Universidad Internacional de la Rioja (UNIR)

Escuela Superior de Ingeniería y Tecnología

Máster en Inteligencia Artificial

Implementación de algoritmos de deconvolución en el software de reducción de datos astrofísicos Gnuastro

Trabajo Fin de Estudios

presentado por: Álvaro Morales Márquez

Dirigido por: Roberto Baena Gallé

Ciudad:Alcalá de Guadaíra

Fecha: 11 de Septiembre de 2024

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Resumen

La deconvolución es una operación matemática que permite recuperar imágenes que han sido distorsionadas por la respuesta óptica del instrumento de observación en presencia de ruido. En este trabajo, se han implementado diferentes algoritmos de deconvolución para el tratamiento de imágenes astronómicas. El objetivo final es compartir dichos algoritmos de manera libre a través de su incorporación en la librería open source Gnuastro. Concretamente, se han implementado los algoritmos de inversión directa, Wiener-Tikhonov, Richardson-Lucy y AWMLE. Además, se han desarrollado librerías de bajo nivel que permiten trabajar con matrices de números complejos y realizar transformaciones de Fourier y wavelet, las cuales también pueden ser aprovechadas en el futuro por cualquier desarrollador o investigador. Para validar los algoritmos se han utilizado tanto imágenes académicas con ruido acotado como imágenes astrofísicas más realistas y complejas que simulan diferentes escenarios astrofísicos. Por último, para asegurar la calidad del código y su correcta integración en Gnuastro, el trabajo ha sido supervisado por un administrador principal del proyecto de manera continua.

Palabras Clave: Gnuastro, Deconvolución, Astrofísica, AWMLE, Richardson-Lucy.

Abstract

Deconvolution is a mathematical operation to recover images that have been distorted by the optical response of the observation instrument in the presence of noise.. In this work, different deconvolution algorithms have been implemented for the treatment of astronomical images. The final objective is to share these algorithms freely through their incorporation in the open source library Gnuastro. Specifically, the direct inversion, Wiener-Tikhonov, Richardson-Lucy and AWMLE algorithms have been implemented. In addition, low-level libraries have been developed to work with complex number matrices and to perform Fourier and wavelet transforms, which can also be used in the future by any developer or researcher. Both academic images with bounded noise and more realistic and complex astrophysical images simulating different astrophysical scenarios have been used to validate the algorithms. Finally, to ensure the quality of the code and its correct integration into Gnuastro, the work has been supervised by a senior project manager on a continuous basis.

Palabras Clave: Gnuastro, Deconvolution, Astrophysics, AWMLE, Richardson-Lucy.

1. Introducción

1.1. Motivación

El software de código abierto (*open source* en inglés) presenta la posibilidad de distribuir de manera gratuita y accesible bibliotecas y funcionalidades a toda la comunidad. Al no solo ofrecer el programa, sino todo el código completo, cualquier persona puede evaluar la funcionalidad del programa, reescribirlas si fuera necesario para su aplicación e incluso contribuir en la mejora de este corrigiendo errores o añadiendo nuevas funcionalidades. Esto también se traduce en no depender de una compañía propietaria del software para modificaciones o actualizaciones (Corrado, 2005).

En este trabajo, el objetivo es implementar nuevas funcionalidades para el proyecto *open source* Gnuastro, contribuyendo así a ampliar su utilidad y potencial en el campo de la astrofísica. Gnuastro es un paquete oficial de GNU (Free Software Foundation, 2024) escrito en C con el propósito de facilitar el desarrollo de aplicaciones de astrofísica. Estas librerías ofrecen desde el procesamiento de ficheros FITS (Pence, Chiappetti, y Shaw, 2010) hasta detectar objetos en una imagen (Akhlaghi, 2018).

Un archivo FITS (*Flexible Image Transport System*) es un formato estándar para almacenar imágenes y datos científicos, comúnmente utilizado en astronomía. Cada archivo FITS se organiza en unidades llamadas HDU (*Header Data Unit*), que contienen dos partes: un encabezado con los metadatos de la imagen o datos (como dimensiones, tipo de datos, y coordenadas) y una matriz de datos que puede representar una imagen, tabla o serie de datos. Un archivo FITS puede incluir múltiples HDUs, comenzando con un HDU primario y seguido opcionalmente por otros de extensión, cada uno con su propia información y datos asociados.

Las nuevas funcionalidades que se agregarán están relacionadas con las técnicas de deconvolución, por ahora no disponibles en Gnuastro. Estas técnicas son fundamentales en la corrección de la distorsión producida por los instrumentos ópticos y permiten obtener imágenes más nítidas y precisas. La deconvolución es uno de los pilares fundamentales de la visión artificial, siendo esencial para mejorar la calidad y la interpretación de las imágenes astronómicas.

1.2. Planteamiento del trabajo

La primera parte del trabajo consiste en analizar algoritmos de deconvolución para aplicaciones de astrofísica y seleccionar los que mejor se adapten a las necesidades del proyecto. Al mismo tiempo, es importante analizar la estructura de Gnuastro y cómo está organizado. Esto se debe a que el código tiene que encajar en el proyecto como una pieza más, no solo en funcionalidad si no en estructura y estilo de escritura (reglas de codificación o *coding rules*).

Con el análisis previo terminado, se puede pasar a una primera ejecución de los algoritmos. Estos algoritmos deben estar escritos en el lenguaje de programación C para garantizar la compatibilidad con Gnuastro.

1.3. Estructura del trabajo

En el Capítulo 2 se desarrolla el contexto y el estado del arte. En el Capítulo 3 se describen los objetivos. El Capítulo 4 se dedica a identificar los requisitos, tanto para ser compatible con Gnuastro como para los objetivos propuestos en este trabajo. En el Capítulo 5 se desarrolla y describe toda la contribución realizada en este trabajo. El Capítulo 6 se reserva para evaluar los resultados obtenidos. Por último, el Capítulo 7 contiene las conclusiones y las proyecciones futuras del trabajo.

2. Contexto y Estado del Arte

2.1. Herramientas matemáticas

2.1.1. Convolución

La convolución es una operación matemática que toma dos funciones y produce una nueva función que describe cómo una de las funciones modifica o filtra a la otra a través de un proceso de integración ponderada. Es una herramienta muy utilizada en diversas áreas de la ingeniería. En el caso de este trabajo se utilizará con imágenes. La convolución en imágenes implica que ambas funciones sean bidimensionales y discretas.

Dada una imagen I con dimensiones m x n y una imagen P con dimensiones 2N+1 x 2N+1, la nueva matrix $O = I * P$ se define como:

$$\mathbf{O[i,j]} = \sum_{\mathbf{r=0}}^{\mathbf{2N+1}} \sum_{\mathbf{s=0}}^{\mathbf{2N+1}} \mathbf{I[i-N+r-1, j-N+s-1]P[r,s]} \tag{2.1}$$

La nueva imagen O, solo está definida en i = [N+1,m-n-1] y j = [N+1,n-N-1], dependiendo de la aplicación se pueden aplicar distintas técnicas: reducir cada dimensión en N píxeles, rellenar la función I con ceros o aplicar formulas diferentes en los bordes de la imagen (Palomares, Monsoriu, y Alemany, 2016). La imagen P suele conocerse como kernel. El kernel es un filtro que puede ejercer diferentes funciones como enfocar, desenfocar, detectar bordes o aplicar filtros gaussianos. En astronomía, se utiliza un kernel para modelar la PSF (*Point Spread Function*).

La PSF describe la respuesta de un sistema óptico frente a un impulso. Aplica un emborronamiento a la imagen, perdiendo detalles y bordes y reduciendo el ruido de más alta frecuencia. Este tipo de distorsión es el que tiene lugar en los sístemas ópticos. Por ejemplo, puede modelarse como un filtro gaussiano o un filtro promedio. En la figura 2.1 pueden observarse dos PSF diferentes.

2.1.2. Transformada de Fourier

La Transformada de Fourier descompone una función en una suma de funciones sinusoidales de diferentes frecuencias. La Transformada de Fourier presenta una gran cantidad de propiedades y usos en diversas áreas. La expresión que permite calcular la Transformada de Fourier de una función continua se expresa como:

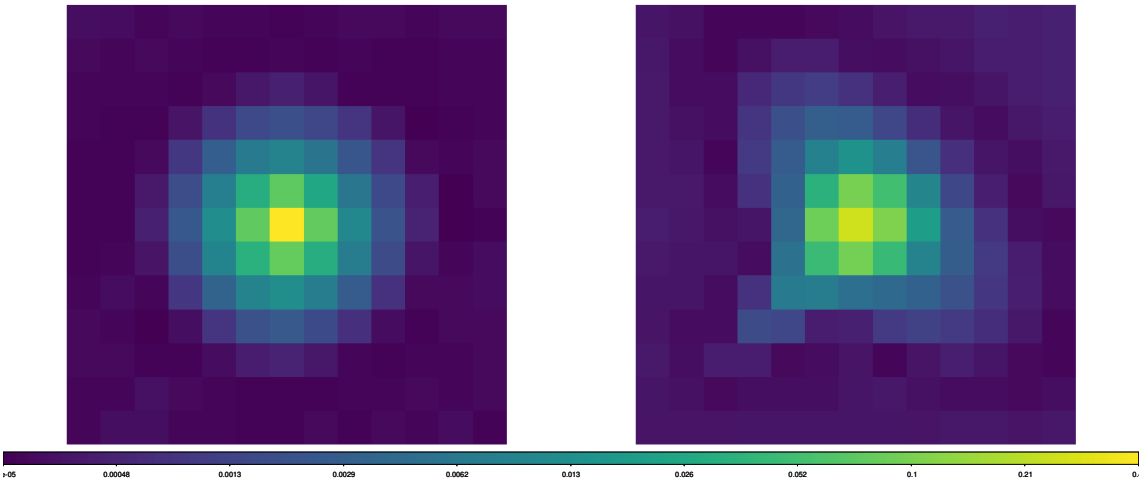

Figura 2.1: Ejemplo de dos PSF.

$$\mathbf{F(s)} = \int_{-\infty}^{\infty} \mathbf{f(x)e^{-i2\pi xs}dx} \tag{2.2}$$

Y su transformada inversa:

$$\mathbf{f(x)} = \int_{-\infty}^{\infty} \mathbf{F(s)e^{i2\pi xs}ds} \tag{2.3}$$

Para el contexto de este trabajo hay una propiedad a destacar, el teorema de la convolución. En el dominio de la frecuencia, la convolución se convierte en una simple multiplicación (Hoffman, 1997). Dicho de otra manera si $g(x) = f(x) * h(x)$, en el dominio de Fourier o frecuencial se puede expresar como $G(s) = F(s)H(s)$.

Hasta ahora la Transformada de Fourier ha sido descrita para funciones continuas. Sin embargo, las imágenes se pueden interpretar como funciones discretas. Para funciones discretas la teoría descrita también es aplicable. Las ecuaciones para pasar del dominio espacial al frecuencial tienen la siguiente forma:

$$\mathbf{X_k} = \sum_{\mathbf{n=0}}^{\mathbf{N-1}} \mathbf{x_n e^{-\frac{2\pi i}{N}kn}} \tag{2.4}$$

Y la transformada inversa discreta:

$$\mathbf{x_n} = \frac{\mathbf{1}}{\mathbf{N}} \sum_{\mathbf{n=0}}^{\mathbf{N-1}} \mathbf{X_k e^{\frac{2\pi i}{N}kn}} \tag{2.5}$$

En el caso concreto de este trabajo, se requiere aplicar la Transformada de Fourier a imágenes, por lo que se hará referencia a la Transformada de Fourier en dos dimensiones.

Esta ampliación a dos dimensiones es fácil de aplicar. Primero, se divide una imagen por filas y a cada fila se le aplica una Transformada de Fourier. Después, esta nueva imagen se divide por columnas y se le vuelve a aplicar una Transformada de Fourier a cada columna. Una característica muy útil es que cada fila y cada columna es independiente y, por lo tanto, se pueden calcular en paralelo.

Por último, cabe destacar que existen algoritmos que calculan la Transformada de Fourier de manera más eficiente computacionalmente conocidos como Transformada de Fourier rápida (FFT por sus siglas en inglés). Estos algoritmos serán utilizados siempre que se necesite trabajar en el dominio de la frecuencia y normalmente se denotarán por FFT a la transformada directa y FFT^{-1} a su transformada inversa.

2.1.3. Transformada Wavelet

La transformada de Fourier es una herramienta útil para analizar las componentes de frecuencia de la señal. Sin embargo, al tomar la transformada de Fourier sobre todo el eje temporal, no podemos saber en qué instante una determinada frecuencia aumenta o disminuye (Chun-Lin, 2010).

Por este motivo, en ocasiones es beneficioso recurrir a herramientas más sofisticadas, como la transformada wavelet. La transformada wavelet permite extraer información tanto de las variaciones estacionarias como no estacionarias de una señal en tiempo y frecuencia, es decir, identificar su frecuencia de aparición y su localización temporal.

Existen dos tipos de transformadas wavelet: la transformada wavelet continua (CWT) y la transformada wavelet discreta, que incluye versiones diezmada (DWT) y no diezmada (NDWT) (Brassarote, Souza, y Monico, 2018). En la wavelet no diezmada no se reduce el muestreo en la salida, por lo que cada escala tendrá el mismo número de coeficientes wavelet. Además, es invariante a traslaciones. Esta categoría de wavelet será la única utilizada en este trabajo. A menos que se indique lo contrario, el término transformada wavelet hará referencia a la transformada wavelet no diezmada.

También es importante introducir el concepto de función padre. La función padre es un kernel que realiza una función de emborronado o distorsión, y en cada iteración se duplica el tamaño del kernel. Por ejemplo, si inicialmente tiene unas dimensiones de 5×5, en la siguiente iteración será de 10×10, luego de 20×20, y así sucesivamente. La siguiente matriz muestra una posible función padre para la transformada wavelet:

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

El proceso de descomposición en D planos siguiendo el algoritmo no diezmado se puede describir de la siguiente manera:

$$\begin{aligned} \mathbf{R_t} &= \mathbf{R_{t-1}} * \mathbf{F_t}, \\ \mathbf{w_t^h} &= \mathbf{R_{t-1}} - \mathbf{R_t} \end{aligned} \tag{2.6}$$

Donde t es el plano correspondiente, R_0 equivale a la imagen de entrada que se quiere descomponer y F_t es la función padre expandida en la iteración t. El resultado final consta de D+1 imágenes, siendo $W_{D+1} = R_D$. R_t es conocido como el residuo de la iteración.

