $\epsilon = \sigma^{\min}$, leading us to the conclusion that the number of steps required to generate an independent sample will be $T \simeq (\sigma^{\max}/\sigma^{\min})^2$.

3.3.3 Gibbs Sampling

Gibbs sampling (GS) is a MC method used for parameter spaces of two or more dimensions, and a probability density $P(\vec{x})$ such that the conditional probabilities $P(x_i|\{x_j\}_{i\neq j})$ are easy to work with. GS is very similar to MH in the sense that we also generate samples from a given density Q. This density is defined by the conditional probabilities of $P(\vec{x})$.

On each iteration, we start from the current state $\vec{x}^{(t)}$ and generate the new step coordinate by coordinate. We first generate $x_1^{(t+1)}$ following the conditional density $P(x_1^{(t+1)}|\{x_j^{(t)}\}_{j=2}^N)$, where N is the size of the parameter space. On the next iteration, we generate $x_2^{(t+1)}$ following the conditional density $P(x_2^{(t+1)}|x_1^{(t+1)}, \{x_j^{(t)}\}_{j=3}^N)$, and so on until generating all coordinates of $\vec{x}^{(t+1)}$. For each iteration we need to generate each coordinate as follows:

$$\begin{aligned}
x_1^{(t+1)} &\rightarrow P(x_1|x_2^{(t)}, x_3^{(t)}, \ldots, x_N^{(t)}), \quad (3.12)\\
x_2^{(t+1)} &\rightarrow P(x_2|x_1^{(t+1)}, x_3^{(t)}, \ldots, x_N^{(t)}), \quad (3.13)\\
x_3^{(t+1)} &\rightarrow P(x_3|x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_N^{(t)}), \quad (3.14)\\
&\cdots\\
x_N^{(t+1)} &\rightarrow P(x_N|x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_{N-1}^{(t+1)}). \quad (3.15)
\end{aligned}$$

Figure 3.3 shows an example of GS in two dimensions.

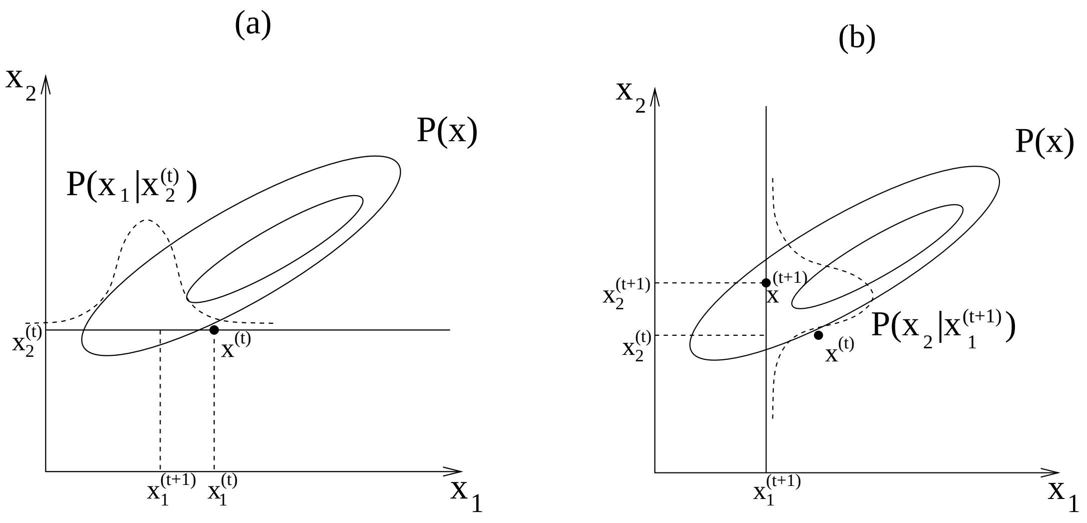

Figure 3.3: A Gibbs sampling step in two dimensions. (a) Starting from $\vec{x}^{(t)}$ we generate $x_1^{(t+1)}$ following de conditional density $P(x_1|x_2^{(t)})$. (b) We generate $x_2^{(t+1)}$ following de conditional density $P(x_2|x_1^{(t+1)})$.

GS can be viewed as a MH method with target density $P(\vec{x})$ and proposed density defined by its conditional densities, where all states are accepted. Because of this, all properties satisfied for MH are satisfied by GS too. The number of steps needed to obtain independent samples, given a density P^* of length scale L and conditional probabilities of length scale ϵ, will be $T \simeq (L/\epsilon)^2$. Though GS explores the space through a slow random walk, it has the advantage that it does not need any adjustable parameters.

3.3.4 Efficient Monte Carlo Methods

The goal of efficient Monte Carlo methods (EMC) is to accelerate the slow random walk behavior of Monte Carlo methods. EMC methods aim to reduce the number of steps to produce independent samples.

Hamiltonian Monte Carlo

We programmed the HMC method, but realized a practical problem that made it not suitable for our problem. On each 'leapfrog' step, the gradient of the function must be evaluated. In our case, this gradient takes too much time to calculate. This makes the generation of independent samples, though in fewer steps than MH, slower.

The Hamiltonian Monte Carlo method (HMC) is a Metropolis Hasting method, based on Hamiltonian dynamics, that uses gradient information in order to accelerate the generation of independent samples. This method is used to generate samples from a density that can be written in the form

$$P(\vec{x}) = \frac{e^{-E(\vec{x})}}{Z}, \tag{3.16}$$

where $E(\vec{x})$ and its gradient can be evaluated.

The state parameter space is augmented by a momentum variable $\vec{p}$. On each step two proposals are made, first a momentum proposal, and second, a proposal that changes $\vec{x}$ and $\vec{p}$. For the second proposal, we introduce the Hamiltonian

$$H(\vec{x},\vec{p}) = E(\vec{x}) + K(\vec{p}), \tag{3.17}$$

where $K(\vec{p}) = \frac{1}{2}\vec{p}\cdot\vec{p}$ is a 'kinetic energy'. The idea is to create samples from the joint density

$$P_H(\vec{x},\vec{p}) = \frac{1}{Z_H}e^{-H(\vec{x},\vec{p})} = \frac{1}{Z_H}e^{-E(\vec{x})}e^{-K(\vec{p})}. \tag{3.18}$$

Because this distribution is separable, when generating samples of $\vec{x}$ and $\vec{p}$ the distribution of the state variable $\vec{x}$ will be the desired density $P(\vec{x})$.

For generating samples and momentums that follow Eq. 3.18, the first proposal draws a new momentum from the Gaussian density $\frac{1}{Z_K}e^{-K(\vec{p})}$, which is always accepted. Once we have this momentum, the second proposal generates a new state $\vec{x}$ determined by the generated momentum, and the momentum changes using the gradient of $E(\vec{x})$, in accordance with the equations

$$\dot{\vec{x}} = \vec{p}, \tag{3.19}$$

$$\dot{\vec{p}} = -\frac{\partial E(\vec{x})}{\partial \vec{x}}. \tag{3.20}$$

As $\vec{x}$ is always moved in the direction of the momentum, the average distance goes linearly with the number of steps. This way, the generation of independent samples requires less iterations. This second proposal is accepted in accordance with the Metropolis Hasting rule. If the Hamiltonian dynamics simulation is perfect, then every proposal would be accepted, because $H(\vec{x},\vec{p})$ is a constant. But as usually this is not the case, some proposals will be rejected.

Figure 3.4 shows the Hamiltonian Monte Carlo algorithm. T 'leapfrog' steps of size ϵ are done following the momentum. The rule of acceptance is evaluated by a difference of the Hamiltonian rather than a division of the probabilities in order to avoid exponential calculations.

```
**HamiltonianMonteCarlo** ($\vec{x}$)
Input: Initial state $\vec{x}$.
$\vec{g} = \vec{\nabla} E(\vec{x})$;
$E = E(\vec{x})$;
for $l = 1$ to $L$ do
   $p =$ Normal (0, 1);;
   $H = \vec{p} \cdot \vec{p}/2 + E$;
   $\vec{x}_{\text{new}} = \vec{x}$; $\vec{g}_{\text{new}} = \vec{g}$;
   for $\tau = 1$ to $T$ do
      $\vec{p} = \vec{p} - \epsilon \cdot \vec{g}_{\text{new}}/2$;
      $\vec{x}_{\text{new}} = \vec{x}_{\text{new}} + \epsilon \cdot \vec{p}$;
      $\vec{g}_{\text{new}} = \vec{\nabla} E(\vec{x}_{\text{new}})$;
      $\vec{p} = \vec{p} - \epsilon \cdot \vec{g}_{\text{new}}/2$;
   end for
   $E_{\text{new}} = E(\vec{x}_{\text{new}})$;
   $H_{\text{new}} = \vec{p} \cdot \vec{p}/2 + E_{\text{new}}$;
   $\Delta H = H_{\text{new}} - H$;
   if $(\Delta H < 0)$ or (random() $< e^{-\Delta H}$) then
      $\vec{x} = \vec{x}_{\text{new}}$; $\vec{g} = \vec{g}_{\text{new}}$; $E = E_{\text{new}}$;
   end if
end for
```

Figure 3.4: Hamiltonian Monte Carlo algorithm.

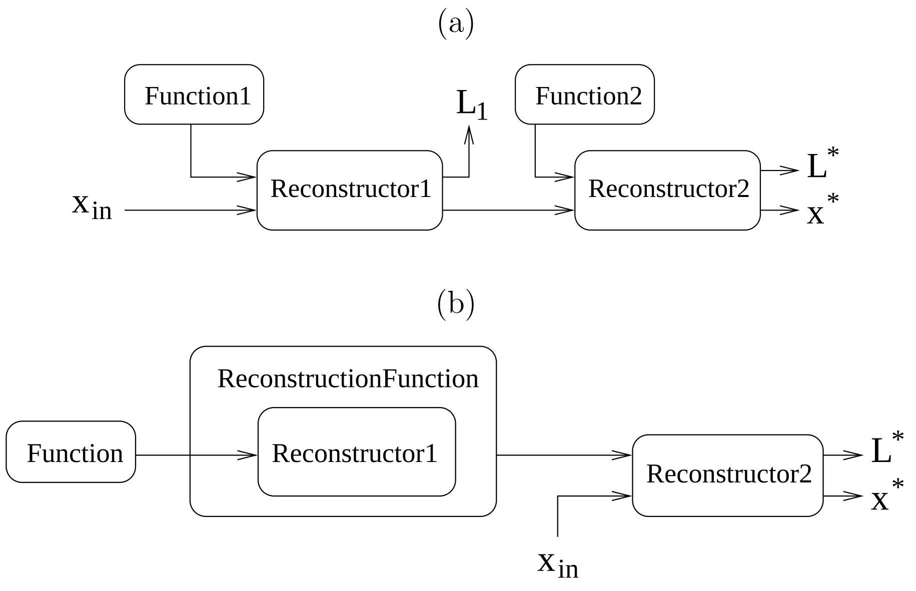

Figure 3.5: Hybrid methods. Both methods receive an initial point $\vec{x}_{\text{in}}$ and return the optimal point $\vec{x}^*$ and value L^*. (a) Two independent reconstructions are done. The output of Reconstructor1 is used as input by Reconstructor2. (b) Reconstructor2 uses as optimization function ReconstructionFunction, which returns the optimal value returned by Reconstructor1.

3.4 Hybrid Methods

Some of the algorithms described above can be useful to approach the global optimum, but fail in obtaining the exact optimum. On the other hand, the CG method fails in obtaining the global optimum, but obtains almost exactly local optima given the initial condition.

In order to use properties from diverse optimization algorithms we mixed some of them. This was done in two ways. The first way was to make a reconstruction with a given algorithm and then give its output as input to the second algorithm. Second, we created a reconstruction function which, given a point $\vec{x}$, does a reconstruction starting from it and returns the optimal value of the function. Figure 3.5 shows diagrams for both cases. An example of the first case is to apply a MH and then apply a CG method over the maxima obtained by the MH. An example of the second case is to use a GA in which the merit function is the local minimum obtained by a CG method applied over each individual.

3.5 Incremental Algorithms

Usually, when the dimension of the parameter space is smaller, the optimization process takes less time than when having a larger space. We created incremental algorithms (IA) that start optimizing with few polygons. On each iteration the number of generators increases and a new optimization can be done.

To decide the new values to be added, we created an *incrementator*. This incrementator knows what rule to follow in order to add new parameters.

3.5.1 Fitting a Voronoi Tessellation to an Image

A particular case of IA is a method that fits a Voronoi diagram to a pixelated image. We start with a mesh consisting in only one polygon. The iterator, on each iteration, adds a new polygon inside the polygon with greatest error. We calculate the error per polygon as

$$e_i^2 = \sum_l (I_i - I_l^{\text{im}})^2, \tag{3.21}$$

where the sum runs over all the pixels that fall inside polygon i, I_i is the intensity of that polygon and I_l^{im} is the intensity of pixel l in the image to be fitted. As said before, in each iteration we add a new polygon inside the one with the greatest error. The new generator is inserted in the position of the pixel that has the most different intensity value with respect to the mesh intensity.

3.5.2 Incremental Conjugate Gradient

We also used our incremental algorithm to create an incremental conjugate gradient method. We started with a Voronoi tessellation consisting of only two generators. We applied a CG to this model and then add a new generator to the mesh. On each iteration we did the same: add a new generator and apply a CG to the tessellation.

We created different incrementators in order to decide where to add the new generator. An example is to add the new generator randomly on the tessellation and give the new generator the intensity of the previous mesh at that point. Another example is to add the generator on the edge of the mesh that has the greatest difference between adjacent polygons.

3.6 VIR

VIR (from Voronoi image reconstruction) uses the CG method from Press et al. (1992) and searches for the position and intensities of the Voronoi polygons, x_i, y_i, I_i, that minimize our merit function L. The CG method modifies the intensities and also moves the positions of the Voronoi generators. This causes the shape of the Voronoi polygons to change as well. A general problem with CG is that it usually converges on local minima. For VIR in particular, though Voronoi polygons intensities adjust quite fine, the positions of the generators are difficult to modify substantially. The VIR parameter space is smooth enough in intensity space to converge on a good solution. But the parameter space in cell generator positions is very structured, and CG is quickly stuck on local minima.

Due to the fact that CG easily falls into local minima, we needed a good approximation for the initial Voronoi diagram. For this purpose we used a pixelated version of the Bayesian algorithm, where the model was a uniform grid. We decided to do a pure χ^2 (maximum likelihood, ML) reconstruction and use the fifth CG iteration as our starting image. We chose this particular iteration because on inspection the modeled images were still smooth. Pure χ^2 reaches convergence with noisy images, where the true image is unrecognizable. We then fitted a Voronoi diagram to the image (see Sec. 3.5.1) and ran CG using the positions and intensities of the generators as our free parameters, which led to our final reconstruction. Truncation to a level of 10^{-5} quanta was used to enforce positivity.

VIR can be seen as an hybrid method. It combines an incremental algorithm to adjust a tessellation to an image and a CG method.

Chapter 4

Design and Implementation

We have designed, and implemented in C++, all our reconstruction methods with modules which include algorithms for:

- the generation of the Voronoi diagram
- calculation of model visibilities
- calculation of the merit function L to be optimized as well as its derivatives
- fitting a Voronoi diagram to an image
- the different optimization methods
- the optimization of the number of polygons

Cabrera (2005) has implemented a maximum likelihood reconstruction method using a Voronoi mesh as model and programmed it in C. We used the same Voronoi tessellation generator and χ^2 function libraries in our tools, but made interfaces in order to use C++ as programming language.

4.1 Use Cases

Figure 4.1 shows the reconstruction use cases. In our case the only actor is the program user, who can reconstruct by different methods. Table 4.1 shows the description of each use case.

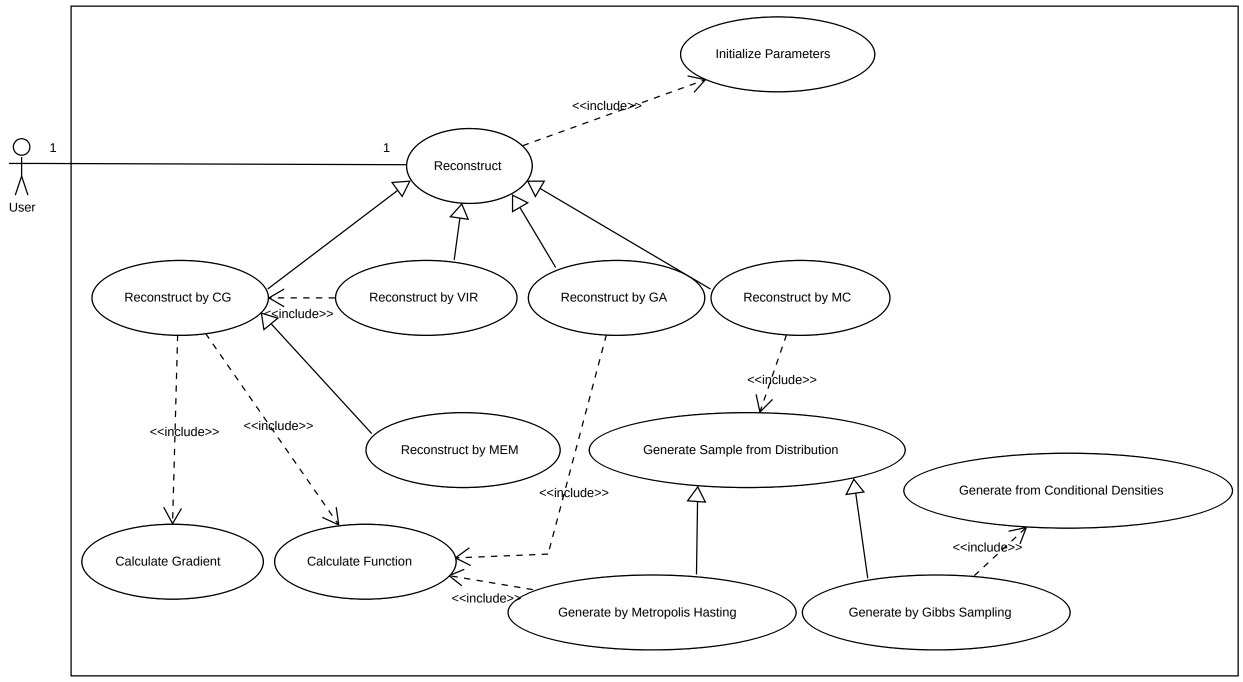

Figure 4.1: Use cases diagram.

Use Case	Actor	Description
Reconstruct	User	Makes a reconstruction.
Reconstruct by CG	User	Makes a reconstruction using the conjugate gradient method.
Reconstruct by VIR	User	Makes a reconstruction using the VIR method.
Reconstruct by GA	User	Makes a reconstruction using the genetic algorithm.
Reconstruct by MC	User	Makes a reconstruction using Monte Carlo methods.
Reconstruct by MEM	User	Makes a maximum entropy reconstruction using a fixed pixel grid model.
Initialize Parameters	User	Initializes the parameters to start with as initial condition.
Calculate Function	User	Calculates the value of the function to be optimized.
Calculate Gradient	User	Calculates the gradient of the function to be optimized.
Generate Sample from Distribution	User	Generates a sample following a certain probability distribution.
Generate by Metropolis Hasting	User	Generates a sample following a certain probability distribution using the Metropolis-Hasting method.
Generate by Gibbs Sampling	User	Generates a sample following a certain probability distribution using Gibbs sampling.
Generate from Conditional Densities	User	Generates one coordinate of the parameter space that follows certain conditional densities.

Table 4.1: Use cases description.

4.2 Module Design

The optimization procedure generally involves the same steps, independent of the method to be used. We start with an initial set of parameters, calculate the merit function and see if the function value satisfies a certain criterion. In the case this criterion is satisfied, we return the optimal parameters and function value. If it is not satisfied, we move the parameters following a certain rule and calculate the function again. Figure 4.2 shows the module diagram for the optimization procedure.

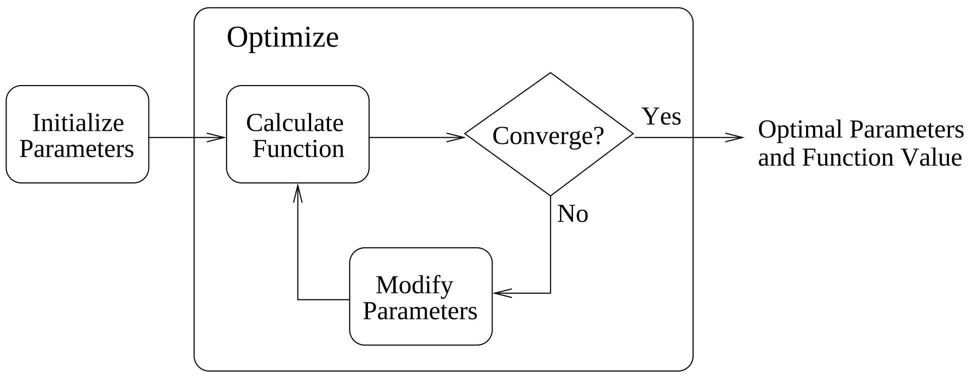

Figure 4.2: Optimization module design.

For all our reconstructions, we used L (CG, GA) or e^{-L} (MC methods) as our merit function. In the case of the CG and HMC methods the modification of parameters block includes the calculation of the gradient of L. Figure 4.3 shows the module diagrams for the calculation of L and its gradient.

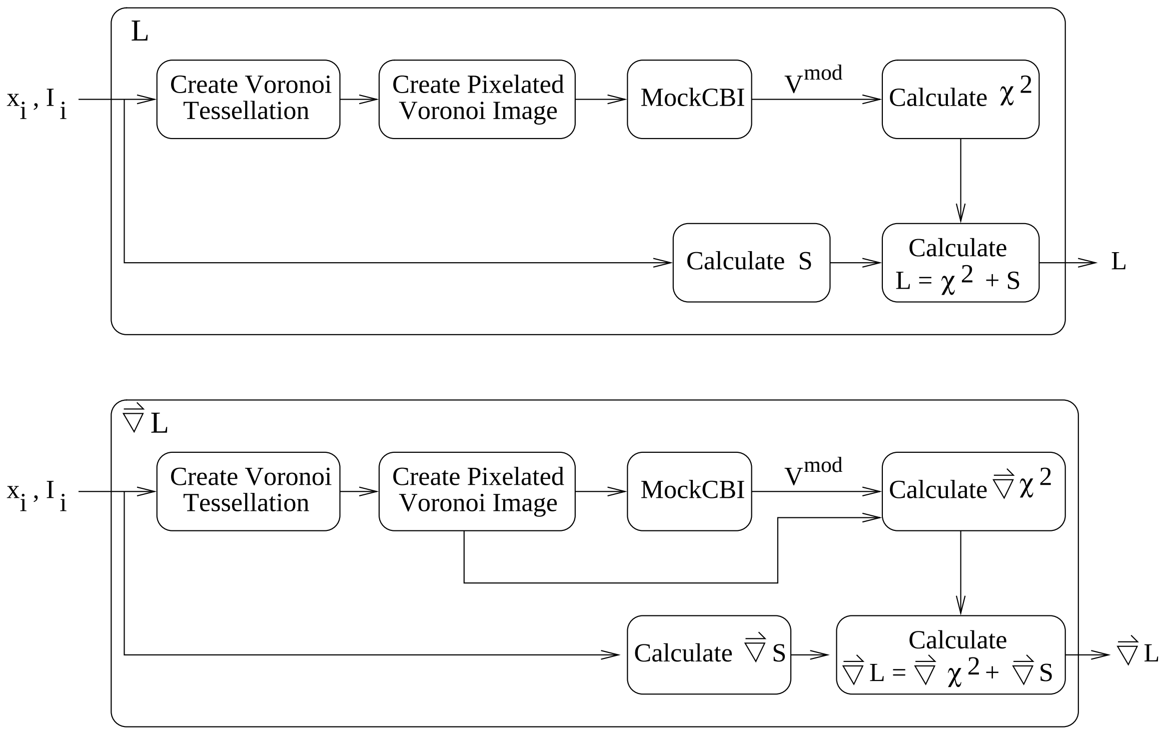
L
x_i, I_i
Create Voronoi Tessellation
Create Pixelated Voronoi Image
MockCBI
V^{mod}
Calculate χ^2
Calculate S
Calculate $L = \chi^2 + S$
L
$\vec{\nabla} L$
x_i, I_i
Create Voronoi Tessellation
Create Pixelated Voronoi Image
MockCBI
V^{mod}
Calculate $\vec{\nabla}\chi^2$
Calculate $\vec{\nabla} S$
Calculate $\vec{\nabla} L = \vec{\nabla}\chi^2 + \vec{\nabla} S$
$\vec{\nabla} L$

Figure 4.3: Function and gradient calculation of L module design.

4.3 Class Design

Class diagrams are unified model language (UML) diagrams used in object oriented programming (OOP) to describe the structure of a system. This is done by showing different classes, their attributes and relations. There are different kinds of relations: **inheritance**, **association**, **composition** and **aggregation**. Inheritance or generalization (represented in the diagram by a triangle) is used when a class (the subclass) is a specialized form of the other (the superclass). The subclass inherits all the attributes from the superclass. The association relation is used when two classes are connected in any way, and is represented in the diagram by a line linking these classes. Aggregation indicates that a class is formed by a collection of others. Aggregation relations are represented by a hollow diamond shape. Composition is a stronger variation of the aggregation relation. In the case the container instance is destroyed, every instance contained by it must be destroyed as well. Compositions are represented by a filled diamond.

The class diagram for our tool is divided into three parts. Figure 4.4 shows the different reconstructor classes plus other classes related to them. Figure 4.5 shows function classes. Notice that the *Reconstructor* class is also shown, as needed by the *ReconstructorFunction* class. Figure 4.6 shows the probability distribution classes. In this figure, some function classes needed by the probability densities are also shown.

In order to make a reconstruction, a *Reconstructor* object is needed (see Figure 4.4). This *Reconstructor* is the one holding all the attributes required to search for the optimum. Different kinds of reconstructors exist, each one with a different algorithm: conjugate gradient reconstructor (*CGReconstructor*), genetic algorithm reconstructor (*GeneticReconstructor*), Bayesian reconstructor (*BayesianReconstructor*), VIR reconstructor (*VIRReconstructor*), incremental reconstructor (*IncrementalReconstructor*) and Hamiltonian reconstructor (*HamiltonianReconstructor*). In order to make an optimization, a *run()* method must be called from the abstract class *Reconstructor*. This method is the one in charge of searching the parameter space for the optimum. In the future, if we want to add a new optimization algorithm, we only need to create a class that inherits from *Reconstructor* and program the algorithm in the *run()* routine.