Debido a que en la descomposición no se ha utilizado un submuestreo, esta descomposición corresponde a la transformada Starlet (J. Starck, Murtagh, y Bertero, 2011). Esto implica que la resolución se ha conservado entre planos y que la imagen original se puede recuperar de manera directa. La transformación inversa, dada su transformada Starlet, se puede expresar como:

$$\mathbf{I} = \sum_{\mathbf{t=1}}^{\mathbf{D}} \mathbf{w_t^h} \tag{2.7}$$

Donde I es la imagen original que fue descompuesta en $D + 1$ imágenes. En la figura 2.2 puede verse una imagen de Marte y su descomponsición wavelet para D=4.

2.1.4. Modelado del ruido

Las imágenes digitales se ven afectadas por diferentes fenómenos durante su captación, por ejemplo debido a la electrónica subyacente. Esto se conoce como ruido y es importante caracterizarlo para poder trabajar con él. En este trabajo se va a trabajar con dos tipos de ruido: gaussiano y de Poisson.

El ruido gaussiano es originado por intereferencias electrónicas y errores en la cuantificación, se rige por una distribución normal:

$$\mathbf{P(z)} = \frac{\mathbf{1}}{\sqrt{\mathbf{2}\pi\sigma}} \mathbf{e}^{-\frac{\mathbf{z}-\bar{\mathbf{z}}}{\mathbf{2}\sigma^{\mathbf{2}}}} \tag{2.8}$$

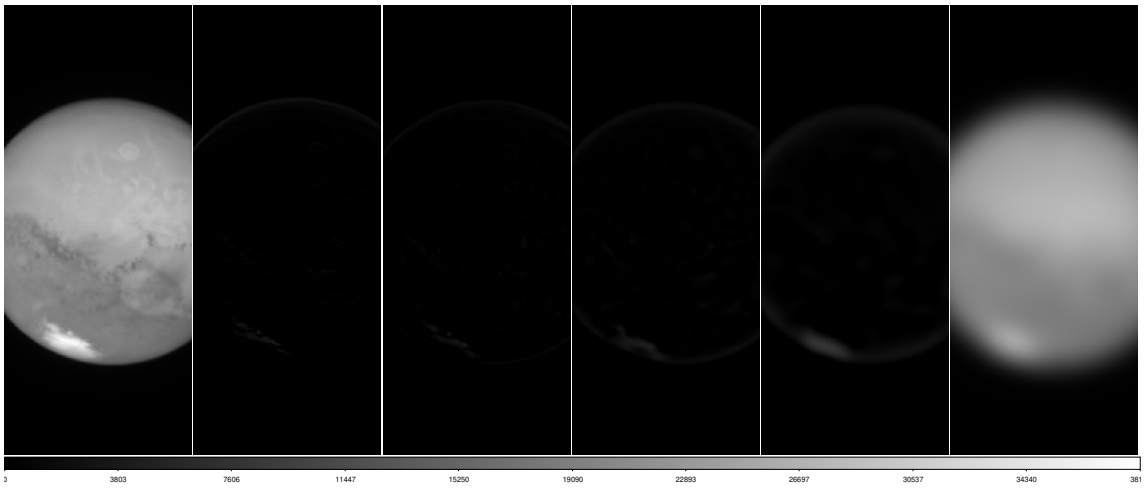

Figura 2.2: Descomposición wavelet de una imagen de Marte. La imagen de la izquierda corresponde con la imagen original y la derecha con el residuo.

Donde σ^2 es la varianza y $\bar{z}$ es la media. La varianza describe como de disperso está el ruido y la media indica el valor central del ruido.

El ruido de poisson, también conocido como ruido de disparo, es un ruido asociado directamente con la medición de la luz, inherente a la naturaleza cuantizada de la luz y a la independencia de las detecciones de fotones. Su magnitud esperada depende de la señal, es inherente a la medición de la luz, no tiene parámetros y es independiente de otras fuentes de ruido (Hasinoff, 2021). Esto significa que, para señales intensas, el ruido absoluto es mayor, pero relativo a la señal, es menor. Para el ruido de Poisson, la probabilidad de contar k fotones, dado un valor promedio λ, está dada por la siguiente expresión:

$$P(k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{2.9}$$

2.2. Deconvolución

Las imágenes astronómicas se degradan fácilmente debido a diversas causas como los defectos inherentes a los aparatos de medición, la óptica de la lente o las turbulencias atmosféricas. Si se consideran estas distorsiones como lineales e invariantes respecto al espacio, podrían modelarse como una convolución entre la PSF (kernel que modela la distorsión de una lente) y la imagen original. Desgraciadamente, esto normalmente no ocurre y usualmente no cumplen dichas condiciones (Jefferies y Christou, 1993) y por tanto hay que añadir un término de ruido no lineal. Una imagen detectada, $I(x, y)$, puede

ser expresada como:

$$I(x,y) = (P * O)(x,y) + N(x,y) \tag{2.10}$$

Donde $I(x,y)$ es la detección realizada por el equipo, $P(x,y)$ es la PSF, $O(x,y)$ es la imagen "ideal" que se pretende reconstruir y $N(x,y)$ es un término de ruido que incluye todas las distorsiones no lineales. La expresión asume que la PSF es lineal. Esta misma expresión puede convertirse a dominio de la frecuencia mediante la transformada de Fourier para facilitar la convolución:

$$\hat{I}(u,v) = \hat{P}(u,v)\hat{O}(u,v) + \hat{N}(u,v) \tag{2.11}$$

El dominio de la frecuencia está denotado por $\hat{y}(u,v)$, donde u y v son frecuencias espaciales.

2.3. Algoritmos de deconvolución

Una posible y lógica solución sería despejar $\hat{O}$(u,v) dividiendo la imagen entre la PSF en el dominio de Fourier y aplicar la transformada inversa de Fourier tal y como se muestra en la ecuación 2.12.

$$O(x,y) = FFT^{-1}\left(\frac{\hat{I}}{\hat{P}}\right) = FFT^{-1}(\hat{O} + \left(\frac{\hat{N}}{\hat{P}}\right)) \tag{2.12}$$

Sin embargo, este procedimiento da pésimos resultados ya que la componente de ruido, al ser dividida por valores pequeños cercanos a la frecuencia de corte de la PSF corrompe la imagen. Concretamente, las altas frecuencias donde el ruido es dominante, crea fuertes distorsiones en el resultado final. Por lo tanto, el problema de la amplificación del ruido se plantea siempre en la deconvolución, esto se denomina "condicionamiento deficiente" en la teoría de problemas inversos (Thiebaut, 2006). Igualmente, ha sido implementado para poder usarse en comparaciones bajo el nombre de inversión directa. Logicamente, este algoritmo asume que no hay ruido, lo cual nunca ocurre en situaciones reales debido a la propia naturaleza del ruido de disparo, asociado a la propia señal.

2.3.1. Wiener-Tikhonov

El algoritmo de Wiener-Tikhonov pertenece a los métodos de regularización lineal. Estos métodos se basan en minimizar la norma de $|I(x,y) - (P * O)(x,y)|$, resultando en un problema de mínimos cuadráticos. Este problema también presenta dificultades

con la presencia de ruido en la imagen, para contrarrestarlo, se introduce un término de regularización.

Una de las regularizaciones mencionadas es la regularización de Tikhonov. La solución final en el dominio de Fourier tiene la siguiente forma:

$$\hat{O}(u,v) \approx \frac{\hat{P}^*(u,v)\hat{I}(u,v)}{|\hat{P}(u,v)|^2 + \lambda|\hat{H}(u,v)|^2} \tag{2.13}$$

Donde λ es un parámetro de la regularización el cual ofrece un equilibrio entre fidelidad y suavidad en la imagen restaurada (Murtagh, Starck, y Pantin, 2002). Existen algunos métodos de validación cruzada (Golub, Heath, y Wahba, 1979) (Galatsanos y Katsaggelos, 1992) para calcular el valor de λ, pero quedan fuera del alcance de este trabajo. La función $\hat{P}^*(u,v)$ es la conjugada de la PSF en el dominio de la frecuencia y $\hat{H}(u,v)$ es un filtro que permite destacar las altas frecuencias. Debido a que las imágenes astronómicas de galaxias no presentan bordes bien definidos y que no es necesario para el desarrollo del proyecto, se va a sustituir $\hat{H}(u,v)$ de la ecuación por la unidad, quedando:

$$\hat{O}(u,v) \approx \frac{\hat{P}^*(u,v)\hat{I}(u,v)}{|\hat{P}(u,v)|^2 + \lambda} \tag{2.14}$$

Este algoritmo asume ruido gaussiano.

2.3.2. Richardson-Lucy

El algoritmo iterativo de Richardson-Lucy está basado en un enfoque bayesiano, supone que los valores de los píxeles siguen una distribución de Poisson y proporciona una solución de máxima verosimilitud.

El enfoque bayesiano se basa en maximar la probabilidad de que una observación sea parte de la imagen original a partir de la la probabilidad condicionada:

$$p(O|I) = \frac{p(I|O)p(O)}{p(I)} \tag{2.15}$$

La solución se encuentra maximizando la parte derecha de la ecuación mediante máxima verosimilitud. Los métodos de máxima verosimilitud se utilizan habitualmente para estimar parámetros a partir de datos ruidosos, normalmente se utilizan con restricciones adicionales. Es decir, se asume que P(O) es diferente de 1. Cuando es 1 entonces es de máxima verosimilitud como el RL.

El algoritmo, que fue obtenido independientemente por Richardson (Richardson, 1972) y Lucy (Lucy, 1974), tiene la siguiente forma:

$$O^{n+1}(x,y) = KO^n(x,y) * \left[\frac{I(x,y)}{(P * O^n)(x,y)} * P^*(x,y) \right]^{\alpha} \tag{2.16}$$

Donde $O^n(x,y)$ es la estimación en la iteración número n y $O^{n+1}(x,y)$ es la estimación en la iteración $n+1$. También se introduce un parámetro α que permite ajustar la velocidad del algoritmo (con un valor por defecto igual a 1). El parametro K se utiliza para ajustar los valores de los píxeles de modo que la suma total de los mismos se mantenga constante entre iteraciones sucesivas. Un ejemplo de pseudocódigo para el algoritmo de Richardson-Lucy es el siguiente:

1. Preprocesado: Ajustar tamaños de I y P.
2. Inicializar: O^0
3. $\hat{P} = \text{FFT}(P)$ (PSF en el dominio de Fourier)
4. $\hat{P}^* = conjugar(\hat{P})$ (PSF^* en el dominio de Fourier)
5. Para $k = 0, ..., n-1$:
   a) $\hat{I}^k = \text{FFT}^{-1}(\hat{P} \times \text{FFT}(O^{(k)}))$ (Modelo k-ésimo)
   b) $O^{(k+1)} = O^{(k)} \times \left[\text{FFT}^{-1}(\hat{P^*} \times \text{FFT}(I/\hat{I}^k))\right]^{\alpha}$ (Nueva estimación)
   c) $K = sum(O^k)/sum(O^{k+1})$ (Calcular K)
   d) $O^{k+1} = KO^{k+1}$ (Ajustar la nueva estimación)
6. Return $O^{(n)}$

Un problema de este algoritmo se ve claramente en su pseudocódigo, cada iteración requiere de 4 FFT (dos directas y dos inversas), por lo que puede llegar a consumir mucho tiempo y recursos en imágenes grandes que requieran muchas iteraciones. Además, se ha observado que el ruido puede incrementarse después de un número excesivo de iteraciones, lo que sugiere la existencia de un valor óptimo de iteraciones para el algoritmo de Richardson-Lucy. A partir de este punto, en lugar de mejorar la calidad de la imagen, el algoritmo comienza a introducir más ruido y artefactos, lo que empeora el resultado final. Este algoritmo asume ruido de Poisson únicamente.

2.3.3. AWMLE

El algoritmo AWMLE (*Adaptive Wavelets Maximum Likelihood Estimator*) puede entenderse como una ampliación del algoritmo previamente descrito Richardson-Lucy. Este algoritmo además asume que la imagen presenta tanto ruido de Poisson como gaussiano. El algoritmo AWMLE realiza una deconvolución multicanal en cada uno de los planos de una descomposición wavelet (Baena-Gallé y Gladysz, 2011). Esto presenta varias ventajas, pero cabe destacar que los ruidos de Poisson y Gauss se concentran en los planos de más alta frecuencia (J.-L. Starck, Murtagh, y Bijaoui, 1998), permitiendo adaptar la deconvolución en función de la relación señal-ruido del plano.

Al igual que el algoritmo de Richardson-Lucy, es un algoritmo iterativo. De manera simplificada, la ecuación que describe como es la estimación en la siguiente iteración es:

$$O^{n+1}(x,y) = KO^n(x,y) * \left[P^*(x,y) * \frac{\sum_{t=1}^{D}[(w_t^h + m_t(w_t^p - w_t^h)]}{(P * O^n)(x,y)} \right]^{\alpha} \tag{2.17}$$

Donde $O^n(x,y)$, $O^{n+1}(x,y)$, α y K desempeñan la misma función que en el algoritmo de Richardson-Lucy. w_t^h es la descomposición wavelet de $(P' * O^n)(x,y)$, w_t^p es la descomposición wavelet de la imagen observada con ruido y m_t es una máscara estadística que determina si realmente hay señal o no es un píxel concreto. La máscara estadística puede calcularse como:

$$f(\sigma_i, \sigma_\omega) = \begin{cases} 1 - \exp\left(-\frac{(\sigma_i^2 - \sigma_\omega^2)}{16\sigma_\omega^2}\right) & \text{si } \sigma_i - \sigma_\omega > 0 \\ 0 & \text{si } \sigma_i - \sigma_\omega \leqslant 0 \end{cases} \tag{2.18}$$

Con

$$\sigma_i = \sqrt{\frac{\sum_{j\in\phi}(\omega_{t,j})^2}{n_f}} \tag{2.19}$$

Donde ω_i es la desviación típica dentro de la ventana ϕ centrada en el píxel i del plano wavelet ω_j, ω_j es la desviación típica en todo el plano wavelet y n_f es el número de píxeles en la ventana. La decomposición wavelet llevada a cabo en este algoritmo es no diezmada, facilitando así su transformación inversa (es el sumatorio en la ecuación 2.17).

2.4. Gnuastro

En el momento en el que se redacta este trabajo, Gnuastro no ha llegado aún a su versión 1.0. No existe un artículo que se centre en introducir y detallar todas las funcionalidades de Gnuastro. Actualmente, cuenta con los siguientes programas:

Arithmetic (astarithmetic): Permite aplicar operaciones aritméticas (como sumar, restar, multiplicar...) a un conjunto de imágenes o datasets.