CGReconstructor, *IncrementalReconstructor* and *GeneticReconstructor* need a function to be optimized. This function is given by the abstract class *Function* (see Figure 4.5). This class has methods for calculating the function value given a set of parameters (*double f(double *p)*, where p is the parameters vector) and its gradient (*void df(double *p, double *g)*, where p is the parameters point and g the gradient to be computed). The abstract class *CBIFunction* represents the functions used for the CBI, while the abstract class *VoronoiFunction* has the Voronoi mesh as attribute. *BayesVoronoiCBIFunction* is the class used to calculate our L merit function. It in-

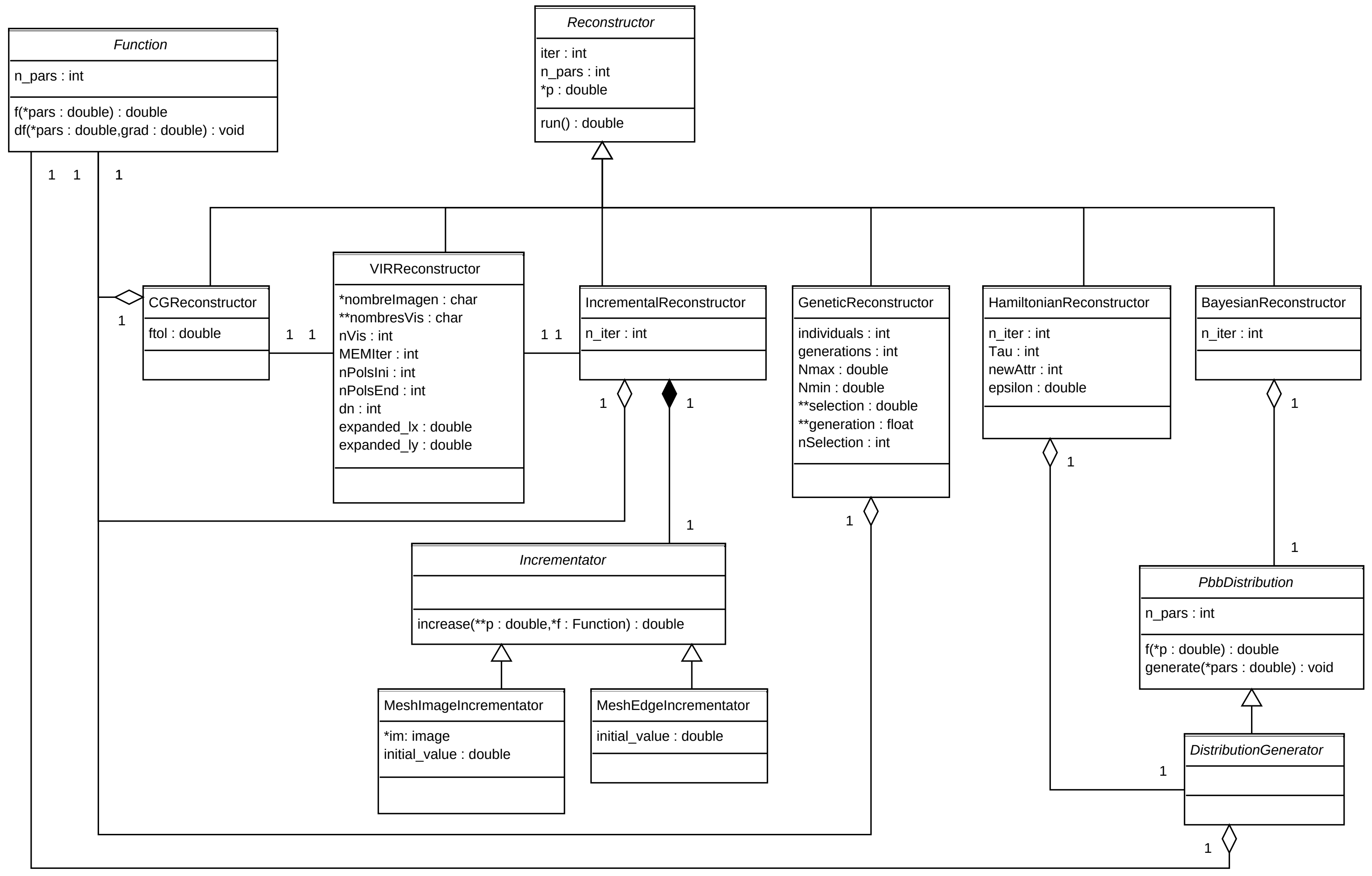

Figure 4.4: Reconstructors class diagram.

herits from both *CBIFunction* and *VoronoiFunction*. The *PbbBayesVoronoiFunction* calculates not L but the a posteriori probability e^{-L}. The *MemCBIFunction* calculates the L_{MEM} values for a fixed pixel grid model. It requires another function to be used as the entropy term. We created two entropy function classes: *ILogIEntropyFunction* ($S = -\sum_i I_i \log(I_i/M)$ described on Chapter 1) and *NaturalEntropyFunction* (the natural entropy $S = \ln\left(\frac{N!}{n^N \prod_i N_i!}\right)$). *ReconstructorFunction* is a function that uses a reconstructor, it makes a reconstruction and returns the optimal value. The purpose of this function is to combine it to a reconstructor in order to create an hybrid method. *DividedFunction* and *NegativeFunction* are used to change between maximization or minimization of a function. Both receive a function f. *DividedFunction* returns the evaluation and gradient of $1/f$, while *NegativeFunction* returns the evaluation and gradient of $-f$. This way, if we wish to use, for example, a maximization algorithm over a function to be minimized, we can do it by using these functions.

The *HamiltonianReconstructor* and *BayesianReconstructor* use probabilities distributions. Figure 4.6 shows the diagram concerning these distributions. Class *PbbDistribution* is an abstract class representing a probability density. It has a method that evaluates that density in a certain point of the parameter space (*double f(double *p)*, where p is the parameters vector) and a method that generates a sample following that distribution (*void generate(double *p)*, where p is the sample to be generated). We created three subclasses: *NormalDistribution*, *GibbsDistribution*, and *DistributionGenerator*. *GibbsDistribution* uses the Gibbs sampling algorithm for generating samples from the conditional densities. *DistributionGenerator* is an abstract class that obtains samples that follow a given function. A particular case is the *MetropolisDistribution*, which uses the *NormalDistribution* as proposal density.

The class diagram shows how easy is to extend the code in order to add new reconstructors, functions or probability distributions. Figure 4.7 shows the pseudocode of the steps needed to make a reconstruction using a *CGReconstructor* over a *BayesVoronoiCBIFunction*. In the case of creating a new reconstructor or function, the procedure is similar (see Figure 4.8).

4.4 Quantum Size and Number of Voronoi Generators

An important issue to consider is the size of the quantum σ_{q}. Sutton & Wandelt (2006) treat σ_{q} as a free parameter. But, as we now explain, σ_{q} was held constant in this implementation of VIR. We treat the number of quanta per cell as a continuous variable in order to use the CG method. Entropy is maximized at $\sigma_{\text{q}} = \infty$, where, for a given configuration of intensities $\{I_i\}$, $N = 0$ and $S = 0$. For every other value

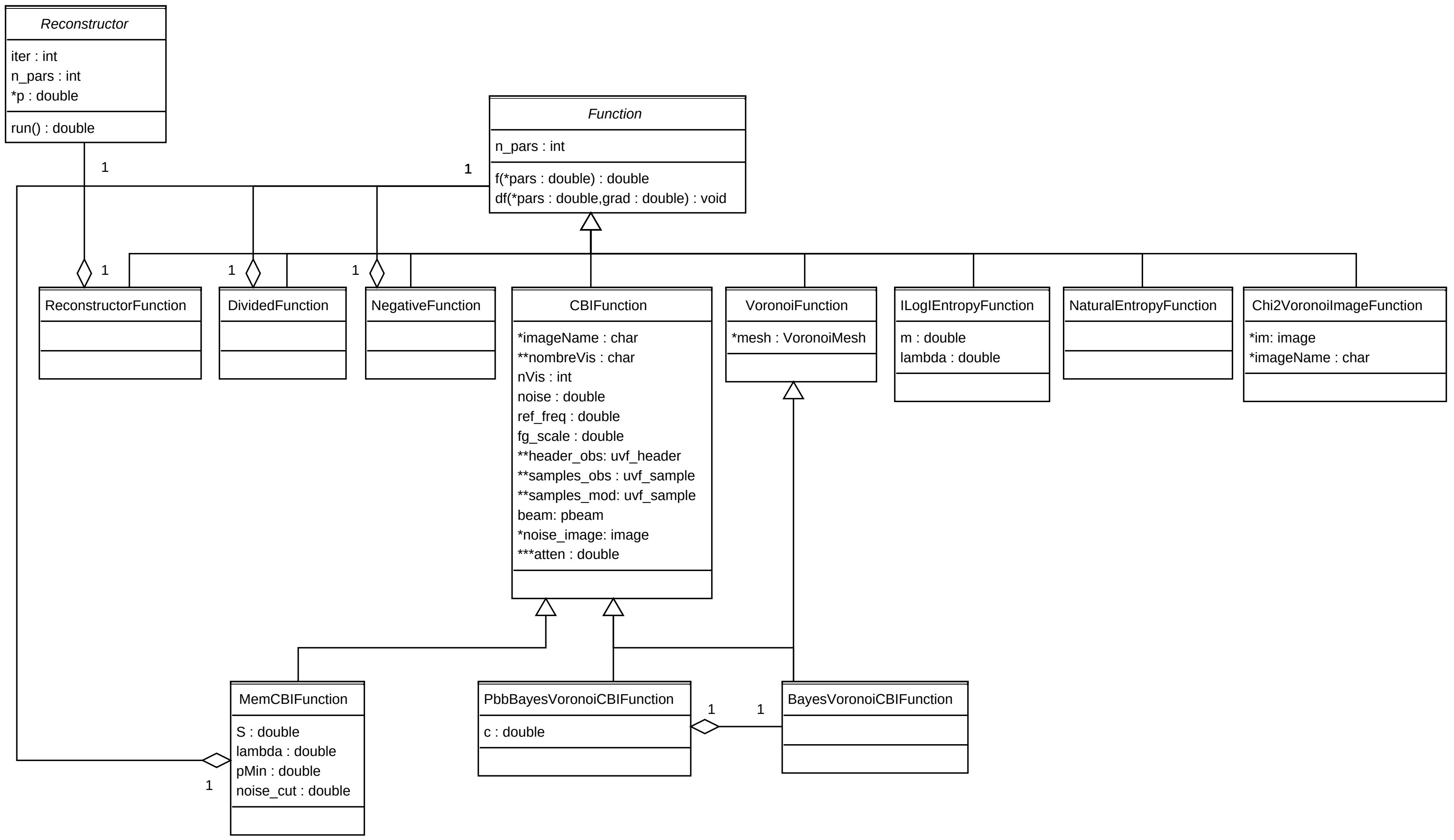

Figure 4.5: Functions class diagram.

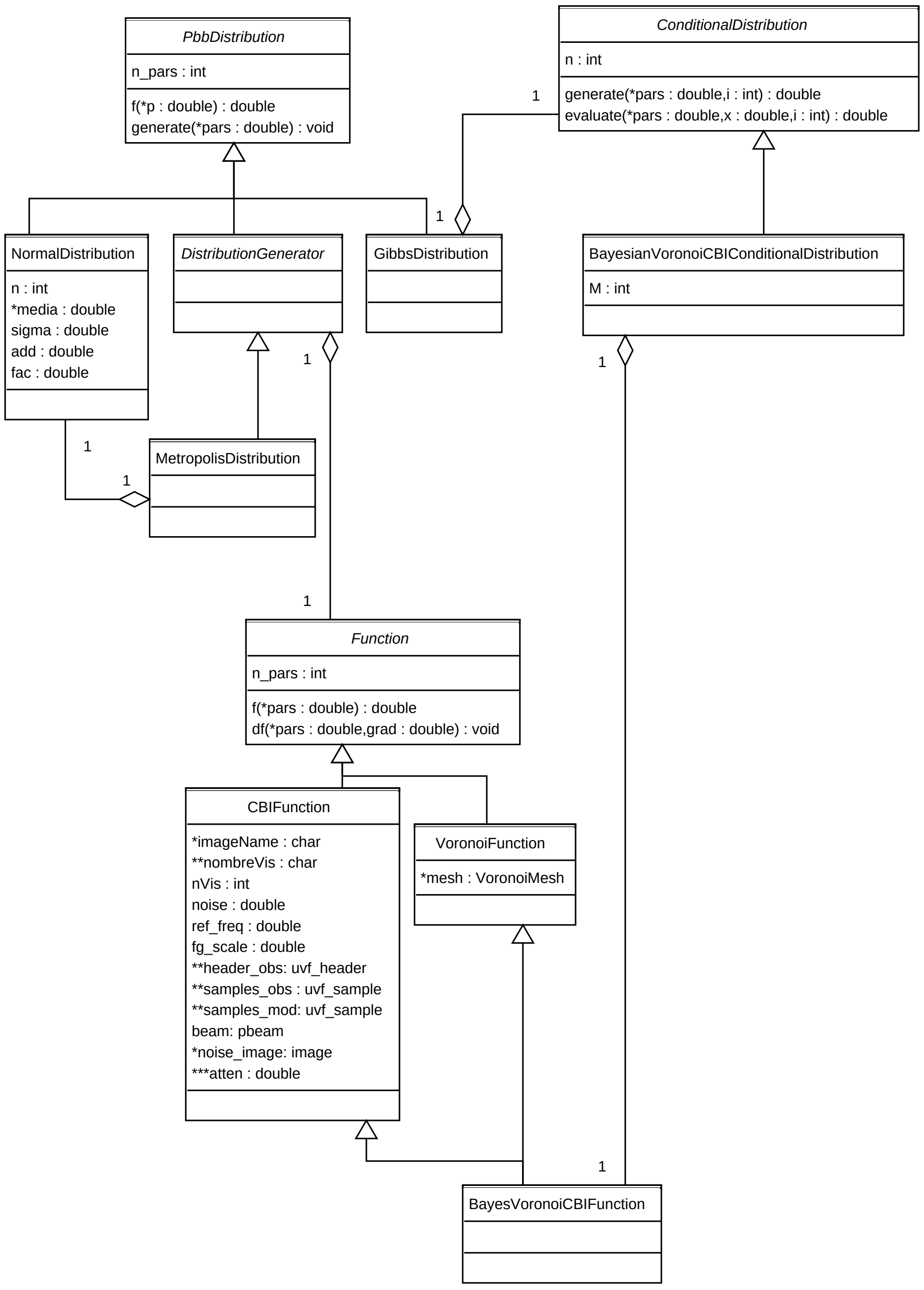

Figure 4.6: Probability distributions class diagram.

of N, the entropy will be negative. This means that even for large $\sigma_{\rm q}$, the intensities $I_i = \sigma_{\rm q} N_i$ can have reasonable values (using small N_i). Figure 4.9 shows S as a function of N for 51 Voronoi generators and 3 different intensity distributions using the model tessellation of Figure 5.9a. We considered: 1- the VIR intensities of Figure 5.9a, 2- a uniform intensity distribution image ($N_i = \frac{N}{n}\ \forall\ i$), 3- a spike where all N are only

```
BayesVoronoiCBIFunction *f = new BayesVoronoiCBIFunction();
CGReconstructor *r = new CGReconstructor(f);
r→run();
```

Figure 4.7: Pseudocode to make a reconstruction.

```
NewFunction *f = new NewFunction();
NewReconstructor *r = new NewReconstructor(f);
r→run();
```

Figure 4.8: Adding new functions and reconstructors.

in one cell ($N_i = N$, $N_j = 0 \; \forall \; j \neq i$). The curves of Figure 4.9 are obtained by keeping the intensities fixed and modifying σ_q in order to obtain different N. It can be seen on Figure 4.9 that the entropy is maximized at $N = 0$, independently of the intensities $\{I_i\}$ of the model, where the optimal value of $\sigma_\mathrm{q} = \infty$ is achieved if the number of quanta per cell is treated as a continuous variable. If the number of quanta per cell were discrete variables, as in Sutton & Wandelt (2006), the choice of a big σ_q would admit only zero values for every cell. Otherwise, if one or more quanta fell in a given cell, the intensity of that cell would diverge as σ_q for arbitrarily large σ_q. This big intensity value, even for one quantum in the cell, would cause a big χ^2 value. Therefore, in our continuous optimization the intensity quantum must be determined a priori.

In the Bayesian description of the entropy we count events that fall in each cell. It seems reasonable to take the noise level as the minimum value of intensity we can distinguish. So, σ_q should approximate the estimated thermal noise in the naturally weighted dirty map. The definition of the weighted dirty map (e.g. Briggs et al., 1999) is

$$I^\mathrm{D}(x,y) \equiv \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} W(u,v)V(u,v)e^{-2\pi i(ux+vy)}dudv, \tag{4.1}$$

$$W(u,v) = \frac{1}{\sum_k w_k}\sum_k w_k\delta(u-u_k, v-v_k), \tag{4.2}$$

where the sums extend over all visibilities, w_k are the weights given to visibility k and δ is the two-dimensional Dirac delta function. Propagating the thermal noise, we get for the standard deviation of the dirty map

$$\sigma_\mathrm{rms}^\mathrm{D} = \sqrt{\frac{\sum_k w_k^2\sigma_k^2}{(\sum_k w_k)^2}}, \tag{4.3}$$

where σ_k are the visibilities standard deviations. To take into account model pixels correlated by the interferometer beam, we should multiply the previous expression by

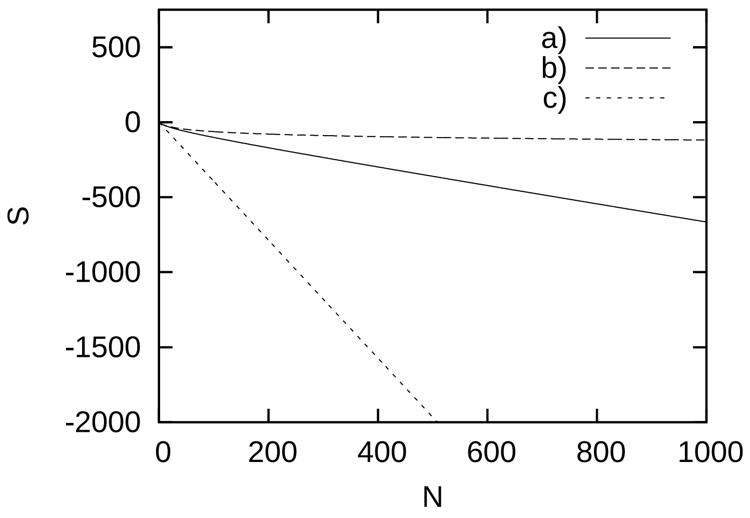

Figure 4.9: Entropy values for different N, $n = 51$ and keeping $\{I_i\}$ fixed. This is achieved by varying σ_{q}. (a) VIR reconstruction intensities. (b) Uniform intensities distribution, $N_i = \frac{N}{n}\ \forall\ i$. (c) Only one cell has all the quanta. $N_i = N$, $N_j = 0\ \forall\ j \neq i$.

$\sqrt{N_{\mathrm{beam}}}$, where N_{beam} is the number of pixels inside a beam pattern. This leads to

$$\sigma_{\mathrm{rms}} = \sqrt{\frac{\sum_k w_k^2 \sigma_k^2}{(\sum_k w_k)^2}} \sqrt{N_{\mathrm{beam}}}. \tag{4.4}$$

For natural weights, $\sigma_k^2 = \frac{1}{w_k}$,

$$\sigma_{\mathrm{rms}} = \sqrt{\frac{N_{\mathrm{beam}}}{\sum_k w_k}} = \sqrt{\frac{N_{\mathrm{beam}}}{\sum_k \frac{1}{\sigma_k^2}}}. \tag{4.5}$$

We calculated the noise with natural weighting, $w_k = \frac{1}{\sigma_k^2}$, because this is the weight we give to each individual visibility data in the optimization of the merit function.

Once we have the value of σ_{q} we search for the optimal number of cells n. In Figure 5.10 we plot the optimal merit function for different n and σ_{q}. These reconstructions were made over a simulation of CBI observations on a mock sky image (Figure 5.1a). We averaged over 100 reconstructions with different realizations of Gaussian noise. The average curves shown in Figure 5.10, start with $n = 10$ and end with $n = 100$ for even n. One single reconstruction for all n took about two hours using an AMD Athlon64 XP3000 processor with 1GB of DDR RAM at 333 MHz, so the 300 reconstructions took about 25 CPU days, but we distributed the work in six computers, so it took about 5 real days in total. It can be seen that for a signal to noise ratio (SNR) of ~ 52, on average, the optimal number of polygons n is between 50 and 55. When σ_{q}

is diminished to $\frac{1}{10}\sigma_{\mathrm{rms}}$, on average, the optimal merit function is found at n close to 30. For $\sigma_{\mathrm{q}} = 10\sigma_{\mathrm{rms}}$, the optimal n is found between 80 and 90. It can be seen that as we increase the value of σ_{q} we reach lower values for our function, as discussed above. Furthermore, the optimal number of polygons increases.

Chapter 5

Results

We used our different algorithms using as input visibilities simulated over a mock image consisting of three Gaussians and a rectangle. To compare between different models, we used residuals, restored images, χ^2 values and direct comparison with our true image.

5.1 Mock Dataset

The mock sky image we used for simulations is a 256×256 image consisting of three Gaussians and a rectangle. Figure 5.1a shows this image on a 128×128 pixel field. Pixels are $0.75' \times 0.75'$, while the CBI's primary beam is of $45'$ FWHM (60 pixels), so most of the emission lies under the beam. We simulated a CBI observation of 3620 visibilities over this image and added Gaussian noise to the visibilities in order to reach a SNR of ~ 52. This SNR was calculated by taking the maximum intensity from the dirty map using natural weights, and using the noise $\sigma_{\rm rms}^{\rm D}$ (see Eq. 4.3). Simulation of the CBI observations is performed with the MOCKCBI program (Pearson 2000, private communication), which calculates the visibilities $V(u,v)$ on the input images $I(x,y)$ with the same uv sampling as a reference visibility dataset (Eq. 2.5). Thus MOCKCBI creates the visibility dataset that would have been obtained had the sky emission followed the true image. Figure 5.1b shows the dirty map calculated over these simulated visibilities using the DIFMAP package (Shepherd, 1997). The CBI's primary beam is drawn as a dashed circle. The secondary side-lobes due to the central discontinuity in u-v coverage can be distinguished in Figure 5.1b at a level comparable to the true emission.

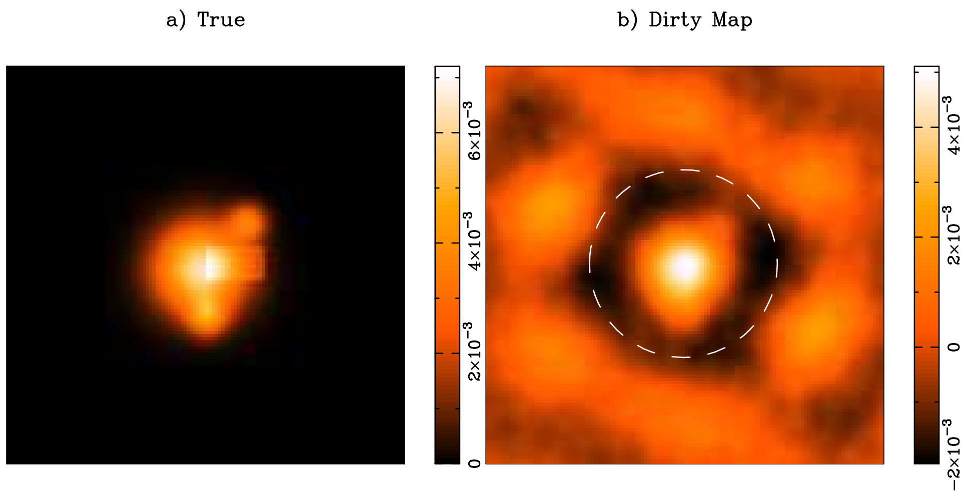

Figure 5.1: Mock dataset for a SNR of ~ 52. (a) The true image. (b) Dirty map.

5.2 Conjugate Gradient Reconstruction

We used the CG method from Press et al. (1992) to try to obtain an optimal model using 51 Voronoi generators [1]. Our optimization parameters were the positions and intensities of our Voronoi polygons ((x_i, y_i) and I_i). The initial conditions used, in this first approach were the positions of the generators distributed randomly over the model field, and the intensities of each polygon starting at a small value of 10^{-5} quanta. Figure 5.2a shows the obtained model inset on a larger 128×128 image. [2].

A great disadvantage of the CG method, is that it usually falls into local minima. In the case of our merit function L, this local minima are highly influenced by the initial distribution of the Voronoi generators. Usually, their positions move very little, although their intensities seem to fit satisfactorily.

5.3 Genetic Algorithm Reconstruction

We used PIKAIA for our reconstruction problem, using as free parameters the positions and intensities of our Voronoi polygons ((x_i, y_i) and I_i). Because PIKAIA maximizes the objective function, instead of using L, we used $L_{\text{GEN}} = \frac{1}{L}$ as our maximization function. We used a population of 100 individuals and stopped after 33 real days. In this time, the program calculated the merit function $\sim 7.9 \cdot 10^5$ times, i.e. $\sim 7.9 \cdot 10^3$

[1]We used 51 Voronoi generators on our reconstructions because this was the optimal number obtained by VIR.

[2]We choose to display the sky images in a larger field than the domain of free parameters; larger fields are required to highlight secondary side-lobes

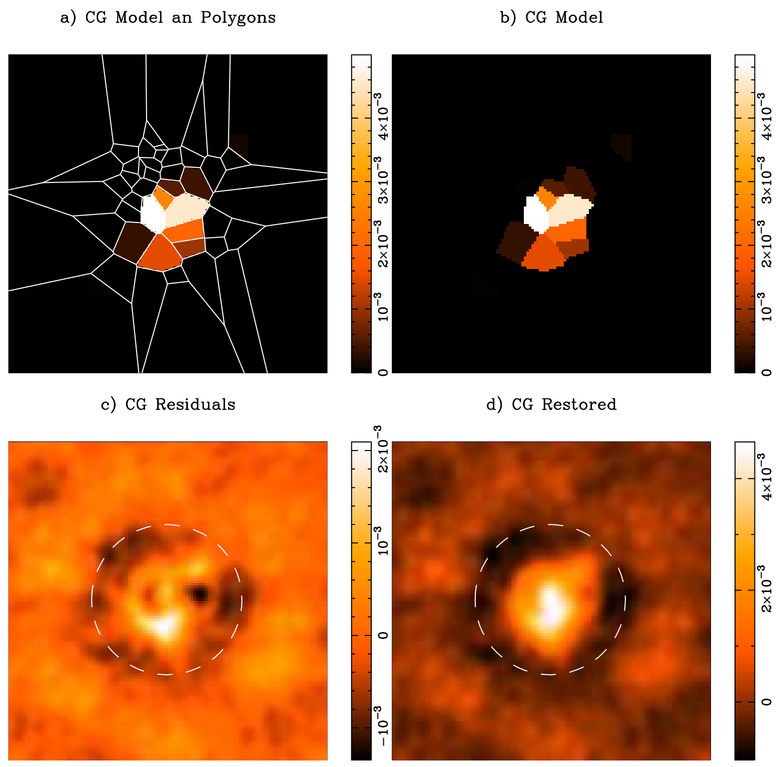

Figure 5.2: Conjugate gradient results. (a) CG reconstruction with its polygons drawn. (b) CG reconstruction. (c) Dirty map of the CG reconstruction residuals. (d) Restored image for the CG model.

generations were breeded. Figure 5.3a shows the best obtained individual (or model) with the Voronoi mesh overlaid inset on a larger 128×128 image.

We also implemented a hybrid method that combines the GA and CG approaches. For this purpose, instead of calculating directly the merit function for each individual, we did a CG optimization and used the resulting local minimum to calculate the merit function. In a 33 day run, the program only did $\sim 3.8 \cdot 10^3$ function calculations (each one including a CG reconstruction), i.e. only 38 generations were breeded. The few generations breeded is caused by the large amount of time the CG takes for each individual. Figure 5.4a shows the best individual obtained with the Voronoi mesh overlaid inset on a larger 128×128 image.

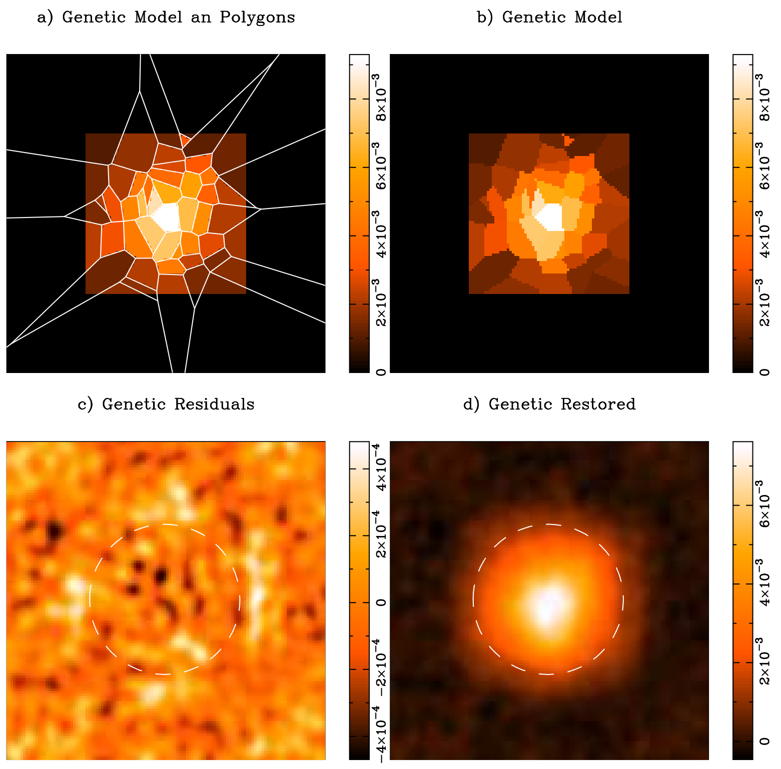

Figure 5.3: Genetic algorithm results. (a) GA reconstruction with its polygons drawn. (b) GA reconstruction. (c) Dirty map of the GA reconstruction residuals. (d) Restored image for the GA model.