BuildProgram (astbuildprog): Permite crear programas ejecutables que dependen de Gnuastro.

ConvertType (astconvertt): Convierte imágenes astronómicas en formato FITS or IMH a formatos de imágenes estándar como JPEG o EPS.

Convolve (astconvolve): Aplica una convolución a una imagen dado un kernel.

CosmicCalculator (astcosmiccal): Resuelve cálculos cosmológicos como la distancia módulo o distancia luminosa.

Crop (astcrop): Recorta un subconjunto de una imagen en una nueva imagen.

Fits (astfits): Permite ver y manipular archivos FITS y sus cabeceras.

MakeCatalog (astmkcatalog): Crea un catálogo de imágenes etiquetadas. Más información (Akhlaghi, 2019).

Match (astmatch): Dados dos catálogos, encuentra la fila que coincide dada una apertura.

NoiseChisel (astnoisechisel): Detecta señal en el ruido incluso por debajo del nivel de ruido existente, (Akhlaghi y Ichikawa, 2015) para más detalles.

Query(astquery): Interfaz de alto nivel para hacer peticiones predefinas a bases de datos.

Segment (astsegment): Segmenta regiones en una imagen.

Statistics (aststatistics): Calcula operaciones estadísticas en una imagen como la media o la varianza.

Table (asttable): Convierte archivos FITS o tablas ASCII en otro tipo de tablas.

Warp (astwarp): Ajusta una imagen a una nueva resolución de píxeles.

Además también existen *scripts* que permiten realizar operaciones de más alto nivel, estos *scripts* empiezan con el prefijo *astscript-*. Por ejemplo *astscript-fits-view* permite visualizar una o varias imagenes FITS ajustando automaticamente algunos parámetros de visualización.

3. Objetivos

3.1. Objetivo general

Aportar una librería para el proyecto Gnuastro que permita realizar funciones de deconvolución y sea incluida en la distribución oficial, facilitando su acceso y utilidad para cualquier persona que la necesite.

3.2. Objetivos específicos

Los objetivos específicos son los siguientes :

- Analizar la estructura, reglas y normas del proyecto Gnuastro.
- Implementar los diferentes algoritmos de deconvolución en C: Inversión directa, Tikhonov, Richardson-Lucy y AWMLE.
- Validar el resultado de los algoritmos.
- Aplicar todas las normas y requerimientos al código que sean requeridos por Gnuastro.
- Mejorar otras áreas de Gnuastro que estén estrechamente ligadas con la deconvolución.
- Seguir las indicaciones de los administradores de Gnuastro para conseguir la aceptación del código generado en este trabajo.

4. Identificación de Requisitos

Dada la naturaleza del trabajo, los requisitos se pueden separar en dos grandes grupos: aquellos que provienen de las reglas de Gnuastro y aquellos que provienen del objetivo principal.

4.1. Requisitos de Gnuastro

Los requisitos de Gnuastro pueden encontrarse más en detalle en la sección 13 (*Developing*) de la documentación oficial (GNU Astronomy Utilities, s.f.). Aquí se muestra una versión simplificada en castellano.

Lenguaje de programación: Debe utilizarse C, ya que Gnuastro está completamente escrito en dicho lenguaje por su simplicidad, eficiencia y portabilidad.

Legibilidad: El código debe ser fácil de entender de un vistazo.

Tamaño de función: Una función debe poder verse por completo en un monitor (aproximadamente 40 líneas como máximo).

Longitud de línea: Máximo 75 caracteres.

Dependencias: Todas las cabeceras que necesite un fichero .c deben estar incluidas dentro del mismo.

Orden de dependencias: A la hora de incluir ficheros .h debe hacerse en un orden específico dejando una línea en blanco entre los distintos grupos.

Nombres de funciones: Todas la variables y funciones deben ir en minúsculas (*lowercase*). Las enumeraciones y constantes en mayúsculas separadas por "_"(*scream snake case*).

Prefijo gal: Todas las cabeceras, funciones, variables y macros que sean exportables deben comenzar por el prefijo gal_.

Nombre base: Todas las funciones y variables deben incluir el nombre base como prefijo separado por barra baja. Por ejemplo, una función exportable dentro de la librería fft debe comenzar por gal_fft_NOMBRE mientras que una función interna quedaría como fft_NOMBRE.

Espacios en blanco: No debe haber ningún espacio en blanco al final de una línea.

Tabulaciones: No debe haber tabulaciones para indentar.

4.2. Requisitos específicos

Algoritmo inversión directa: Debe implementarse un algoritmo capaz de realizar la deconvolución simple (dividir la imagen entre la función de dispersión) y estar disponible para cualquier usuario de Gnuastro.

Algoritmo Tikhonov: Debe implementarse el algoritmo de Tikhonov y estar disponible para cualquier usuario de Gnuastro.

Algoritmo Richardson-Lucy: Debe implementarse el algoritmo de Richardson-Lucy y estar disponible para cualquier usuario de Gnuastro.

Algoritmo AWMLE: Debe implementarse el algoritmo de AWMLE y estar disponible para cualquier usuario de Gnuastro.

Parametrización de los algoritmos: Los parámetros que sean requeridos por los algoritmos (como número de iteraciones), deben de ser leídos desde línea de comandos o fichero de configuración.

Librerías de bajo nivel: Aquellas librería auxiliares para el desarrollo de los algoritmos también deben estar accesibles para futuros desarrollos.

Programa deconvolución: Debe crearse un programa de alto nivel que pueda ser llamado desde la terminal de comandos bajo el nombre de *astdeconvolve*.

Validación de los algoritmos: Todos los algoritmos deben ser validados y si fuese posible se debería estimar en que rango de ruido y distorsión funcionan correctamente.

5. Desarrollo del trabajo

5.1. Análisis e instalación de Gnuastro

Toda la información necesaria para trabajar con Gnuastro se puede encontrar en su página web oficial (Gnuastro Savannah Repository, s.f.). Es importante destacar que el código fuente no está alojado en GitHub debido a que Github no es un proyecto de código abierto; en cambio, se encuentra disponible en Savannah.

Todo el trabajo se realizó utilizando un subsistema Linux para Windows (WSL2), dado que las distribuciones están diseñadas exclusivamente para sistemas basados en Linux. Concretamente se instaló un sistema Ubuntu en su versión 22.04. Para el desarrolo del código se usó Visual Studio Code debido a su simplicidad y buena integración con WSL2. Además, Visual Studio Code facilita el formateo automático del código para ajustarse a las convenciones de GNU.

Gnuastro es fácil de instalar siguiendo los tutoriales de su web. Una vez instalado Gnuastro, se puede acceder a sus funcionalidades desde la consola de comandos. En la imagen 5.1 se puede observar una terminal en la que primero se escribe 'ast' y al presionar dos veces la tecla de tabulación se despliegan todas las posibles opciones. Después, se muestra un ejemplo de ayuda para el comando 'astcrop'.

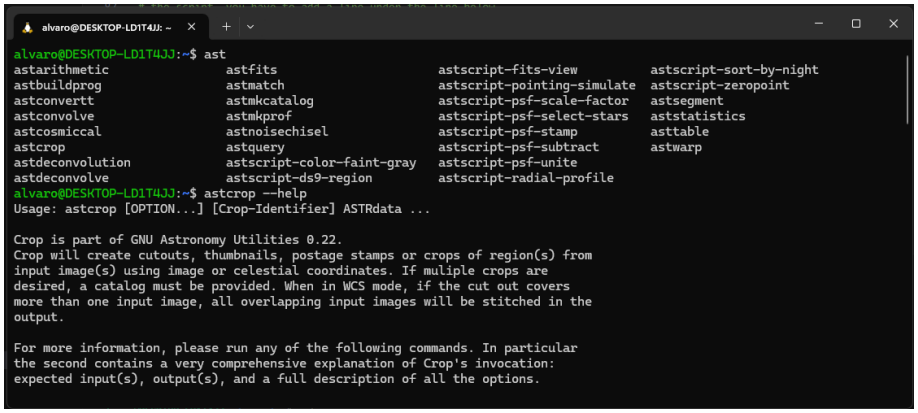

Figura 5.1: Terminal con Gnuastro instalado

Dentro del proyecto se encuentran los siguientes directorios de interés para este trabajo:

bin : Contiene las aplicaciones o programas que los usuarios ejecutan desde la consola de comandos. Aquí se validan los argumentos y opciones necesarios antes de ejecutar la

función llamada. Para este trabajo se creó el programa *deconvolve*.

lib : Contiene las bibliotecas necesarias para los diferentes programas. Los ficheros .c se encuentran en este directorio y los .h en el subdirectorio /lib/gnuastro. Para este trabajo se crearon bibliotecas para la deconvolución (*deconvolve*), para trabajar con vectores complejos (*complex*) y para realizar transformaciones de Fourier (*fft*).

doc : Contiene la documentación del proyecto. Está escrita en formato textinfo, similar a LaTeX.

5.2. Desarrollo de las librerías

5.2.1. Desarrollo de la librería complex.h

La librería *complex.h* implementa operaciones con vectores de números complejos. Se basa en las funcionalidades de la librería *complex* de GSL, una dependencia de Gnuastro que permite realizar cálculos matemáticos (*GNU Scientific Library*). Los datos se representan con el tipo *gsl_complex_packed_array*, que es un array de tipo *double* en el cual los índices pares corresponden a la parte real y los índices impares a la parte imaginaria. Por lo tanto, cada número complejo ocupa dos *doubles*.

Las funciones de esta librería pueden categorizarse de la siguiente manera:

- **Conversión y Preparación**: Funciones que convierten y preparan datos entre formatos reales y complejos. Por ejemplo, `gal_complex_create_padding`, `gal_complex_real_to_complex`, `gal_complex_to_real`.
- **Operaciones Elementales**: Funciones que normalizan, conjugan, escalan, suman escalares o elevan a potencias los elementos de un array complejo. Por ejemplo, `gal_complex_normalize`, `gal_complex_conjugate`, `gal_complex_scale`, `gal_complex_add_scalar`, `gal_complex_power`.
- **Operaciones entre Arrays**: Funciones que realizan multiplicación, división y resta entre elementos de dos arrays complejos. Por ejemplo, `gal_complex_multiply`, `gal_complex_divide`, `gal_complex_substract`.
- **Operaciones Acumulativas**: Funciones que suman todos los elementos de un array complejo. Por ejemplo, `gal_complex_cumulative_sum`.

5.2.2. Desarrollo de la librería fft.h

La librería *fft.h* implementa la transformada de Fourier rápida (*Fast Fourier Transform*). Se basa en las funcionalidades de la librería *gsl_fft_complex* de GSL y también trabaja con el tipo de datos *gsl_complex_packed_array*.

Permite realizar tanto la transformada directa como la inversa y tiene la posibilidad de ser ejecutada en múltiples hilos.

Presenta únicamente dos funciones para el usuario: una para realizar la transformada FFT y otra para desplazar el centro de una imagen. Esta segunda función se necesita ya que el centro de referencia de un kernel no coincide con el esperado para la transformada de Fourier. Para corregir esto, se realiza un desplazamiento horizontal y vertical, de tal manera que el centro de la imagen original termine en el píxel más abajo a la izquierda.

Además se han realizado funciones internas para gestionar los recursos necesarios para la fft y para llevar a cabo la fft en una única dirección.

5.2.3. Desarrollo de la librería wavelet.h

La librería *wavelet.h* implementa la transformada wavelet y otras funciones auxiliares. Esta implementación no se basa en ninguna librería externa; el algoritmo está implementado desde cero. No obstante, la librería utiliza funciones de *fft.h* y *complex.h* para realizar las transformadas rápidas de Fourier y operaciones con números complejos, respectivamente.

En este caso se ofrecen dos funciones para el usuario: una para la transformación wavelet no decimada y una segunda función para poder liberar los recursos de todos los planos wavelet de una transformada. El resultado de la transformada wavelet es una *linked list* de *gal_data_t* que contiene $N + 1$ elementos, donde N es el números de planos.

Además, también se incluyen funciones internas para hacer *padding*, generar y expandir la función padre y realizar operaciones entre planos.

5.2.4. Desarrollo de la librería deconvolve.h

La librería *deconvolve.h* implementa los algoritmos de deconvolución. Esta librería se apoya en las librerías previamente descritas *fft.h*, *complex.h* y *wavelet.h*, y es la librería de más alto nivel desarrollada en este trabajo.

La librería *deconvolve.h* incluye los cuatro algoritmos de deconvolución descritos en el marco teórico: Inversión directa, Tikhonov, Richardson-Lucy y AWMLE. Además, se han

implementado funciones internas para hacer el código de los algoritmos más modular y facilitar su mantenimiento y ampliación.

Todos los algoritmos necesitan tanto la PSF como la imagen a deconvolucionar. A continuación, se detallan los parámetros específicos requeridos por cada algoritmo:

- **Inversión Directa**: No necesita parámetros adicionales.
- **Tikhonov**: Requiere el parámetro de regularización λ.
- **Richardson-Lucy**: Necesita el número de iteraciones y el factor de crecimiento (α).
- **AWMLE**: Requiere el número de iteraciones, el número de planos wavelet, el factor de crecimiento (α), la tolerancia y la estimación del ruido gaussiano.

Como resultado, todos los algoritmos devuelven la imagen deconvolucionada. Además, en el caso de AWMLE, también se devuelve el número de iteración si se cumple el criterio de tolerancia.

Más detalles sobre las funciones implementadas se puede encontrar en el anexo A.

5.3. Desarrollo de `astdeconvolve`

`Astdeconvolve` es la aplicación final que será utilizada desde la terminal de comandos. El propósito de este módulo es enlazar las funciones de deconvolución previamente descritas con los parámetros que introducirá el usuario final, así como con las funciones de validación y lectura y escritura de ficheros.

Para su desarrollo, existe un directorio de ejemplo denominado *TEMPLATE* dentro de Gnuastro, el cual, al renombrarlo a *deconvolve*, permite desarrollar rápidamente un programa en Gnuastro. Este directorio presenta funciones para facilitar la validación de argumentos por terminal, así como la lectura de archivos.

En la tabla 5.1 se pueden ver las opciones y argumentos específicos para `astdeconvolve`. Por ejemplo, para aplicar el algoritmo de Tikhonov a una imagen *ejemplo.fits* usando una PSF *psf.fits*, con un valor de λ=0.1, el comando sería: `astdeconvolve ejemplo.fits --hdu=0 --kernel=psf.fits --khdu=0 -L 0.1 --algorithm=tikhonov`. El resultado sería una imagen *ejemplo_deconvolved.fits*.

Nombre	**Flags**	**Tipo**	**Descripción**
Algoritmo	-A --algorithm	string	Algoritmo a utilizar.
Imagen de entrada	Primer argumento	string	Ruta a la imagen de entrada (.fits).
HDU	-h --hdu	string	HDU correspondiente a la imagen de entrada.
Kernel	-k --kernel	string	Ruta a la PSF (.fits).
Kernel HDU	--khdu	string	HDU correspondiente al kernel.
Imagen de salida	-o --output	string	Ruta al archivo de salida (.fits)
Iteraciones	-i --iterations	integer	Número de iteraciones para Richardson-Lucy o AWMLE.
Alpha	-a --alpha	float	Factor de crecimiento para Richardson-Lucy o AWMLE.
Lambda	-L --tikhonov-lambda	float (0,1)	Parámetro lambda para el algoritmo Tikhonov
Planos wavelet	-w --awmle-waves	integer	Número de planos wavelets para AWMLE
Tolerancia	-t --awmle-tolerance	integer	Tolerancia para AWMLE
Desviación gaussiana	-s --awmle-sigma	float	desviación estándar del ruido gaussiano para AWMLE

Tabla 5.1: Argumentos de `astdeconvolve`

6. Evaluación

En esta sección se van a validar los algoritmos implementados para Gnuastro. Se dividirá en dos grandes bloques. El primer bloque evaluará cada algoritmo de manera individual, aplicando únicamente el tipo de ruido que cada algoritmo supone (Poisson, gaussiano o ambos). En el segundo bloque, se comparará el rendimiento de los algoritmos utilizando la misma entrada (imagen y cantidad de ruido) con simulaciones más realistas.

En los experimentos realizados para evaluar los algoritmos, se van a utilizar dos kernels (PSF) diferentes para simular la distorsión que ocurre durante la captura de la imagen. En la imagen 6.1 se pueden ver sus funciones de distribución. En general, cuando se mencione baja distorsión se hace referencia al kernel de la izquierda (PSF1), y alta distorsión al de la derecha (PSF2).

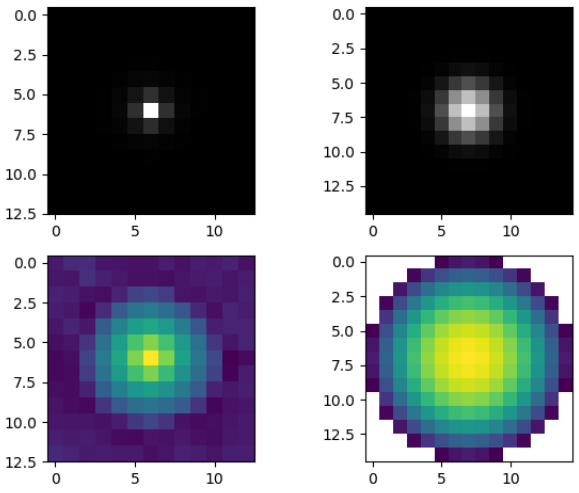

Figura 6.1: Kernels utilizados en los experimentos. Arriba a la izquierda: baja distorsión (PSF1) en escala natural, abajo a la izquierda: baja distorsión (PSF1) en escala logarítmica, arriba a la derecha: alta distorsión (PSF2) en escala natural, abajo a la derecha: alta distorsión (PSF2) en escala logarítmica.

6.1. Evaluación de los algoritmos

En este primer bloque de validación se pondrán a prueba los algoritmos de manera individual con varias imágenes. Como métrica principal se utilizará el RMSE (raíz del error cuadrático medio) entre la reconstrucción y la imagen original. Dado que los algoritmos

suelen tener parámetros de usuario que afectan al resultado, se realizarán múltiples experimentos variando dichos parámetros para comparar y determinar el parámetro óptimo, así como identificar aquellos que no son útiles. Además, a cada imagen se le aplicarán varios niveles de distorsión y ruido para evaluar cómo afectan a la reconstrucción. Para calcular el RMSE entre dos imágenes se utilizará la siguiente fórmula:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(A_i - B_i)^2} \tag{6.1}$$

Donde A y B son dos imágenes con las mismas dimensiones y un total de N píxeles. Dado que la diferencia se eleva al cuadrado no importa intercambiar la imagen A por la B.

En la figura 6.2 se muestran las dos imágenes seleccionadas para la evaluación: Saturno y la galaxia M100. Estas imágenes representan dos situaciones distintas. Saturno, al ser un planeta, presenta detalles visuales únicos gracias a sus anillos, mientras que M100, una galaxia espiral, exhibe una gran cantidad de estrellas y áreas con diferentes niveles de iluminación. La imagen de Saturno presenta un rango dinámico de intensidad de píxeles que va de 0.0 a 970.0427, mientras que la imagen de M100 tiene un rango de 44.41 a 30128.23.

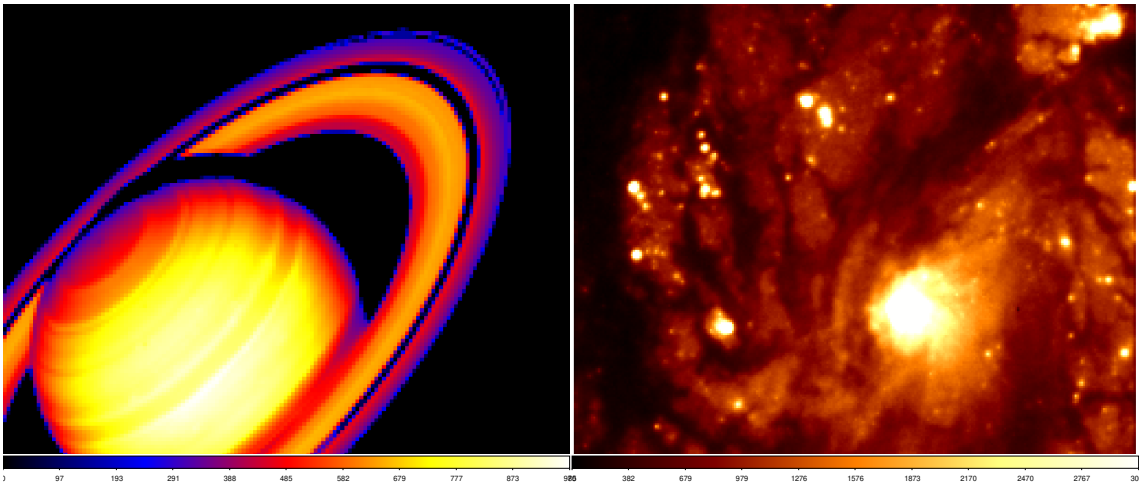

Figura 6.2: Imágenes seleccionadas para la evaluación de los algoritmos. A la izquierda se presenta Saturno, mientras que a la derecha se muestra la galaxia M100.

6.1.1. Evaluación del algoritmo de inversión directa

En primer lugar, se va a comenzar por la evaluación del algoritmo más simple, la inversión directa. Para este algoritmo, se presupone que las imágenes no tienen ruido. Además, al tratarse de un algoritmo que no tiene hiperparámetros a ajustar, no es necesario comparar distintos resultados.

En las figuras 6.3 y 6.4 puede verse la reconstrucción de la imagen de Saturno para baja y alta distorsión respectivamente. En ambos casos, visualmente es inapreciable una diferencia entre la imagen original y la reconstrucción; se aprecia una mejora positiva en la resolución espacial de la división de Encke entre la imagen distorsionada y la imagen reconstruida. En la figura 6.5 puede verse que en el caso de M100 con baja distorsión el algoritmo funciona perfectamente como en los casos anteriores. Sin embargo, en la figura 6.6 se puede ver cómo el resultado es ruido. No se ha encontrado una causa posible para este caso. Incluso modificando ligeramente el kernel (sumando un valor muy pequeño como 1e-6) para evitar ceros, no se ha encontrado ninguna diferencia.

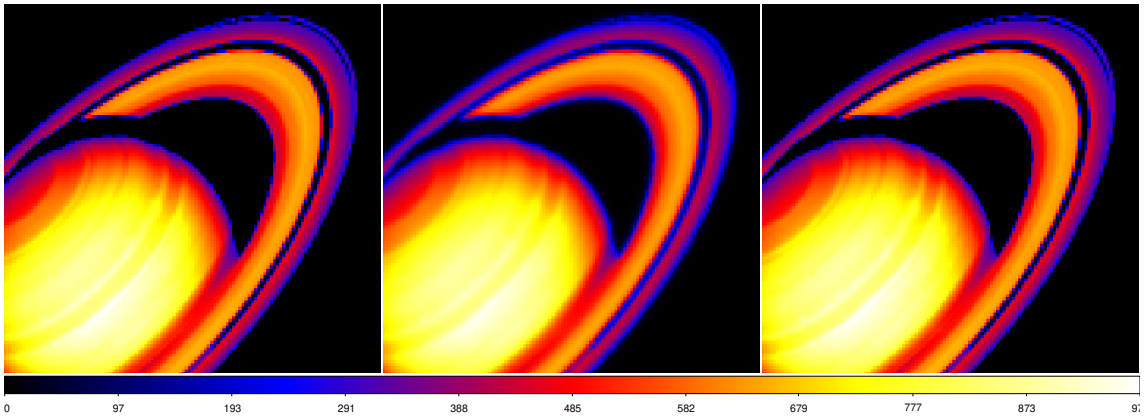

Figura 6.3: Evaluación del algoritmo de inversión directa en una imagen de Saturno con baja distorsión. De izquierda a derecha: Imagen original, imagen distorsionada, reconstrucción.

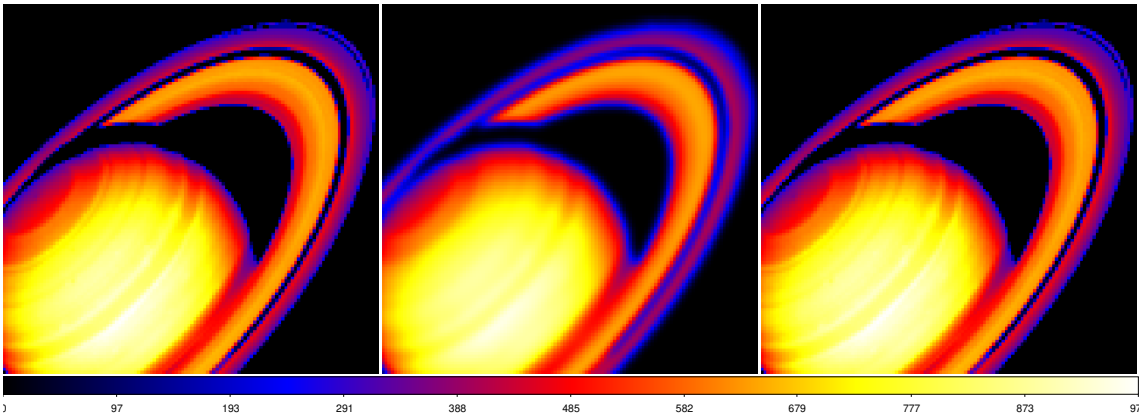

Figura 6.4: Evaluación del algoritmo de inversión directa en una imagen de Saturno con alta distorsión. De izquierda a derecha: Imagen original, imagen distorsionada, reconstrucción.

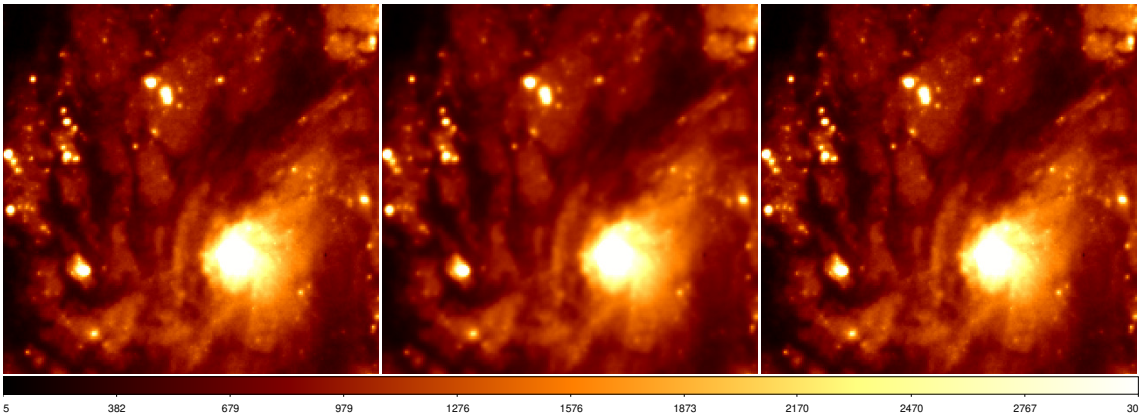

Figura 6.5: Evaluación del algoritmo de inversión directa en una imagen de M100 con baja distorsión. De izquierda a derecha: Imagen original, imagen distorsionada, reconstrucción.

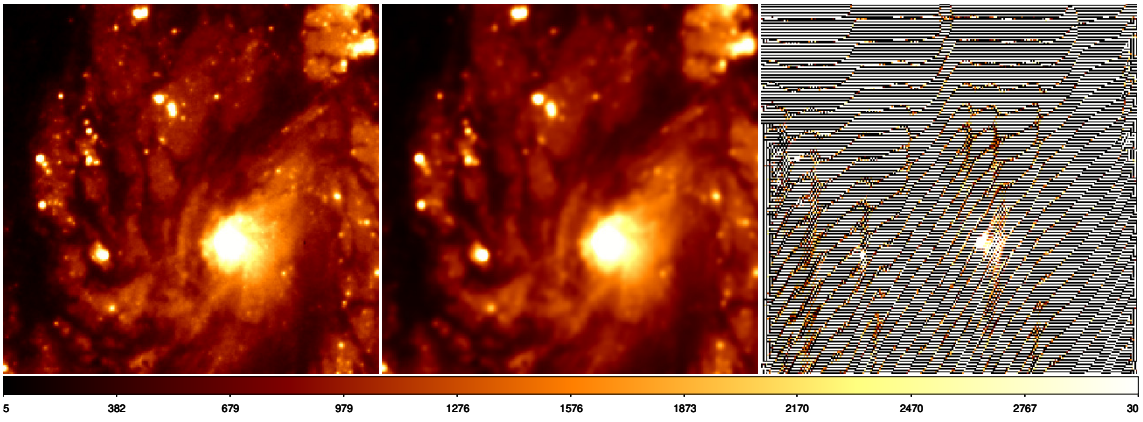

Figura 6.6: Evaluación del algoritmo de inversión directa en una imagen de Saturno con alta distorsión. De izquierda a derecha: Imagen original, imagen distorsionada, reconstrucción fallida.