5.4 Monte Carlo Reconstructions

5.4.1 Metropolis-Hasting

We implemented MH to generate samples from $P(\{x_i, y_i, I_i\}) = e^{-L(\{x_i,y_i,I_i\})}$. We create a Gaussian proposal density Q centered on the current point of the parameter space, of dispersion 0.0009. In the case of the position of the generators, this dispersion is in units of the total size of the square image. In the case of the intensities, this dispersion is in units of the quantum size. The code is modular enough so that to add new proposal densities, or even new objective distributions, it is only necessary to program each density.

Figure 5.5 shows our MH method. For this reconstruction we used 51 generators. The method took 30 days in which $\sim 4 \cdot 10^5$ samples where generated.

Although MH results did not give satisfactory results, the model could be close to the global optimum. In order to obtain the local optimum closest to the MH reconstruction, we run a CG method starting from the MH model. Figure 5.6 shows such

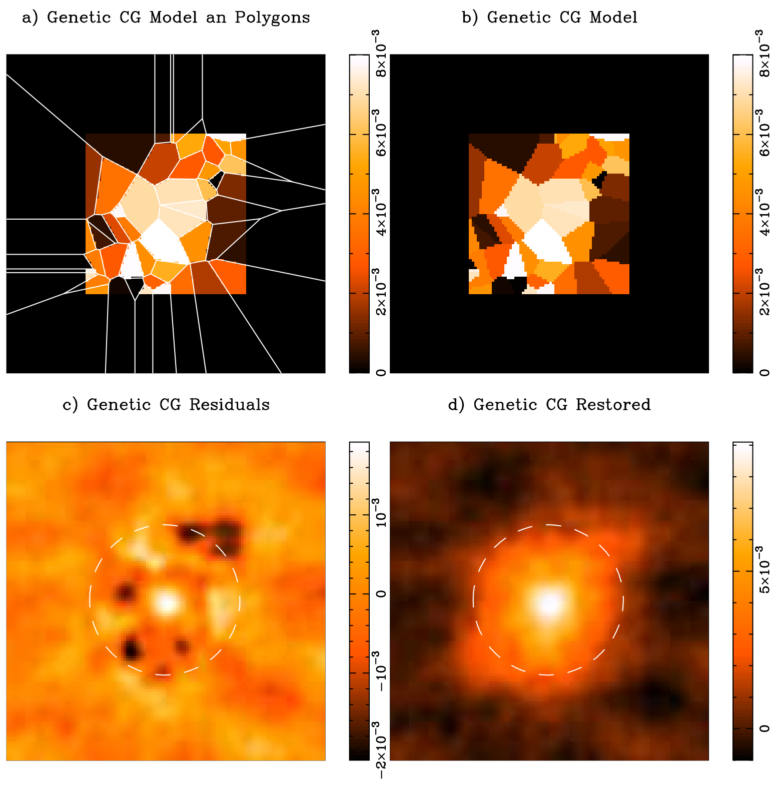

Figure 5.4: Genetic algorithm plus conjugate gradient hybrid method results. (a) GA + CG reconstruction with its polygons drawn. (b) GA + CG reconstruction. (c) Dirty map of the GA + CG reconstruction residuals. (d) Restored image for the GA + CG model.

reconstruction. It can be seen that results are not significantly different to MH.

5.4.2 Gibbs Sampling

To implement GS, we first needed the conditional probabilities of our objective density. We followed the approach of Sutton & Wandelt (2006) for these densities extended to our parameter space:

$$P(z_i^{(t+1)}|\{z_j^{(t+1)}\}_{j=1}^{i-1}, \{z_j^{(t)}\}_{j=i+1}^{N}) \propto \frac{P(z_i^{(t+1)}, \{z_j^{(t)}\}_{j\neq i})}{P(\{z_i^{(t)}\})}, \tag{5.1}$$

$$= e^{L^{(t)}-L^{(t+1)}}, \tag{5.2}$$

where, z_i is a generalized variable describing x_i, y_i and I_i, $L^{(t)} \equiv L(\{z_i^{(t)}\})$ and $L^{(t+1)} \equiv L(z_i^{(t+1)}, \{z_j^{(t)}\}_{j\neq i})$.

To generate samples from the conditional density of Eq. 5.1 we simply used the naive algorithm described on Section 3.3.1. This does not take infinite time because

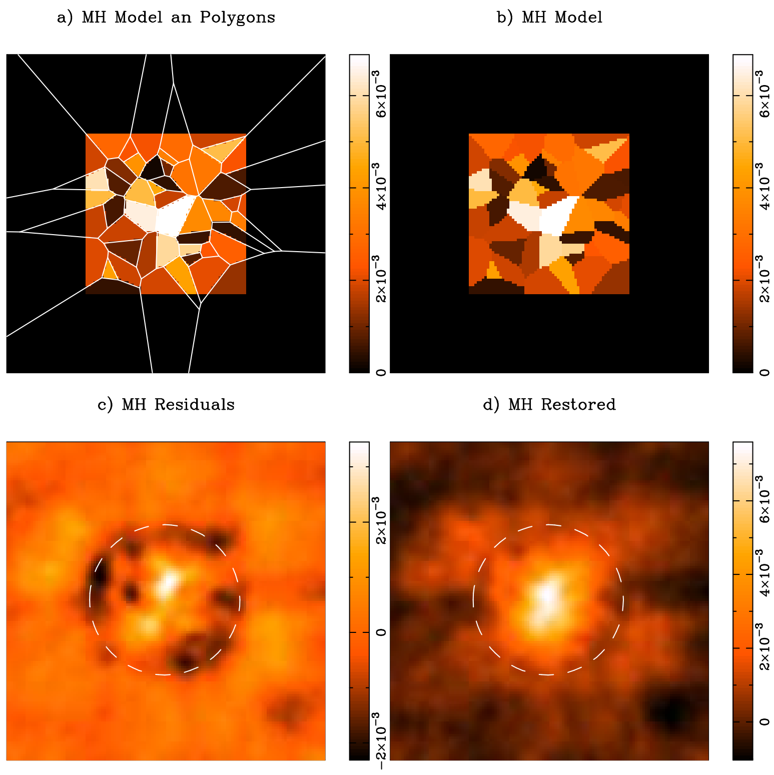

Figure 5.5: Metropolis Hasting algorithm results. (a) MH reconstruction with its polygons drawn. (b) MH reconstruction. (c) Dirty map of the MH reconstruction residuals. (d) Restored image for the MH model.

there is only one search coordinate, so the number of function calculations will be only M on each direction ($N \times M$ in total), where M is the number of divisions of the space, and N the dimension of the parameter space. The calculation of the probability is done by calculating L, but if $L^{(t)} - L^{(t+1)}$ is too big, the computer calculation of its exponential will diverge to ∞. In order to search only in the zone where the parameters make sense and where the function can be calculated, we first had to set bounds to the parameter space. This bounds depend of the current point.

Figure 5.7 shows our GS method. For this reconstruction we used 51 generators and $M = 50$. The method took 22 days in which 180 samples where generated. This number of samples is really small compared to the number of samples generated by MH, but the result reconstruction seems better comparing Figure 5.5 and Figure 5.7. This is caused because of the slow random walk done by MH using the selected proposal density. In the case of GS, such arbitrary parameters are not required.

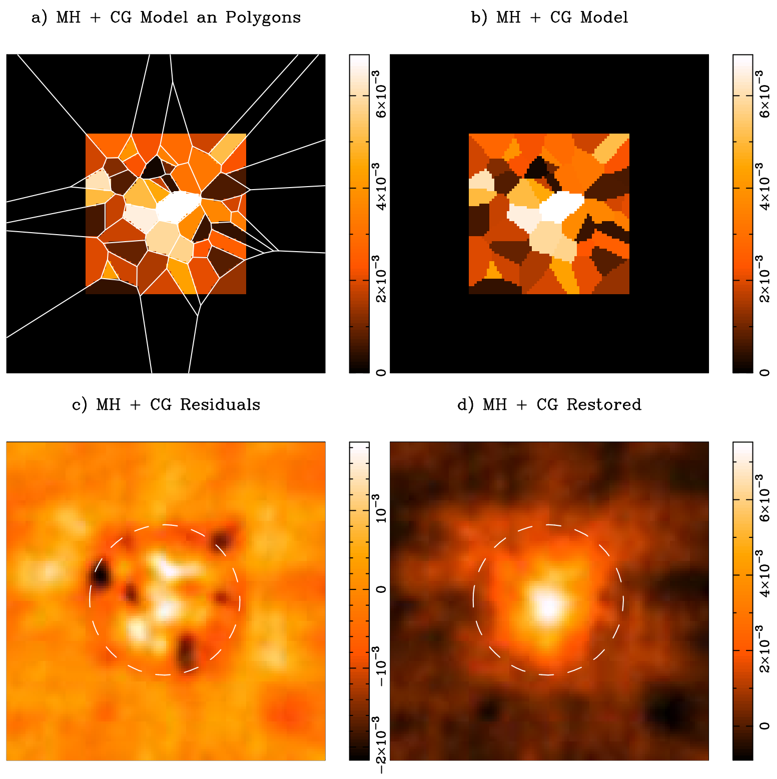

Figure 5.6: Conjugate Gradient applied over a Metropolis Hasting reconstruction. (a) MH + CG reconstruction with its polygons drawn. (b) MH + CG reconstruction. (c) Dirty map of the MH + CG reconstruction residuals. (d) Restored image for the MH + CG model.

5.5 MEM Reconstruction

The VIR method was compared with the MEM algorithm described in Casassus et al. (2006). To fit the model image to the observed visibilities, MEM calculates the model visibilities required by its merit function $L_{\rm MEM}$. The model visibilities are those obtained by a simulation of CBI observations had the sky followed the model image . The free-parameters of our MEM model are the pixels in the model 64×64 image. The model functional we minimize is $L_{\rm MEM} = \chi^2 - \lambda S$, with the entropy $S = -\sum_i I_i \log I_i/M$, where M is a default pixel value well below the noise level, and $\{I_i\}_{i=1}^{N}$ is the model image. We started with the fifth iteration of a pure χ^2 reconstruction ($\lambda = 0$) as initial condition for the CG minimization. This is the same ML initial condition used in our VIR method. Figure 5.8a shows the reconstructed image using $\lambda = \frac{100}{\sigma_{\rm rms}}$ and $M = 10^{-2}\sigma_{\rm rms}$ inset on a larger 128×128 image

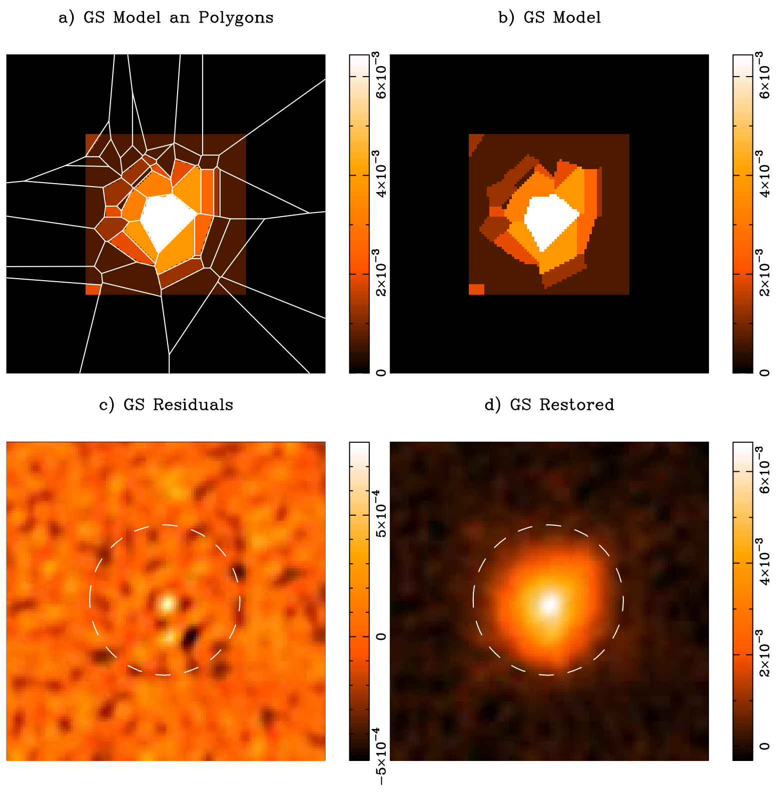

Figure 5.7: Gibbs sampling algorithm results. (a) GS reconstruction with its polygons drawn. (b) GS reconstruction. (c) Dirty map of the GS reconstruction residuals. (d) Restored image for the GS model.

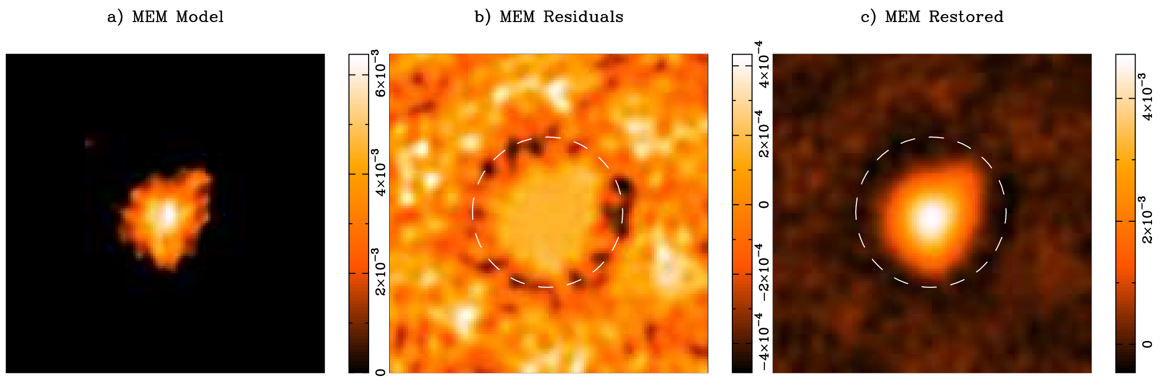

Figure 5.8: MEM results. (a) MEM reconstruction. (b) Dirty map of the MEM reconstruction residuals. (c) Restored image for the MEM model.