6.1.2. Evaluación del algoritmo Tikhonov

A continuación, se evaluará el algoritmo de Tikhonov, cuyo único parámetro es λ. Para cada caso, se variará este parámetro con el fin de determinar la reconstrucción óptima (entre nivel de mejora de la resolución espacial y amplificación de ruido) y observar la evolución del RMSE. El algoritmo será evaluado añadiendo ruido gaussiano a las imágenes, con un 0.01 % y 0.1 % de la desviación estándar de la imagen original, para analizar el efecto de incrementar el ruido. Se evaluarán las imágenes de Saturno y M100 utilizando ambas PSF y ambos niveles de ruido, resultando en un total de 8 experimentos diferentes.

Imagen de Saturno

Para la imagen de Saturno, se probará el algoritmo primero con una imagen modificada con baja distorsión (PSF1) y ruido gaussiano con una desviación estándar de 2.32 (0.01 % de la desviación estándar de la imagen distorsionada). En la figura 6.7 se puede ver la comparativa entre la imagen original, la imagen solo con la distorsión, la imagen con distorsión y ruido, y la deconvolución. En la figura 6.8 se muestra cómo el parámetro λ afecta al RMSE entre la imagen original y cada una de las reconstrucciones. Como referencia, el RMSE de la imagen sin reconstruir es 17.88 y el RMSE para λ óptima ($\lambda = 0.005$) es 6.55.

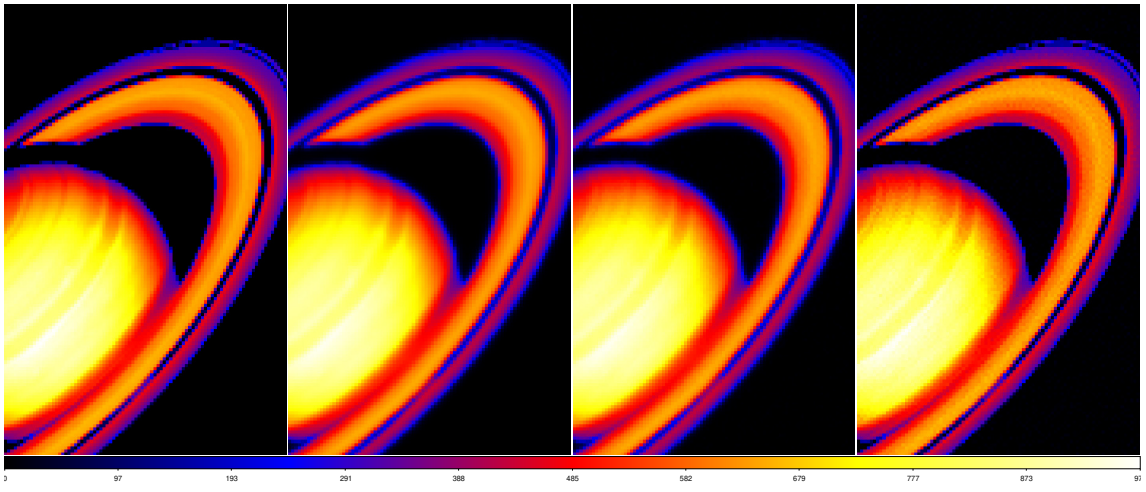

Figura 6.7: Evaluación del algoritmo de Tikhonov en una imagen de Saturno utilizando la PSF1 y ruido gaussiano con una desviación estándar de 2.32. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido, reconstrucción óptima ($\lambda = 5 \times 10^{-3}$).

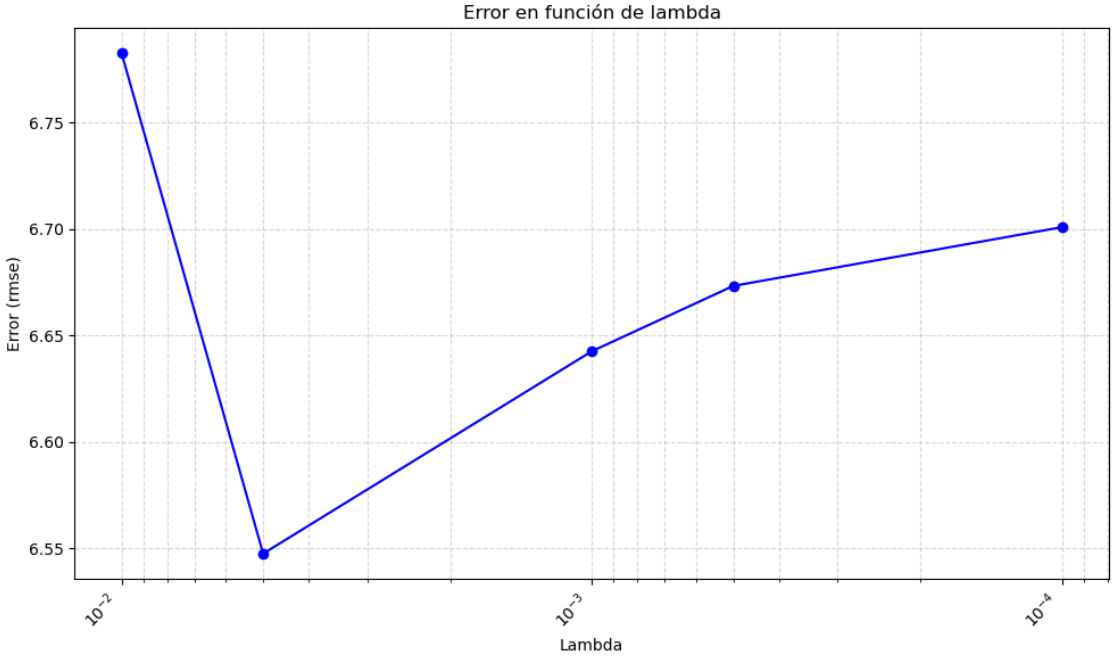

Figura 6.8: Evolución del RMSE entre la imagen original y la reconstrucción en función del parámetro lambda (Saturno con PSF1 y ruido gaussiano (σ=2.32)).

La segunda prueba consiste en aumentar el ruido gaussiano a 23.17 (0.1 % de la desviación estándar de la imagen distorsionada). En este experimento, se observa un efecto similar al del primer experimento. En la figura 6.9 se presenta una comparativa entre las diferentes fases de la reconstrucción. En la figura 6.10, se puede ver que el RMSE alcanza un valor mínimo de 18.55 para $\lambda = 0.005$.

A continuación, se van a repetir los dos experimentos anteriores, pero utilizando la PSF2, que introduce un mayor nivel de distorsión. En la figura 6.11, se presenta la comparativa entre la imagen original, las modificaciones y la reconstrucción final para un ruido gaussiano con una desviación estándar de 2.3 (0.01 % de la desviación estandar de la imagen distorsionada). La figura 6.12 muestra la evolución del RMSE en función de λ. En este caso, se observa un mínimo en $\lambda = 0.005$, y a partir de este valor, el RMSE empieza a aumentar considerablemente.

Por último, se presenta el caso más desfavorable, en el cual la desviación estándar del ruido gaussiano es diez veces mayor que en el caso anterior, siendo 23.1. El proceso de reconstrucción puede observarse en la figura 6.13, donde es evidente que el algoritmo comienza a no responder de manera ideal. En la figura 6.14, se puede ver la evolución del RMSE en función del parámetro λ.

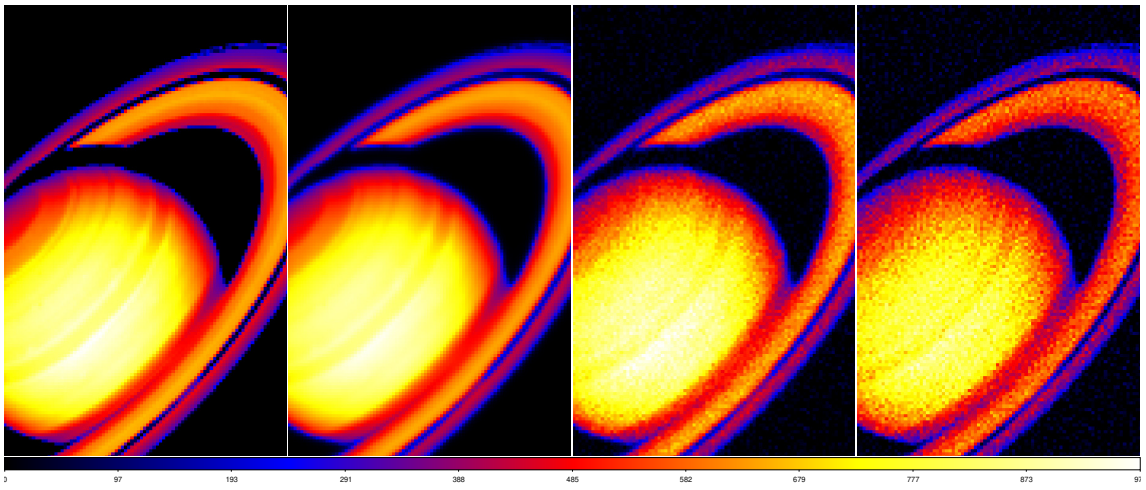

Figura 6.9: Evaluación del algoritmo de Tikhonov en una imagen de Saturno utilizando la PSF1 y ruido gaussiano con una desviación estándar de 23.17. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido, reconstrucción óptima ($\lambda = 5 \times 10^{-2}$).

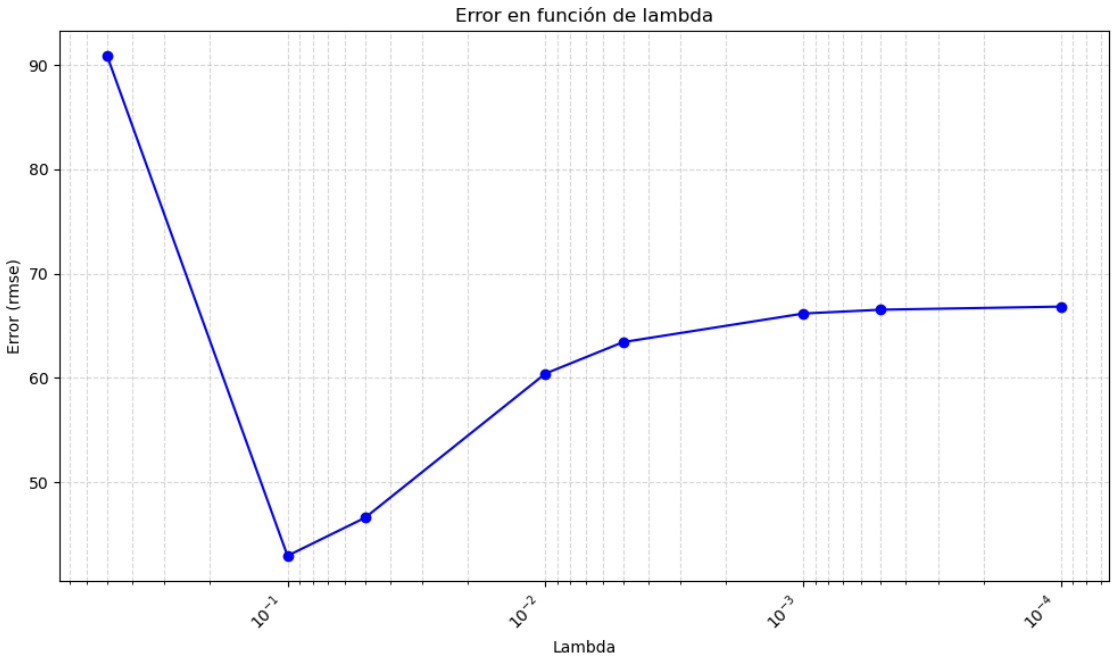

Figura 6.10: Evolución del RMSE entre la imagen original y la reconstrucción en función del parámetro lambda (Saturno con PSF1 y ruido gaussiano (σ=23.17)).

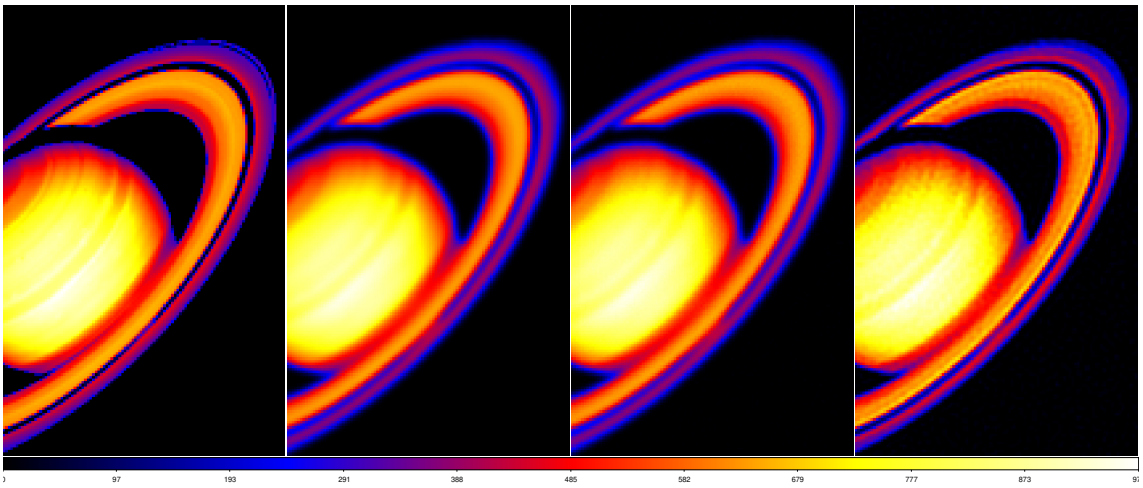

Figura 6.11: Evaluación del algoritmo de Tikhonov en una imagen de Saturno utilizando la PSF2 y ruido gaussiano con una desviación estándar de 2.3. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido, reconstrucción óptima ($\lambda = 5 \times 10^{-3}$).

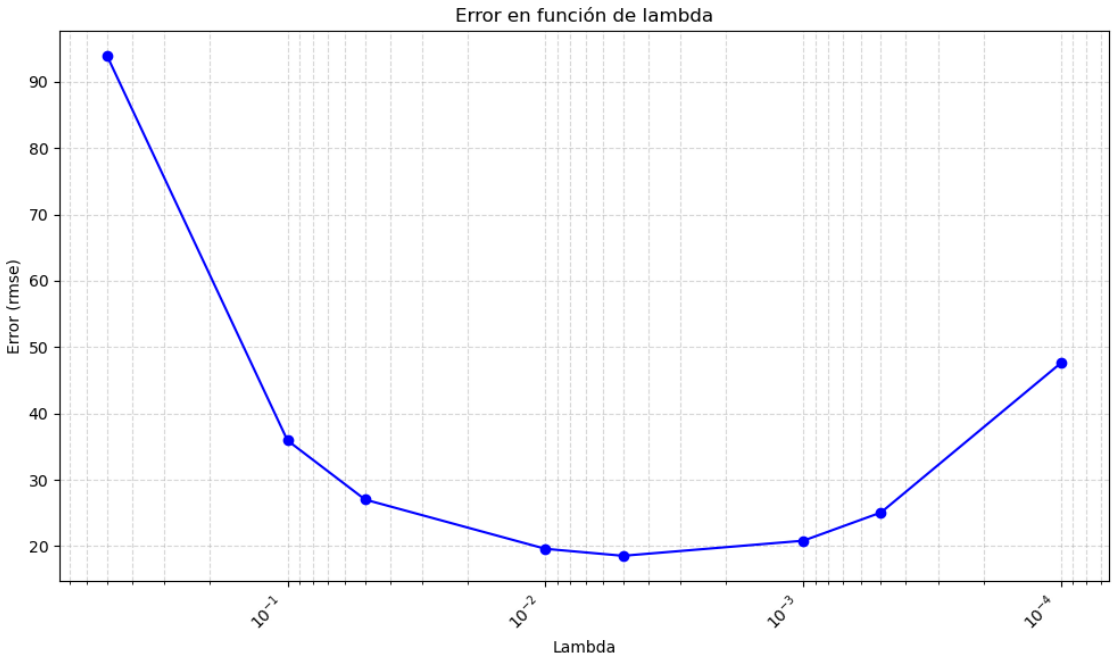

Figura 6.12: Evolución del RMSE entre la imagen original y la reconstrucción en función del parámetro lambda (Saturno con PSF2 y ruido gaussiano (σ=2.3)).

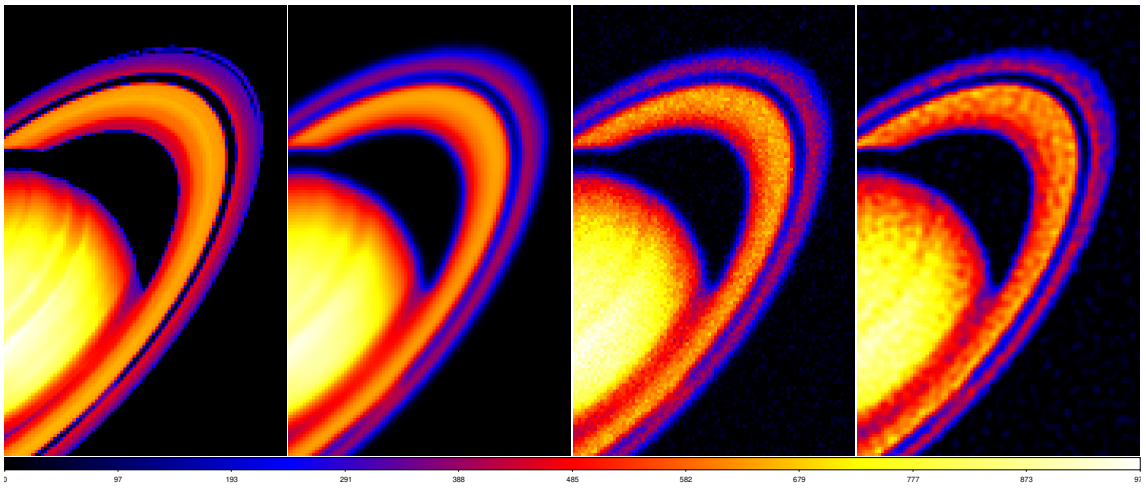

Figura 6.13: Evaluación del algoritmo de Tikhonov en una imagen de Saturno utilizando la PSF2 y ruido gaussiano con una desviación estándar de 23.1. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido, reconstrucción óptima ($\lambda = 5 \times 10^{-2}$).

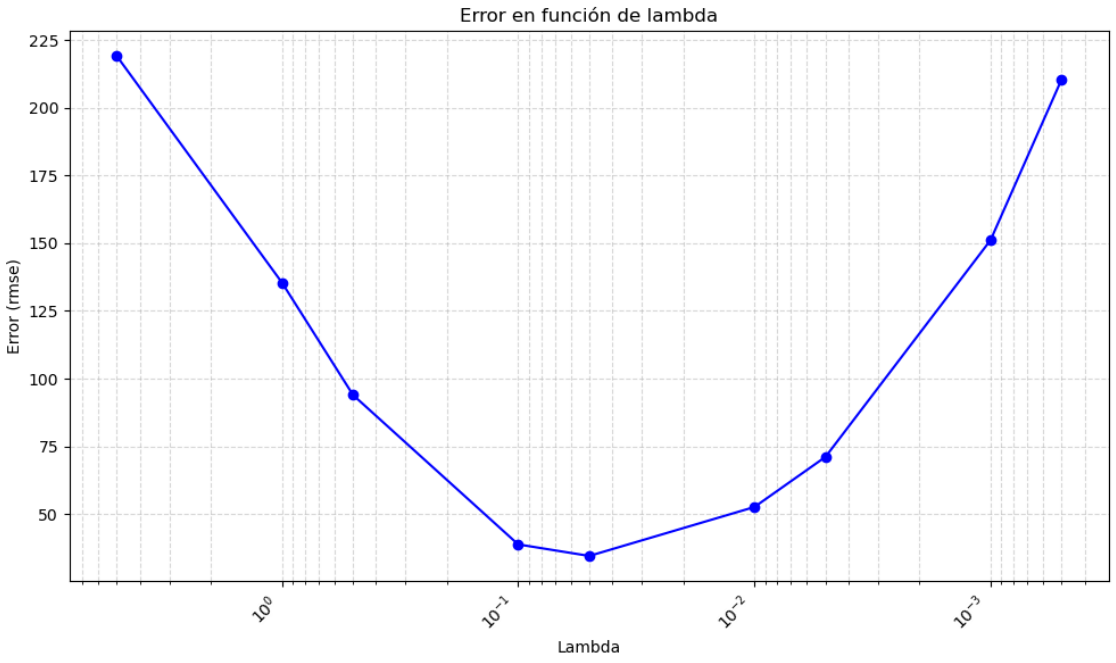

Figura 6.14: Evolución del RMSE entre la imagen original y la reconstrucción en función del parámetro lambda (Saturno con PSF2 y ruido gaussiano (σ=23.1)).

Imagen de M100

El proceso para evaluar la imagen de M100 será el mismo que el utilizado para la imagen de Saturno. Se comenzará con la PSF1 y el nivel de ruido más bajo ($\sigma = 6.54$). En la figura 6.15 puede observarse que, al tratarse del caso más sencillo, la reconstrucción es muy buena. La evolución del RMSE en función de λ se muestra en la figura 6.16.

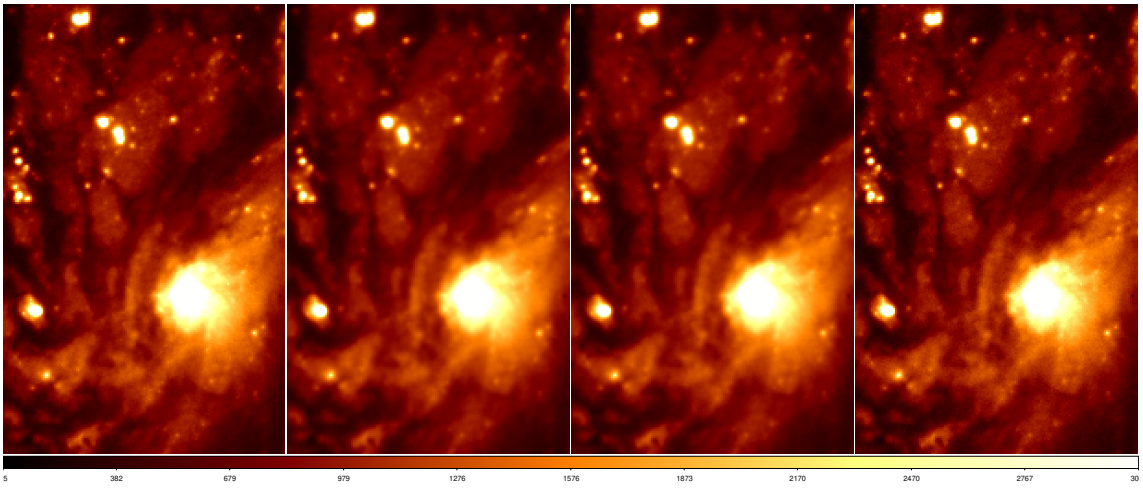

Figura 6.15: Evaluación del algoritmo de Tikhonov en una imagen de M100 utilizando la PSF1 y ruido gaussiano con una desviación estándar de 6.54. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido, reconstrucción óptima ($\lambda = 10^{-2}$).

Aumentar el ruido gaussiano hasta $\sigma = 65.42$ implica una peor reconstrucción, como puede verse en la figura 6.17. En la reconstrucción se observa que no todo el ruido ha sido eliminado, a diferencia del caso anterior. En la figura 6.18 se observa que la evolución del RMSE sigue la misma tendencia que en el experimento anterior, pero con niveles de error mayores y con un mínimo en un valor de λ diferente.

Para continuar con la evaluación, se aplicará el algoritmo de Tikhonov a la imagen de M100 utilizando la PSF2 , que introduce un mayor nivel de distorsión en comparación con la primera. En primer lugar, se añadirá un bajo nivel de ruido gaussiano, equivalente al 0.01 % de la desviación estándar de la imagen original (6.33). En la figura 6.19 se puede observar la comparativa de imágenes, donde se ve claramente una mejora entre la imagen distorsionada y la reconstrucción. Por su parte, en la figura 6.20 se muestra cómo la evolución del RMSE empieza a crecer considerablemente a partir del valor óptimo de λ.

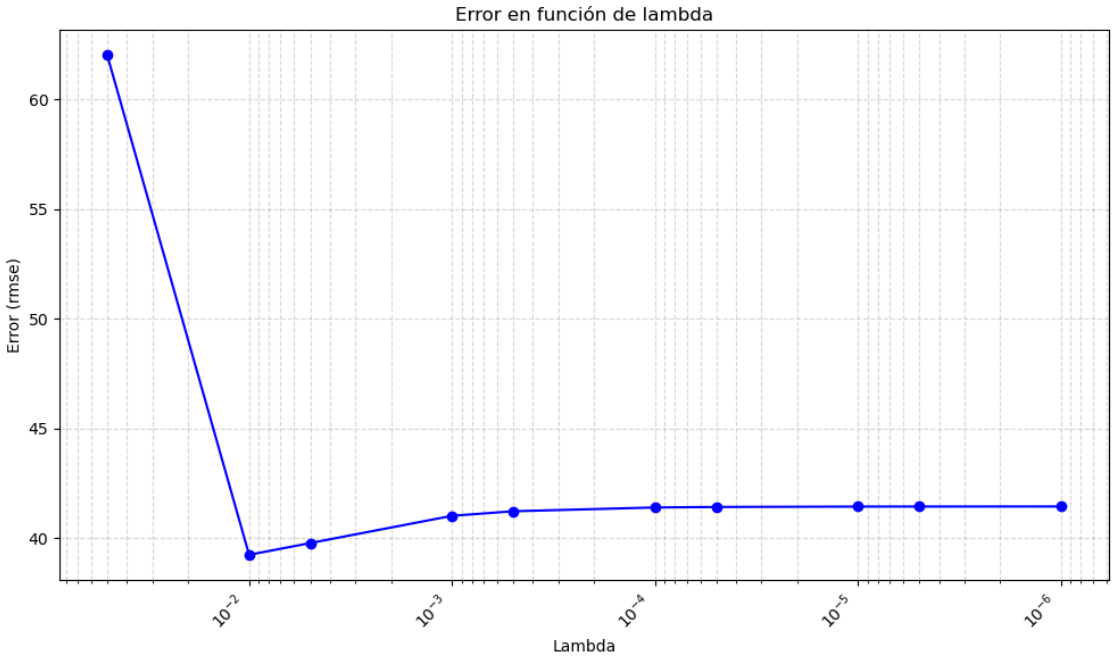

Figura 6.16: Evolución del RMSE entre la imagen original y la reconstrucción en función del parámetro lambda (M100 con PSF1 y ruido gaussiano (σ=6.54)).

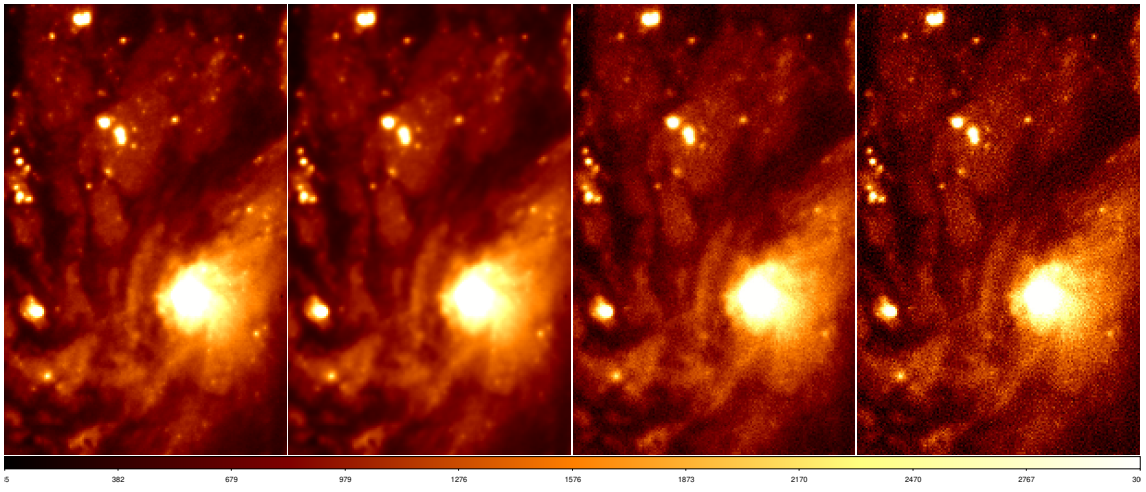

Figura 6.17: Evaluación del algoritmo de Tikhonov en una imagen de M100 utilizando la PSF1 y ruido gaussiano con una desviación estándar de 65.42. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido, reconstrucción óptima ($\lambda = 5 \times 10^{-2}$).