5.6 VIR Reconstruction

The MEM algorithm described above requires the prior assignment of the λ and M parameters as well as the entropy formula. In contrast, our VIR algorithm is free from such arbitrary parameters (provided the optimal σ_q is indeed equal to $\sigma_{\rm rms}$). For our VIR method, we only need to find the number of polygons to be used. In order to find the optimal number of polygons we reconstructed with different numbers of generators in a range covering each natural number from $n = 6$ to $n = 100$. We found a minimum at $n = 51$. Figure 5.10 summarizes this search. The whole search for a particular realization of noise took about 10 hours on the AMD Athlon64 XP3000 processor with 1GB of DDR RAM at 333 MHz. The VIR reconstruction using 51 polygons is shown in Figure 5.9a, where the Voronoi cells have also been drawn. Figure 5.9b shows the same model but without drawing the Voronoi mesh.

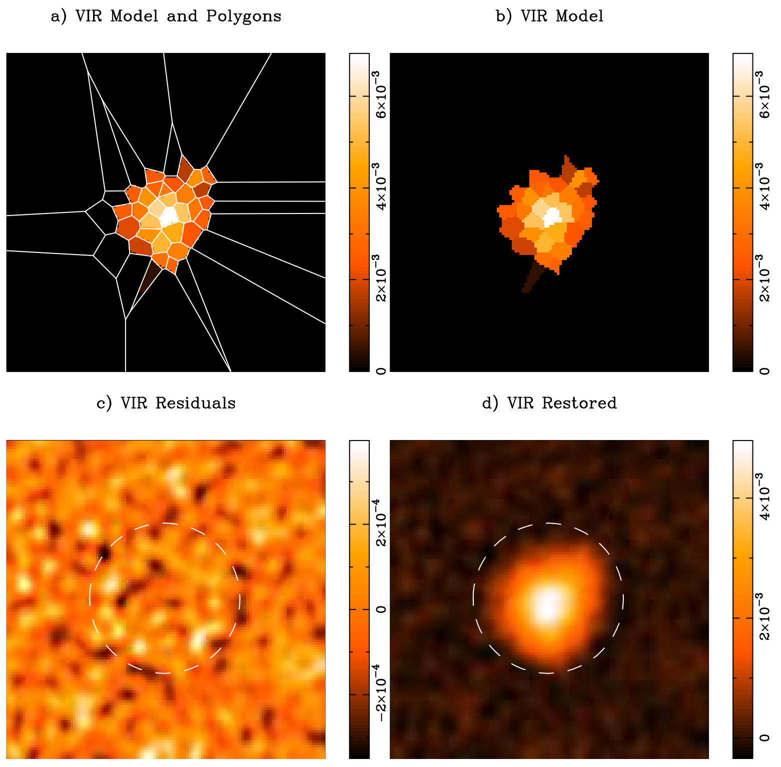

Figure 5.9: VIR results. (a) VIR reconstruction with its polygons drawn. (b) VIR reconstruction. (c) Dirty map of the VIR reconstruction residuals. (d) Restored image for the VIR model.

5.7 Results

The quality of each reconstruction can be assessed by visual inspection, comparing the model images with the true image. The CG, GA + CG, MH and MH + CG models do not look at all like the true map. GA and GS model are similar to the true image, but the shape of the polygons do not seem to fit well the shape of the source. The MEM

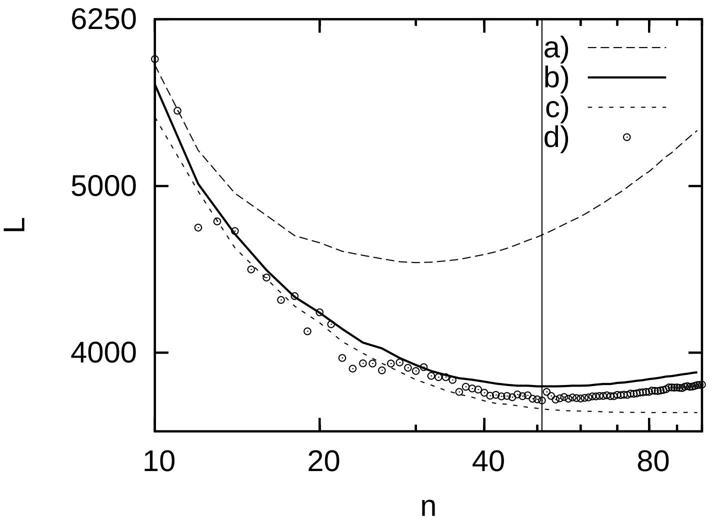

Figure 5.10: The merit function L for different σ_{q} and number of polygons n. The lines are averages taken over 100 different realizations of noise for each n. (a) Reconstructions made using $\sigma_{\mathrm{q}} = \frac{1}{10}\sigma_{\mathrm{rms}}$. (b) Reconstructions made using $\sigma_{\mathrm{q}} = \sigma_{\mathrm{rms}}$. (c) Reconstructions made using $\sigma_{\mathrm{q}} = 10\sigma_{\mathrm{rms}}$. (d) L as a function of n for a practical application of VIR to the simulated visibilities used in the reconstructions of Figure 5.1. In this practical application, the minimum L was found at $n = 51$, and is indicated by a vertical line.

model looks similar to the true image but is noisy. The density of Voronoi generators in the VIR model is greater where there is more emission in the true image, approximating the true image with only a few polygons. We calculated $\chi^2_{\mathrm{im}} = \sum_i (I_i^{\mathrm{mod}} - I_i^{\mathrm{true}})^2$, where I_i^{mod} is the intensity at pixel i of the model image, I_i^{true} is the intensity at pixel i of the true image, and the sum extends over all pixels in the images. χ^2_{im} gives a measure of how well the model fits the true image. It can be seen in Table 5.1 that the VIR reconstruction has the best χ^2_{im}, even better than MEM, showing that the VIR model is closer to the true image than the rest.

Residual visibilities are the subtraction of the model visibilities to the observed visibilities. In the case of a good model, residual visibilities would be only noise. Figures 5.2c, 5.3c, 5.4c, 5.5c, 5.6c, 5.7c, 5.8b and 5.9c show each reconstruction residuals. Residual images are the dirty map of the residuals of the visibilities, calculated over the optimal model visibilities. It can be noted on Figure 5.9c that the VIR residuals are very good, showing only noise. On the other hand, in the MEM residuals (Figure 5.8b) the object shape can clearly be distinguished as well as the CBI's side-lobes. In the GA residuals the object does not seem to appear, but the CBI's side-lobes do. A bright point can be seen in the GS residuals, showing that the GS reconstruction does not model the center of the image in an appropriate way. CG, GA + CG, MH and

	χ^2	$\frac{\chi^2}{n_{\text{data}}}$	L	χ^2_{im}
GA	7534.24	1.041	3895.97	0.015521
GA + CG	14598.60	2.016	7459.07	0.041122
MH	16809.17	2.322	8525.52	0.020575
MH + CG	13755.30	1.900	6998.34	0.020073
GS	7319.17	1.011	3749.32	0.002393
MEM	7354.85	1.016	12192.60	0.001608
VIR	7221.04	0.997	3753.28	0.001396

Table 5.1: Comparison between reconstructions made by different methods.

MH + CG residuals reflect that their respective models does not fit well the data. The object seems to be more compact in the model than in its residuals; as expected these residuals are convolved with the synthetic beam.

Restored images are obtained by convolving the models with a Gaussian point spread function (PSF), given by DIFMAP, and adding the dirty map of the residuals visibilities. Restored images represent the image that would have been obtained in the case of using a single dish telescope. Restored images are shown in Figures 5.2d, 5.3d, 5.4d, 5.5d, 5.6d, 5.7d, 5.8c and 5.9d. On Figures 5.9d and 5.8c it can be assessed that VIR produces improved restored images relative to MEM. The VIR restored image is similar to that expected given the instrumental noise: it approximates the true image convolved with a Gaussian PSF plus a uniform noise level. In the other restored image, on the other hand, the CBI side-lobes can still be distinguished.

The number of optimization parameters in MEM are $64 \times 64 = 4096$, while the other methods have only 51 triplets (cell's (x, y) position and intensity) i.e. 153 free parameters. This smaller number of parameters causes the Bayesian entropy to be greater than the pixelated version, obtaining a smaller value for our merit function L to be minimized. GS and VIR obtained the smallest L values, GS even lower than VIR. We can conclude that, though GS takes a lot more time than VIR, it reaches a better optimum than VIR's conjugate gradient. But for practical use, VIR obtains a better model than GS.

Table 5.1 also shows $\frac{\chi^2}{n_{\text{data}}}$ values, where n_{data} is the number of data points (3620×2 in our case). A good reconstruction should have a $\frac{\chi^2}{n_{\text{data}}}$ value close to 1. It can be seen that the VIR model gives the closest $\frac{\chi^2}{n_{\text{data}}}$ value to 1.

Chapter 6

Conclusions

We have introduced a Bayesian Voronoi image reconstruction (VIR) technique for interferometric data where the image is represented by a Voronoi tessellation in place of the usual pixelated image. VIR consists in adjusting a Voronoi tessellation to an interrupted maximum likelihood pixelated reconstruction and, afterwards, running a conjugate gradient using that tessellation as initial condition. The advantage of Voronoi models is that we can use a smaller number of free parameters, as required by the Bayesian analysis of a discretized intensity field. Our purpose is not optimal CPU efficiency; we search for the optimal image and model from a Bayesian point of view. The free parameters of our model are the Voronoi generators positions (x_i, y_i) and intensities I_i. The following points summarize our work:

- We discretized the intensity field in order to calculate *a priori* probabilities. We defined a quantum intensity value σ_{q} such that $I_i = \sigma_{\mathrm{q}} N_i$, where I_i is the intensity at cell i and N_i the number of quanta in cell i.

- We used conjugate gradient, genetic algorithm and Monte Carlo methods to obtain the optimal model image given the data. We also did a MEM reconstruction to compare with.

- We calculated the analytical derivatives required by the conjugate gradient and cross checked them by finite differences. Because the parameter space in cell generators positions is very structured, the positions of the Voronoi generators are difficult to change. As initial condition we took a Voronoi tessellation adjusted to an interrupted maximum likelihood reconstruction.

- We simulated a CBI observation over a true image and reconstructed sky images from this mock visibility dataset using all our reconstruction methods.

- We defined the value of σ_{q} as the estimated noise of the dirty map and searched for the optimal number of Voronoi polygons for our example dataset.
- We finally compared models, residuals and restored images for all our methods. The VIR model is closer to our true image than the rest. Residuals and restored images are also better in VIR than in the rest. We found that VIR model visibilities give a better fit to the data than the rest, in the sense that χ^2 is closer to its expected value, though Gibbs sampling obtains a better optimum from the merit function point of view.
- We designed our programs using object oriented programming (OOP).
- We developed a tool that allows us to reconstruct using different optimization algorithms and functions. The code is easily extensible. OOP design allows us to add new algorithms and functions without needing to make substantial modifications.

This work has been published in *The Astrophysical Journal*, volume 672, page 1272.

Appendix A

Derivatives

Our merit function for minimization is

$$L = \frac{1}{2}\sum_{k=1}^{N_{\text{Vis}}} \frac{||V_k^{\text{mod}} - V_k^{\text{obs}}||^2}{\sigma_k^2} - \ln\left(\frac{N!}{n^N \prod_{i=1}^{n} N_i!}\right) \tag{A.1}$$

$$= \frac{1}{2}\chi^2 - S. \tag{A.2}$$

So, the derivative of L with respect to any variable x is

$$\frac{\partial L}{\partial x} = \frac{1}{2}\frac{\partial \chi^2}{\partial x} - \frac{\partial S}{\partial x} \tag{A.3}$$

A.1 Calculation of the Derivatives of χ^2

χ^2 derivatives with respect to any variable x can be obtain as follows

$$\begin{aligned}
\frac{\partial}{\partial x}\frac{1}{2}\chi^2 &= \frac{\partial}{\partial x}\left(\frac{1}{2}\sum_{k=1}^{N_{\text{Vis}}}\frac{||V_k^{\text{mod}} - V_k^{\text{obs}}||^2}{\sigma_k^2}\right) \\
&= \frac{1}{2}\sum_{k=1}^{N_{\text{Vis}}}\frac{1}{\sigma_k^2}\frac{\partial}{\partial x}\left((V_k^{\text{mod}} - V_k^{\text{obs}})(V_k^{\text{mod}} - V_k^{\text{obs}})^*\right) \\
&= \frac{1}{2}\sum_{k=1}^{N_{\text{Vis}}}\frac{1}{\sigma_k^2}\frac{\partial}{\partial x}\left(\text{Re}(V_k^{\text{mod}} - V_k^{\text{obs}})^2 + \text{Im}(V_k^{\text{mod}} - V_k^{\text{obs}})^2\right) \\
&= \sum_{k=1}^{N_{\text{Vis}}}\frac{1}{\sigma_k^2}\left(\text{Re}(V_k^{\text{mod}} - V_k^{\text{obs}})\text{Re}\left(\frac{\partial V_k^{\text{mod}}}{\partial x}\right) + \text{Im}(V_k^{\text{mod}} - V_k^{\text{obs}})\text{Im}\left(\frac{\partial V_k^{\text{mod}}}{\partial x}\right)\right),
\end{aligned} \tag{A.4}$$

where its necessary to calculate the model visibilities derivatives with respect to x.

A.1.1 Calculation of $\frac{\partial V_k^{\rm mod}}{\partial I_i}$

In our Voronoi tessellation representation of the sky image

$$V(\vec{k}) = \sum_i^{N_{\rm V}} I_i \int_{\mathcal{V}_i} A(\vec{x}) e^{2\pi i \vec{k}\vec{x}} d\vec{x}, \tag{A.5}$$

where $N_{\rm V}$ is the number of polygons, $\mathcal{V}_i$ is polygon i and I_i its intensity. We neglected the $\sqrt{1-x^2-y^2}$ term which is close to 1, but it can easily be included in $A(\vec{x})$. After derivation and defining $f_k(\vec{x}) \equiv A(\vec{x}) e^{2\pi i \vec{k}_k \vec{x}}$ we obtain

$$\begin{aligned} \frac{\partial V_k^{\rm mod}}{\partial I_i} &= \int\int_{\mathcal{V}_i} f_k(\vec{x}) d^2x, & \text{(A.6)} \\ &= \frac{\sin(\pi u_k \Delta x)\sin(\pi v_k \Delta y)}{\pi^2 u_k v_k} \sum_{\text{pixels } l\epsilon\mathcal{V}_i} A_l e^{2\pi i(u_k x_l + v_k y_l)}, & \text{(A.7)} \\ &\simeq \Delta x \Delta y \sum_{\text{pixels } l\epsilon\mathcal{V}_i} A_l e^{2\pi i(u_k x_l + v_k y_l)} & \text{(A.8)} \end{aligned}$$

for small Δx and Δy.