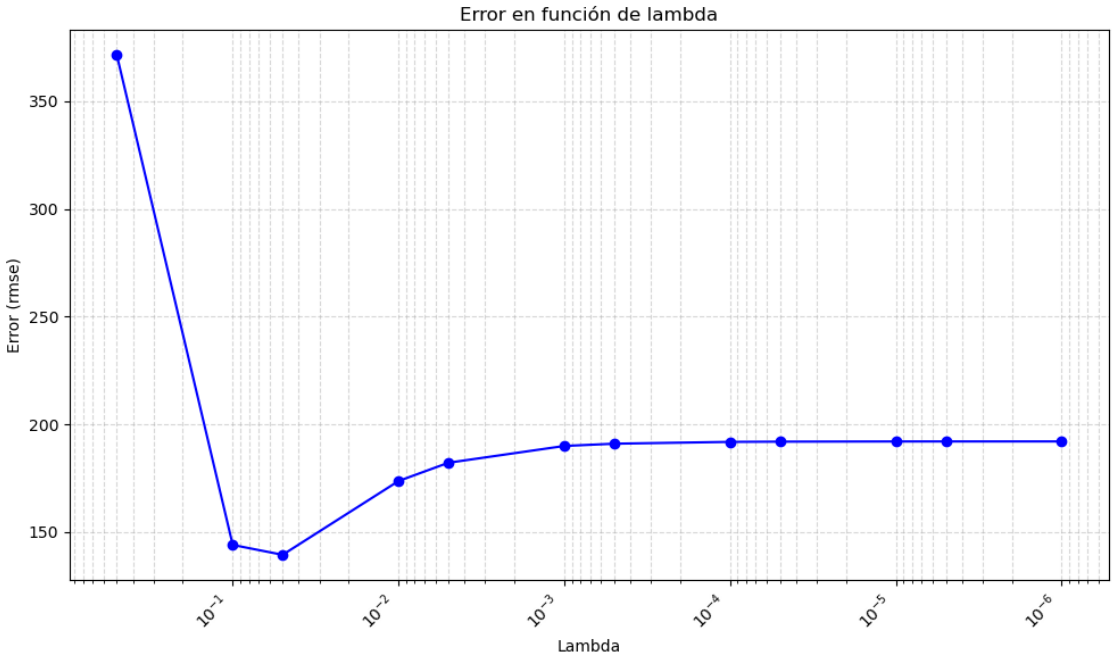

Figura 6.18: Evolución del RMSE entre la imagen original y la reconstrucción en función del parámetro lambda (M100 con PSF1 y ruido gaussiano (σ=65.42)).

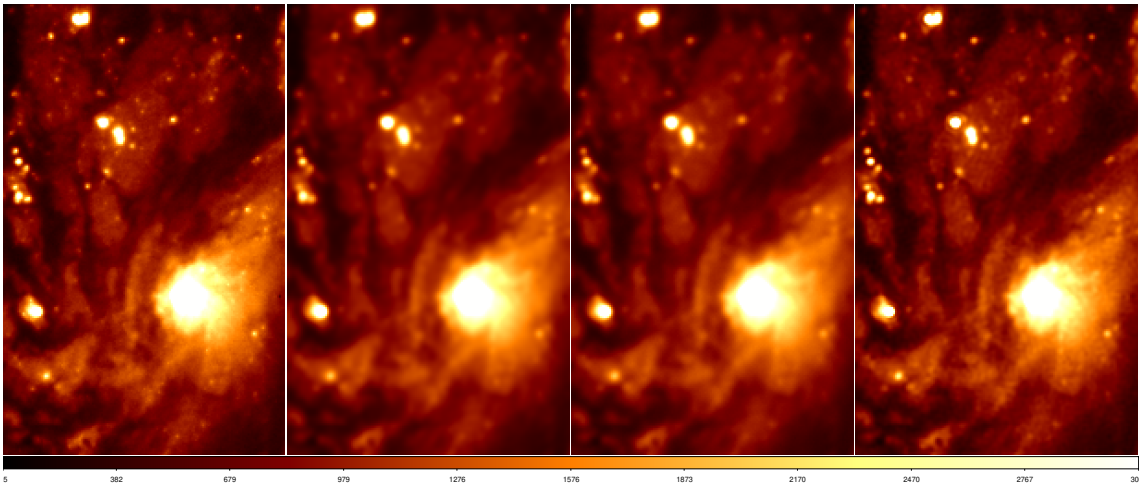

Figura 6.19: Evaluación del algoritmo de Tikhonov en una imagen de M100 utilizando la PSF2 y ruido gaussiano con una desviación estándar de 6.33. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido, reconstrucción óptima ($\lambda = 10^{-2}$).

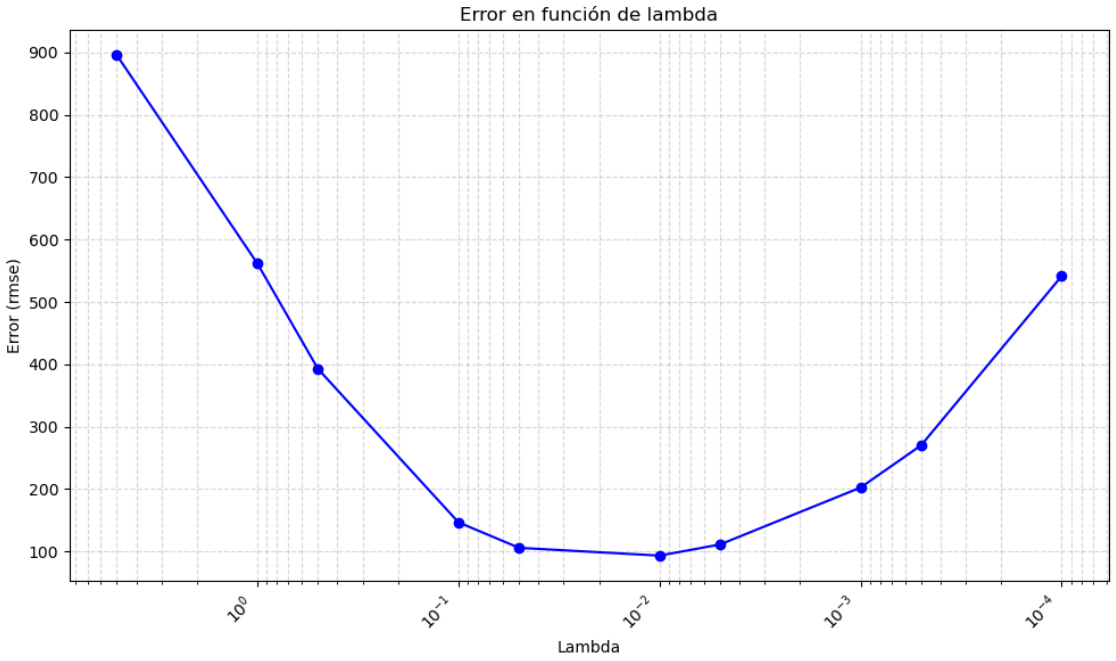

Figura 6.20: Evolución del RMSE entre la imagen original y la reconstrucción en función del parámetro lambda (M100 con PSF2 y ruido gaussiano (σ=6.33)).

Como último experimento para evaluar el algoritmo de Tikhonov, se aumentará el ruido gaussiano hasta un 0.1 % de la desviación estándar de la imagen distorsionada (63.26). En la figura 6.21 se puede observar que, aunque el ruido ha aumentado en la imagen, parte de la distorsión ha sido reducida. La figura 6.22 muestra la evolución del RMSE para este caso, siguiendo un patrón similar a los experimentos anteriores con alta distorsión.

Con estos experimentos, se puede concluir que el algoritmo de Tikhonov funciona bien para casos sencillos, logrando reducir la distorsión en imágenes con bajo nivel de ruido. Además, se ha observado que el parámetro λ óptimo varía según las características de cada caso, lo que resalta la importancia de su ajuste adecuado para obtener resultados óptimos.

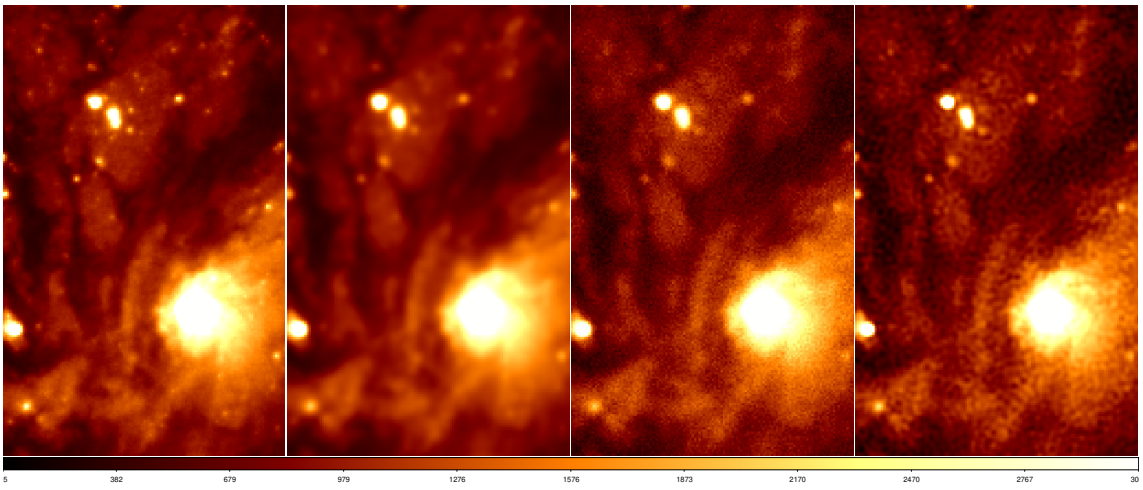

Figura 6.21: Evaluación del algoritmo de Tikhonov en una imagen de M100 utilizando la PSF2 y ruido gaussiano con una desviación estándar de 63.26. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido, reconstrucción óptima ($\lambda = 5 \times 10^{-2}$).

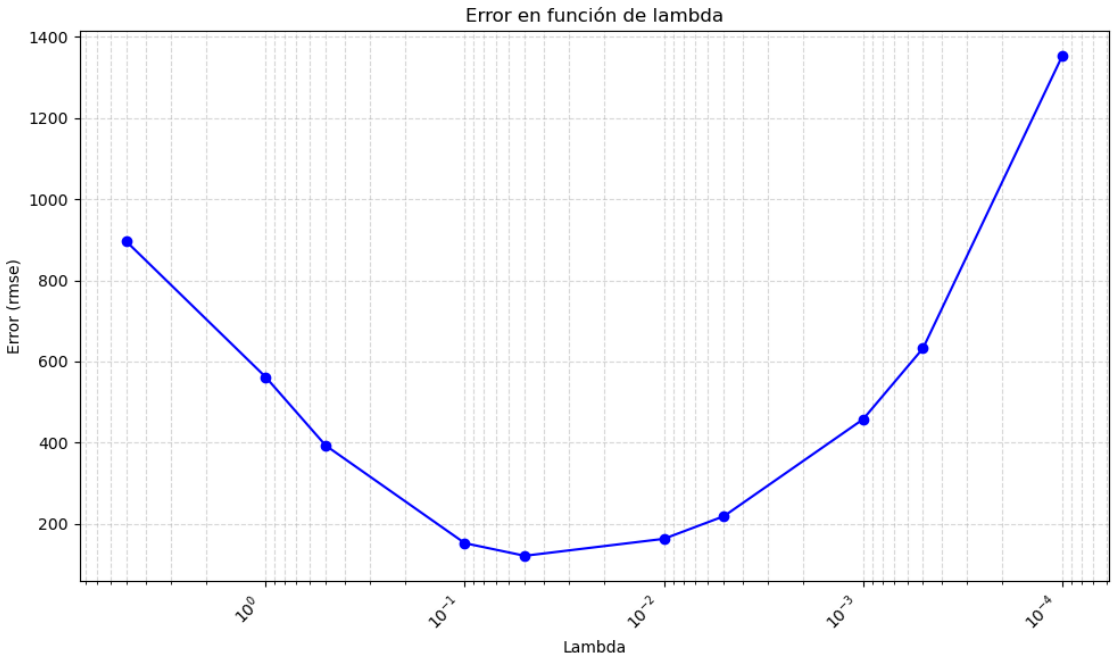

Figura 6.22: Evolución del RMSE entre la imagen original y la reconstrucción en función del parámetro lambda (M100 con PSF2 y ruido gaussiano (σ=63.26)).

6.1.3. Evaluación del algoritmo Richardson-Lucy

El algoritmo Richardson-Lucy está diseñado para manejar imágenes afectadas por ruido de Poisson, lo que lo hace especialmente útil en contextos donde la intensidad de los píxeles sigue una distribución Poissoniana, como en imágenes astronómicas. A diferencia del ruido gaussiano, el ruido de Poisson no se puede ajustar directamente, ya que depende de la intensidad de cada píxel.

En los experimentos que se presentan a continuación, se evaluará la eficacia del algoritmo Richardson-Lucy en la reconstrucción de imágenes con diferentes niveles de distorsión y ruido de Poisson. Como en el resto de los casos, se utilizarán las imágenes de Saturno y M100 junto con las dos PSF, lo que dará un total de cuatro experimentos. Se analizará la calidad de las imágenes reconstruidas y se observará cómo evoluciona el algoritmo a través de múltiples iteraciones, evaluando su capacidad para mejorar la calidad de las imágenes distorsionadas bajo estas condiciones. El parámetro alpha (α) se fijará a 1, evaluando solo el efecto del número de iteraciones.

Imagen de Saturno

Para el caso de la imagen de Saturno, se puede ver como el efecto del ruido de Poisson es bastante significativo. En la figura 6.23 se puede ver el efecto en la reconstrucción para la PSF1 y en la figura 6.24 la correspondiente evolución del RMSE con respecto al número de iteraciones. Aumentar la distorsión usando la PSF2 tiene un resultado similar al ya mencionado como se puede ver en la figura 6.25 donde se muestra la reconstrucción. Tambien se muestra la figura 6.26 donde se muestra el RMSE para este caso.