A.1.2 Calculation of $\frac{\partial V_k^{\rm mod}}{\partial x_i}$ and $\frac{\partial V_k^{\rm mod}}{\partial y_i}$

To evaluate $\frac{\partial V_k}{\partial x_i}$ we move the generator $\vec{x}_i$ an infinitesimal quantity δ_x parallel to the $\hat{x}$ axis as in Figure A.1. We will calculate

$$\frac{\partial V_k}{\partial x_i} = \lim_{\delta_x \to 0} \frac{\Delta V}{\delta_x}, \tag{A.9}$$

where $\Delta V_k = V_k(\vec{x}_1, \cdots, \vec{x}_i + \vec{\delta}_x, \cdots, \vec{x}_{N_V}) - V_k(\vec{x}_1, \cdots, \vec{x}_i, \cdots, \vec{x}_{N_V})$.

It can be seen in Figure A.1 that when moving the generator $\vec{x}_i$, the only polygons modified are $\mathcal{V}_i$ and its neighbors. Using this, Eq. A.5 leads to

$$\begin{aligned} \Delta V_k &= I_i' \int_{\mathcal{V}_i'} f_k(\vec{x}) d\vec{x} - I_i \int_{\mathcal{V}_i} f_k(\vec{x}) d\vec{x} \\ &\quad + \sum_{j \in J_i} \Big(I_j' \int_{\mathcal{V}_j'} f_k(\vec{x}) d\vec{x} - I_j \int_{\mathcal{V}_j} f_k(\vec{x}) d\vec{x} \Big), & \text{(A.10)} \end{aligned}$$

where $\mathcal{V}_i$ is the polygon generated by $\vec{x}_i$ before moving, $\mathcal{V}_i'$ is the same polygon after moving $\vec{x}_i$, J_i is the set of indices of the polygons that are neighbors to $\mathcal{V}_i$ and J_i' is the set of indices of the polygons that are neighbors to $\mathcal{V}_i'$.

It can be seen in Figure A.1 that

$$\mathcal{V}_i = (\mathcal{V}_i \cap \mathcal{V}_i') \cup (\mathcal{V}_i \setminus \mathcal{V}_i \cap \mathcal{V}_i'), \quad \mathcal{V}_i' = (\mathcal{V}_i \cap \mathcal{V}_i') \cup (\mathcal{V}_i' \setminus \mathcal{V}_i \cap \mathcal{V}_i'), \tag{A.11}$$

$$\mathcal{V}_j = (\mathcal{V}_j \cap \mathcal{V}_j') \cup (\mathcal{V}_i' \cap \mathcal{V}_j), \qquad \mathcal{V}_j' = (\mathcal{V}_j \cap \mathcal{V}_j') \cup (\mathcal{V}_i \cap \mathcal{V}_j'), \tag{A.12}$$

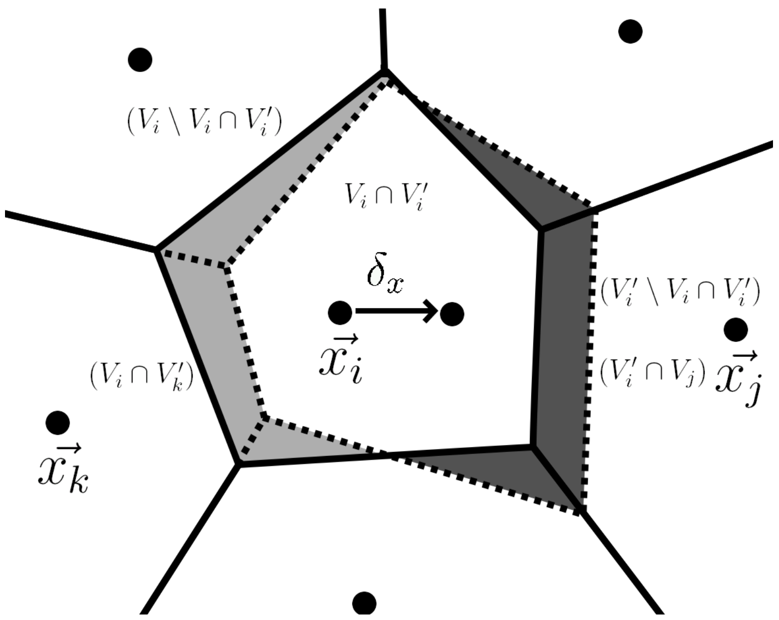

Figure A.1: Voronoi tessellation before and after translating the site $\vec{x_i}$ by $\vec{\delta_x}$. Voronoi generators are represented with dots. The solid lines are the polygons before moving $\vec{x_i}$. The dotted lines represent the new polygons after varying $\vec{x_i}$.

so, Eq. A.10 is

$$\begin{aligned} \Delta V_k &= (I_i' - I_i) \int_{\mathcal{V}_i' \cap \mathcal{V}_i} f_k(\vec{x}) d\vec{x} \\ &\quad + \sum_{j \in J_i} \Big[(I_i' - I_j) \int_{\mathcal{V}_i' \cap \mathcal{V}_j} f_k(\vec{x}) d\vec{x} + (I_j' - I_i) \int_{\mathcal{V}_i \cap \mathcal{V}_j'} f_k(\vec{x}) d\vec{x} \Big]. \end{aligned} \tag{A.13}$$

In our case the cells' intensities don't depend of the position of the generators, so we obtain

$$\Delta V_k = \sum_{j \in J_i} \Big[(I_i - I_j) \Big(\int_{\mathcal{V}_i' \cap \mathcal{V}_j} f_k(\vec{x}) d\vec{x} - \int_{\mathcal{V}_i \cap \mathcal{V}_j'} f_k(\vec{x}) d\vec{x} \Big) \Big]. \tag{A.14}$$

It can be seen in Figure A.1 that to obtain ΔV_k we must integrate only over the shaded regions. For this purpose, for each region between $\vec{x}_i$ and $\vec{x}_j$ we will define a coordinate system

$$\hat{s} = -\cos\alpha_j \hat{x} + \sin\alpha_j \hat{y}. \quad \hat{t} = \sin\alpha_j \hat{x} + \cos\alpha_j \hat{y}, \tag{A.15}$$

where α_j is the angle formed by the $-\hat{x}$ axis and the edge a_{ij} between $\vec{x}_i$ and $\vec{x}_j$ (see Figure A.2). Using this change of coordinates, the integral over the region of interest is

$$\int_{\mathcal{V}_i' \cap \mathcal{V}_j} f_k(x, y) dx dy = \int_{\mathcal{V}_i' \cap \mathcal{V}_j} f_k(s, t) ds dt. \tag{A.16}$$

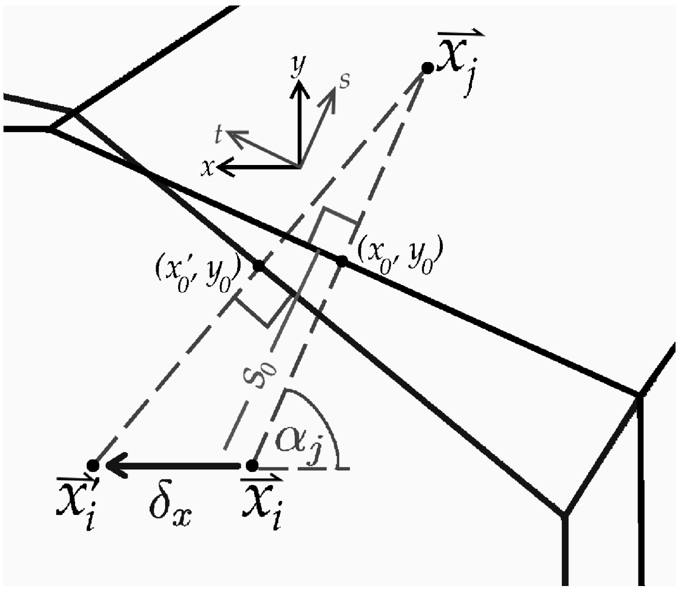

Figure A.2: Change of coordinates from (x, y) to (s, t).

Let $\vec{x_i} = (x_i, y_i)$ be the position of the i cell's generator, $\vec{x_j} = (x_j, y_j)$ one of its neighbor, and $\vec{x_i}' = (x_i + \delta_x, y_i)$ the site's position after moving it a quantity δ_x. We define $\vec{x_0} \equiv (x_0, y_0)$ as the point in the intersection of the segment formed by $\vec{x_i}$ and $\vec{x_j}$ and its respective edge a_{ij}. The same way, we define $\vec{x_0}' = (x_0', y_0')$ as the point in the intersection of the segment formed by $\vec{x_i}'$ and $\vec{x_j}$ and its respective edge a_{ij}'. It can be seen on Figure A.2 that $x_0 = \frac{x_i + x_j}{2}$, $x_0' = x_0 + \frac{\delta}{2}$ and $y_0' = y_0 = \frac{y_i + y_j}{2}$.

The edge a_{ij} is defined in the new coordinate system by

$$s = s_0 = -x_0 \cos \alpha_j + y_0 \sin \alpha_j. \tag{A.17}$$

In the same way, the edge a_{ij}' is defined in the original coordinate system by

$$y = m(x - x_0') + y_0, \tag{A.18}$$

where

$$m \equiv \frac{x_i + \delta_x - x_j}{y_j - y_i}. \tag{A.19}$$

We can define the same line in our new coordinate system as

$$s = m't + b', \tag{A.20}$$

where

$$m' \equiv -\frac{\cos \alpha_j + m \sin \alpha_j}{\sin \alpha_j - m \cos \alpha_j}, \tag{A.21}$$

$$b' \equiv \frac{-m x_0' + y_0}{\sin \alpha_j - m \cos \alpha_j}. \tag{A.22}$$

This can be approximated to first order in δ_x as

$$m' \simeq \delta_x M_x, \tag{A.23}$$
$$b' \simeq s_0 + \delta_x B_x, \tag{A.24}$$

where

$$M_x \equiv \frac{\sin^2\alpha_j}{y_j - y_i} = \frac{\sin\alpha_j \cos\alpha_j}{x_i - x_j}, \tag{A.25}$$
$$B_x \equiv \frac{\sin\alpha_j}{y_j - y_i}(s_0 \cos\alpha_j + x_i) = \frac{\cos\alpha_j}{x_i - x_j}(s_0\cos\alpha_j + x_i). \tag{A.26}$$

The integral in Eq. A.14 using our new coordinate system will be

$$\mathcal{I} = \int_{\mathcal{V}_i' \cap \mathcal{V}_j} f_k(\vec{x}) d\vec{x} - \int_{\mathcal{V}_i \cap \mathcal{V}_j'} f_k(\vec{x}) d\vec{x} \tag{A.27}$$
$$= \int \int_{a_{ij}}^{a'_{ij}} A(\vec{x}) e^{2\pi i(ux+vy)} dxdy. \tag{A.28}$$

If we use $A(\vec{x})$ in the (s,t) coordinate system as a pixelated image, Eq. A.28 will be

$$\mathcal{I} = \sum_{l \,\epsilon\, \text{pixeles de } a_i} A_l \int_{t^1_{ijl}}^{t^2_{ijl}} \int_{s_0}^{m't+b'} e^{2\pi i(ux(s,t)+vy(s,t))} dsdt, \tag{A.29}$$

where t^1_{ijl} and t^2_{ijl} are the t coordinate of the beginning and end of the portion of the edge a_{ij} that intersects pixel l. Developing the previous expression,

$$\mathcal{I} = \sum_l A_l \int_{t^1_{ijl}}^{t^2_{ijl}} \int_{s_0}^{m't+b'} e^{2\pi i(u(-s\cos\alpha_j + t\sin\alpha_j) + v(s\sin\alpha_j + t\cos\alpha_j))} dsdt, \tag{A.30}$$
$$\simeq \sum_l \frac{A_l}{\pi c_2} e^{2\pi i(s_0 c_1 + \bar{t}_{ijl} c_2)} \kappa_{ijl} \delta_x, \tag{A.31}$$

where we defined

$$c_1 \equiv -u\cos\alpha_j + v\sin\alpha_j, \tag{A.32}$$
$$c_2 \equiv u\sin\alpha_j + v\cos\alpha_j, \tag{A.33}$$
$$\kappa_{ijl} \equiv (M_x \bar{t}_{ijl} + B_x)\sin(\pi c_2 \Delta t_{ijl}) + i\frac{M_x}{2}\Big(\frac{\sin(\pi c_2 \Delta t_{ijl})}{\pi c_2} - \Delta t_{ijl}\cos(\pi c_2 \Delta t_{ijl})\Big), \tag{A.34}$$
$$\bar{t}_{ijl} \equiv \frac{t^1_{ijl} + t^2_{ijl}}{2}, \tag{A.35}$$
$$\Delta t_{ijl} \equiv \frac{t^2_{ijl} - t^1_{ijl}}{2}. \tag{A.36}$$

In the calculation above we integrated over the fraction of the edge that falls inside pixel l and then summed these integrals over the whole edge a_i. It is also possible to approximate the integral of Eq. A.30 as $\int_{t_{ijl}^1}^{t_{ijl}^2} g(t)dt = g(\bar{t}_{ijl})\Delta t_{ijl}$, which is equivalent to taking the limit over the integral $\mathcal{I}$ of Eq. A.31, $\lim_{\Delta t_{ijl}\to 0}\mathcal{I}$, obtaining

$$\mathcal{I} = \sum_l A_l \Delta t_{ijl}(M_x \bar{t}_{ijl} + B_x) e^{2\pi i(\bar{t}_{ijl}c_2 + s_0 c_1)}\delta_x. \tag{A.37}$$

We found by direct evaluation that the difference between Eq. A.37 and Eq. A.31 is negligible, so, for simplicity, we will use Eq. A.37. Introducing Eq. A.37 in Eq. A.14, we obtain

$$\Delta V_k = \delta_x \sum_{j\in J_i}\Big[(I_i - I_j)\sum_l A_l \Delta t_{ijl}(M_x \bar{t}_{ijl} + B_x) e^{2\pi i(\bar{t}_{ijl}c_2 + s_0 c_1)}\Big], \tag{A.38}$$

so, according to Eq. A.9, the derivative of the k visibility with respect to the position x of polygon i is

$$\begin{aligned}\frac{\partial V_k}{\partial x_i} &= \lim_{\delta_x\to 0}\frac{\Delta V}{\delta_x}, &\text{(A.39)}\\ &= \sum_{j\in J_i}\Big[(I_i - I_j)\sum_l A_l \Delta t_{ijl}(M_x \bar{t}_{ijl} + B_x) e^{2\pi i(\bar{t}_{ijl}c_2 + s_0 c_1)}\Big]. &\text{(A.40)}\end{aligned}$$

Similarly, for the derivative with respect to the position y of the i polygon we obtain

$$\frac{\partial V_k}{\partial y_i} = \sum_{j\in J_i}\Big[(I_i - I_j)\sum_l A_l \Delta t_{ijl}(M_y \bar{t}_{ijl} + B_y) e^{2\pi i(\bar{t}_{ijl}c_2 + s_0 c_1)}\Big], \tag{A.41}$$

where

$$\begin{aligned}M_y &\equiv \frac{\cos^2\alpha_j}{x_i - x_j} = \frac{\sin\alpha_j\cos\alpha_j}{y_j - y_i}, &\text{(A.42)}\\ B_y &\equiv \frac{\sin\alpha_j}{y_j - y_i}(s_0\sin\alpha_j - y_i) = \frac{\cos\alpha_j}{x_i - x_j}(s_0\sin\alpha_j - y_i). &\text{(A.43)}\end{aligned}$$

A.2 Calculation of the Derivatives of S

We defined our entropy as

$$\begin{aligned}S &= \ln\left(\frac{N!}{n^N\prod_{i=1}^n N_i!}\right) &\text{(A.44)}\\ &= \ln(N!) - N\ln(n) - \sum_{i=1}^n \ln(N_i!) &\text{(A.45)}\\ &= \ln\Big(\Gamma(N+1)\Big) - N\ln(n) - \sum_{i=1}^n \ln\Big(\Gamma(N_i+1)\Big), &\text{(A.46)}\end{aligned}$$

where $N_i = \frac{I_i}{\sigma_{\rm q}}$ is the number of quanta in cell i, $N = \sum_i N_i$ and Γ is the Gamma function. It can be seen that this function does not depend on the position of the Voronoi generators, so

$$\frac{\partial S}{\partial x_i} = \frac{\partial S}{\partial y_i} = 0. \tag{A.47}$$

Using Weierstrass' definition of the Gamma function

$$\Gamma(z) = z^{-1} e^{-\gamma z} \prod_{n=1}^{\infty} \left[\left(1 + \frac{z}{n}\right)^{-1} e^{z/n} \right],$$

where γ is Euler's constant, we can obtain

$$\frac{\partial \ln \Big(\Gamma(z+1)\Big)}{\partial z} = -\gamma + \sum_{n=1}^{z} \frac{1}{n} \tag{A.48}$$

so, the derivative of S with respect to I_i is

$$\frac{\partial S}{\partial I_i} = \frac{1}{\sigma_{\rm q}} (\sum_{k=1}^{N} \frac{1}{k} - \ln n - \sum_{k=1}^{N_i} \frac{1}{k}). \tag{A.49}$$

A.3 Finite Difference Cross Check on the Derivatives

Numerical calculation of the derivatives by finite differences is not very accurate, in particular for the position of the generators. Finite difference derivatives are calculated as $\frac{\partial L}{\partial x} = \frac{L(x+\delta) - L(x)}{\delta}$, where δ is a small displacement of x. In the case of the positions of the generators, if δ is too small, the pixelization of the Voronoi diagram (needed to obtain the model visibilities) will not change after the displacement δ. On the other hand, if δ is too big, the generator displacement may cause the function to change abruptly, as explained below. That is why we calculated the analytical expression for the derivatives.

To verify that our derivatives are correctly calculated and programmed, we compared our analytical result with a numerical calculation. We created a Voronoi tessellation of 50 polygons with random positions and intensities and calculated the analytical and numerical derivatives using these parameters $\{x_i, y_i, I_i\}$. For the case of $\frac{\partial L}{\partial x_i}$ and $\frac{\partial L}{\partial y_i}$ this numerical cross check consists of moving each Voronoi generator a quantity δ from -0.1 to 0.1 with an interval of 10^{-3} in units of the total size of the square image. We evaluate the merit function L at each position intervals, thus obtaining two sequences $\{L_i\}_{i=1}^{2\times10^2}$. We then fitted a polynomial of order four to the curve defined by each sequence $\{L_i\}$ and calculated the derivative of the polynomial at $\delta = 0$. For the case

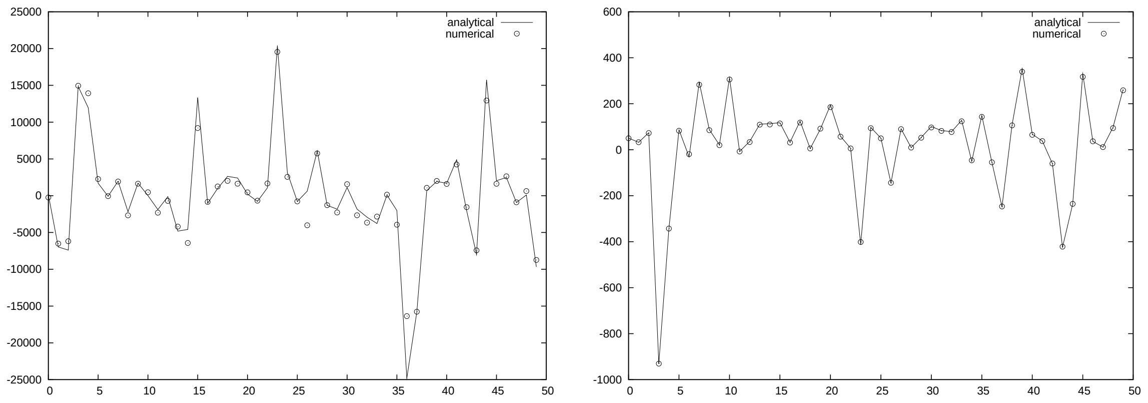

Figure A.3: Verification of the derivatives. The solid line shows analytical derivatives, and dots show numerical approximations. *Left:* $\frac{\partial L}{\partial x_i}$. *Right:* $\frac{\partial L}{\partial I_i}$. The polygon identifier i is indicated on the x-axis.

of $\frac{\partial L}{\partial I_i}$ we varied the intensity of cell i from $-\sigma_\mathrm{q}$ to σ_q and did the same approximation to a polynomial of order four and calculated its derivative. Figure A.3 shows this cross check for $\frac{\partial L}{\partial x_i}$ and $\frac{\partial L}{\partial I_i}$. Although the derivatives are similar, they are not exactly the same for $\frac{\partial L}{\partial x_i}$. This is caused by the polynomial coarseness fit, as explained below.

Figure A.4 shows the curve fit for $\frac{\partial L}{\partial x_i}$ for three different generators (generator number 37, 36 and 18 respectively). It can be seen in Figure A.4 that the polynomial fit adjusts quite well to the function values for polygon number 37, so on Figure A.3 both derivatives are the same. On the contrary, for polygons number 36 and 18, the fitted polynomial does not resemble the function L at $\delta = 0$, causing a slight difference in their derivatives on Figure A.3. For polygon number 18 the polynomial does not fit the curve at all. This is the main problem of using a numerical approximation for the derivatives of $\{\vec{x_i}\}$: when two polygons are closer than δ, the generator displacement causes the function L to change abruptly (see Figure A.5).

It can be seen that the analytical and numerical derivatives on Figure A.3 are almost the same. As explained above, differences are produced because there are cases where the polynomials do not fit well to the variations of the merit function L (for example, when two generators are too close). In an accurate calculation it is necessary to use the analytical derivatives.

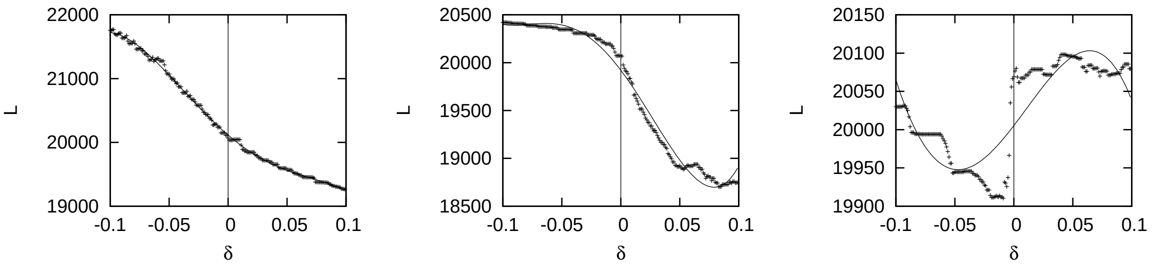

Figure A.4: Examples of polynomial fits, used to determine numerical derivatives of the optimization function L for a particular generator. Dots represent L vs the polygon displacement δ in x and the solid line shows the fourth order polynomial fit to L. A vertical line is drawn at $\delta = 0$, where the derivatives were calculated. *Left:* Generator number 37, with a satisfactory polynomial fit. *Center:* Generator number 36, the curve is not a good fit at $\delta = 0$. *Right:* 18th generator, the curve is not a good fit because L shows an abrupt variation near $\delta = 0$, which is due to the proximity of another generator.

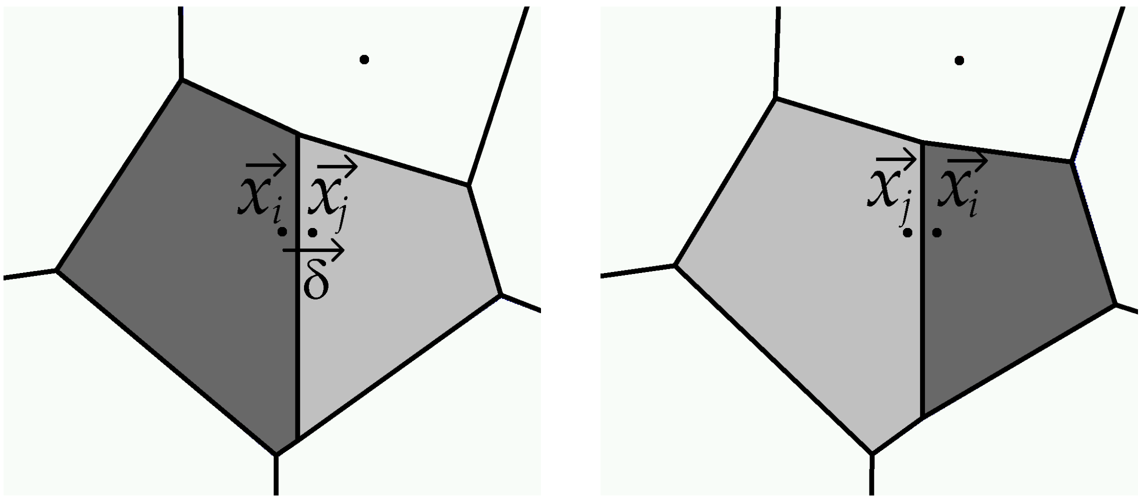

Figure A.5: Translation of a generator close to another. *Left:* Before moving generator $\vec{x_i}$, polygon i, the darker polygon in the image, is on the left. *Right:* After moving generator $\vec{x_i}$, by a displacement of δ, polygon i is on the right of polygon j. When displacing generator $\vec{x_i}$ the diagram changes considerably, with a concomitant abrupt variation in L.

Bibliography

Briggs D. S., Schwab F. R., Sramek R. A., in Taylor G. B., Carilli C. L., Perley R. A., eds, Synthesis Imaging in Radio Astronomy II, *ASP Conf. Ser.*, 1999, n° 180, Astron. Soc. Pac., San Francisco, p. 127 - 150

Cabrera Vives, Guillermo Felipe. Síntesis de Imágenes Interferométricas Basada en Diagramas de Voronoi. Memoria (Ingeniero Civil en Computación). Santiago, Chile. Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 2005, 63 pages.

Casassus, S., Cabrera, G. F., Förster, F, Pearson, T. J., Readhead, A. C. S., Dickinson, C., Morphological Analysis of the Centimeter-Wave Continuum in the Dark Cloud LDN 1622, *The Astrophysical Journal*, 2006, n° 639, p. 951 - 964

Cornwell, T. J. & Evans, K. F., A Simple Maximum Entropy Deconvolution Algorithm, *Astronomy & Astrophysics*, 1985, n° 143, p. 77 - 83

High Altitude Observatory, PIKAIA [on line] <http://www.hao.ucar.edu/Public/models/pikaia/pikaia.html> [Consulted January 20, 2008].

Högbom, J. A., Aperture Synthesis with a Non-Regular Distribution of Interferometer Baselines, *Astronomy & Astrophysics Supplement Series*, 1974, n° 15, p. 417 - 426

MacKay, D. J. C., Information Theory, Inference, and Learning Algorithms, C. Cambridge University Press, 2004

Narayan, Ramesh & Nityananda, Rajaram, Maximum Entropy Image Restoration in Astronomy, *Ann, Rev, Astronomy & Astrophysics*, 1986, n° 24, p. 127 - 170

Okabe, A., Boots, B. & Sugihara, K., Spacial Tessellations Concepts and Applications of Voronoi Diagrams, John Wiley & Sons, 1992

Padin, S., et al, The Cosmic Background Imager, *Publications of the Astronomical Society of the Pacific*, 2002, n° 114, p. 83 - 97

Piña, R. K. & Puetter, R. C., Bayesian Image Reconstruction: The Pixon and Optimal Image Modeling, *Publications of the Astronomical Society of the Pacific*, 1993, n° 105, p. 630 - 637

Press, W. H., Flannery, B. P., Teukilsky, S. A., Vettering, W. Y., Numerical Recipes in C, C. Cambridge University Press, 1992

Shepherd, M.C., in Hunt, G & Payne H. E. eds, Astronomical Data Analysis Software and Systems VI, *ASP Conf. Ser.*, 1997, n° 125, Astron. Soc. Pac., San Francisco, p. 77 - 84

Sutton, E. C. & Wandelt, B. D., Optimal Image Reconstruction in Radio Interferometry, *The Astrophysical Journal Supplement Series*, 2006, n° 162, p. 401 - 416