Imagen de M100

En el caso de M100 se observa como el ruido de Poisson no afecta tanto. En la figura 6.27, se observa el resultado obtenido utilizando la PSF1. Aunque el ruido no se elimina por completo, la reconstrucción final es bastante satisfactoria. En la figura 6.28, se aprecia cómo el RMSE converge rápidamente con el número de iteraciones, indicando una mejora en la calidad de la imagen a medida que avanza el proceso. Sin embargo, al aumentar la distorsión con la PSF2, la calidad del resultado disminuye en comparación con el caso anterior, como se muestra en la figura 6.29. A pesar de esto, la evolución del RMSE sigue una tendencia similar, como se puede observar en la figura 6.30.

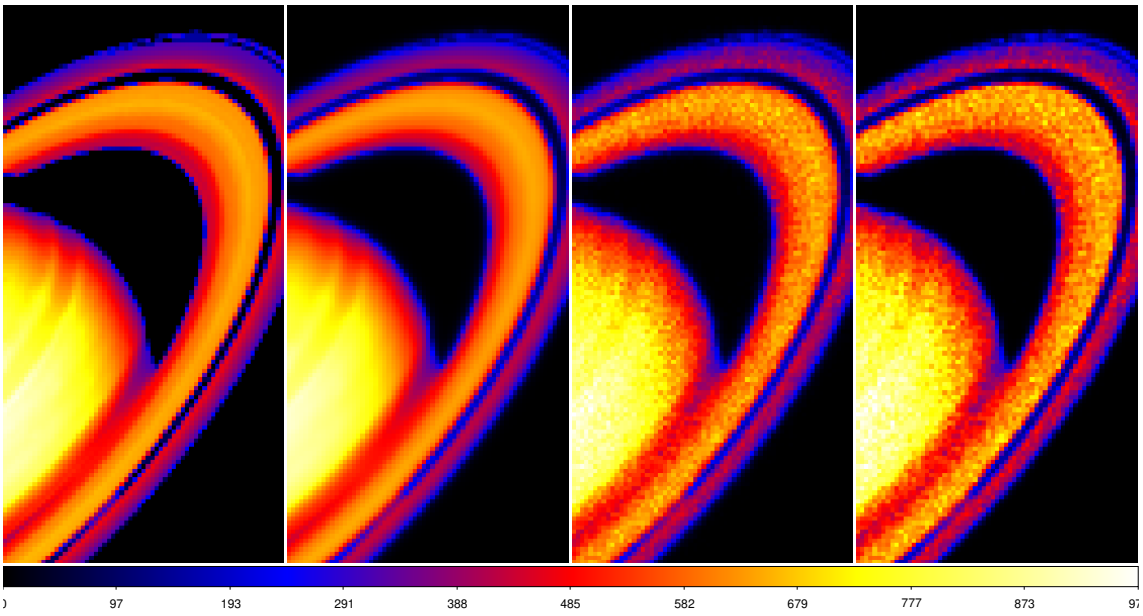

Figura 6.23: Evaluación del algoritmo de Richardson-Lucy en una imagen de Saturno con la PSF1 y ruido de Poisson. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido y reconstrucción.

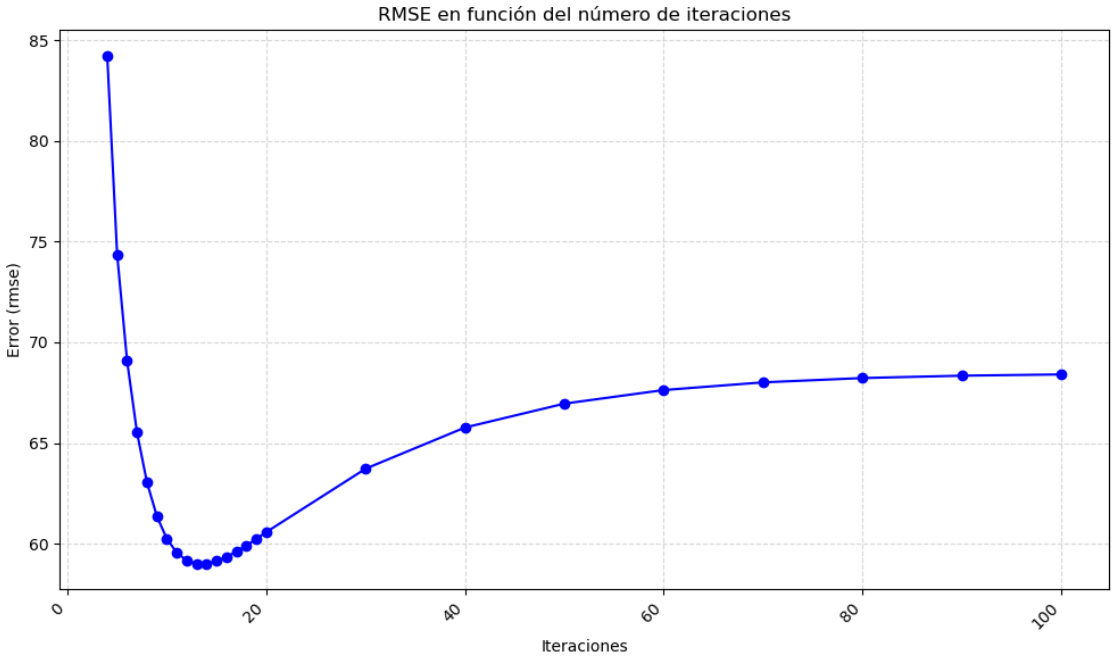

Figura 6.24: Evolución del RMSE entre la imagen original y la reconstrucción en función del número de iteraciones (imagen de Saturno con la PSF1 y ruido de Poisson) para el algoritmo de Richardson-Lucy.

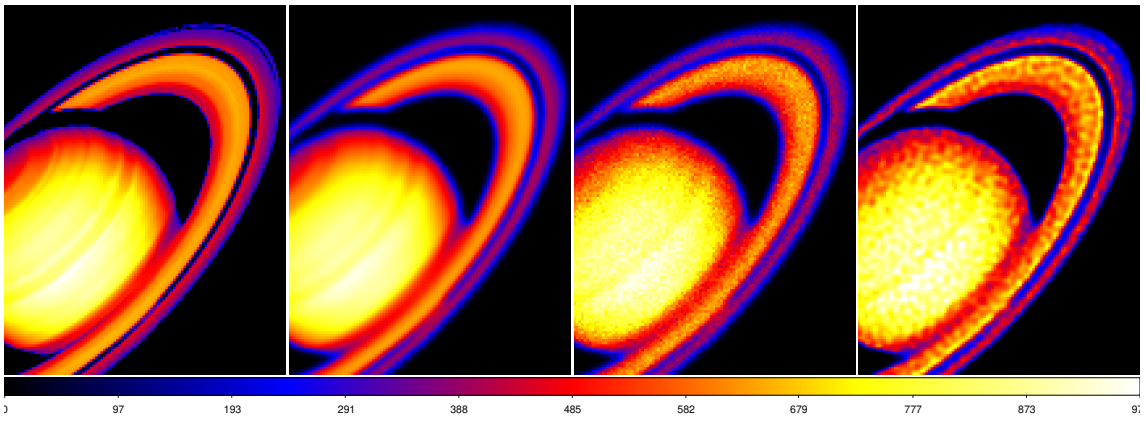

Figura 6.25: Evaluación del algoritmo de Richardson-Lucy en una imagen de Saturno con la PSF2 y ruido de Poisson. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido y reconstrucción.

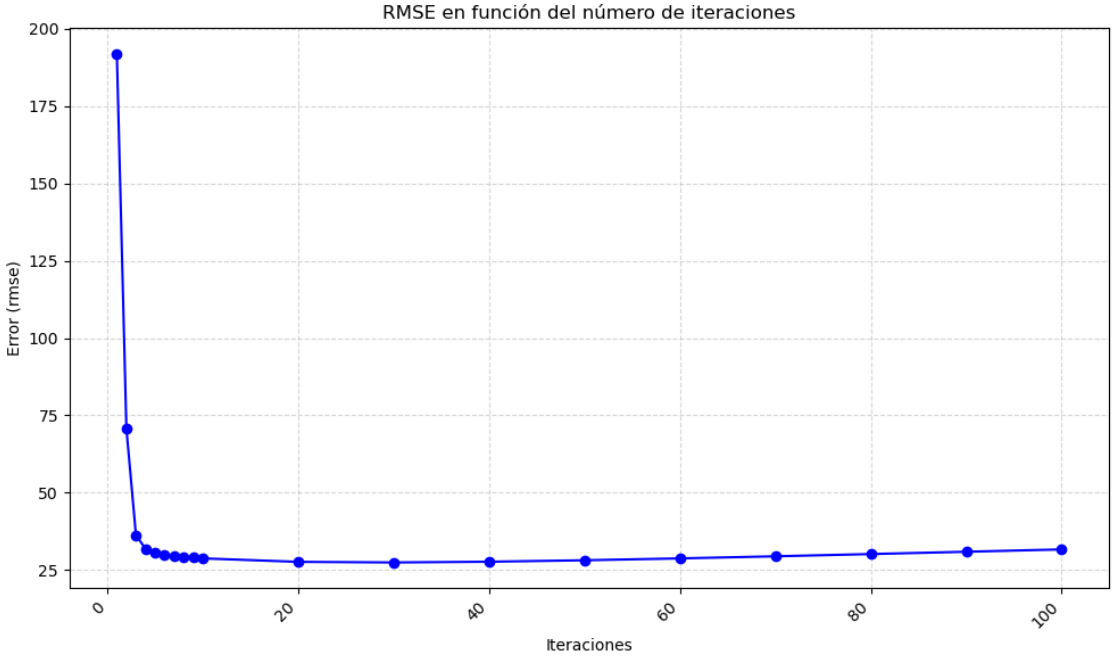

Figura 6.26: Evolución del RMSE entre la imagen original y la reconstrucción en función del número de iteraciones (imagen de Saturno con la PSF2 y ruido de Poisson) para el algoritmo de Richardson-Lucy.

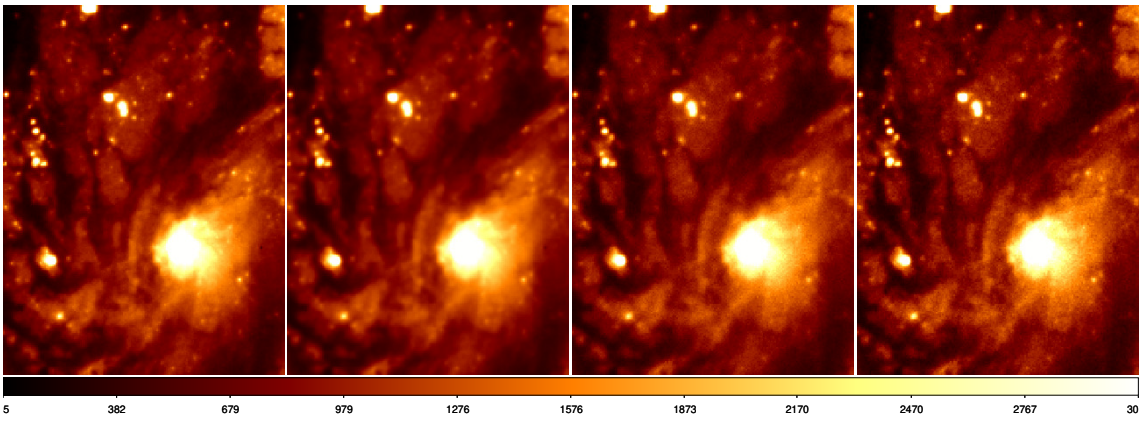

Figura 6.27: Evaluación del algoritmo de Richardson-Lucy en una imagen de M100 con la PSF1 y ruido de Poisson. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido y reconstrucción.

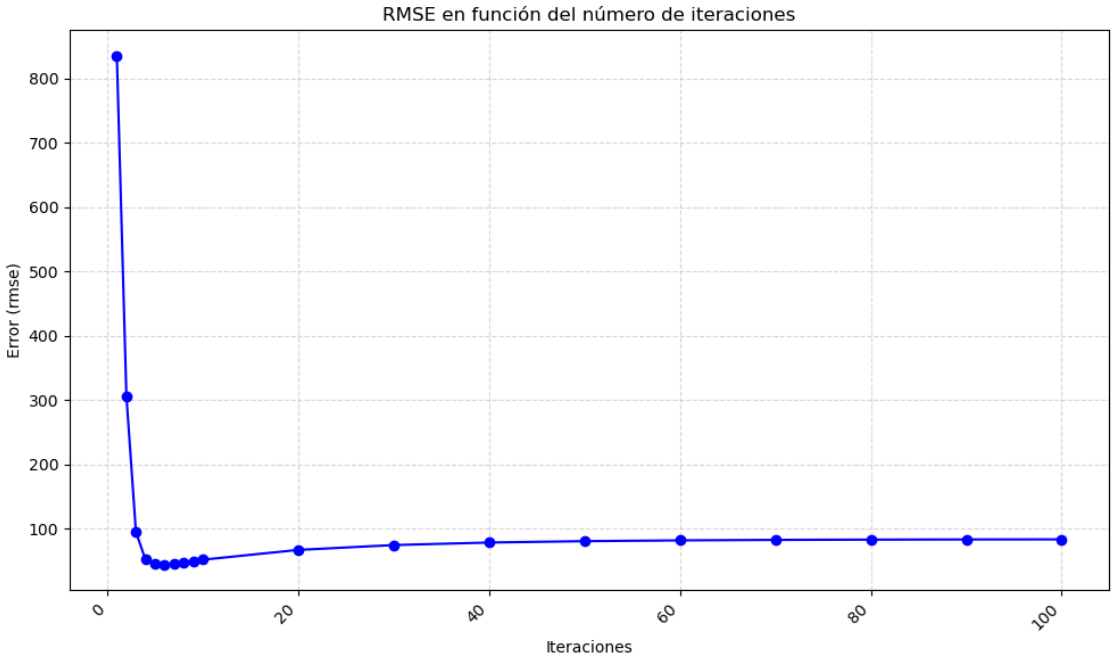

Figura 6.28: Evolución del RMSE entre la imagen original y la reconstrucción en función del número de iteraciones (imagen de M100 con la PSF1 y ruido de Poisson) para el algoritmo de Richardson-Lucy.

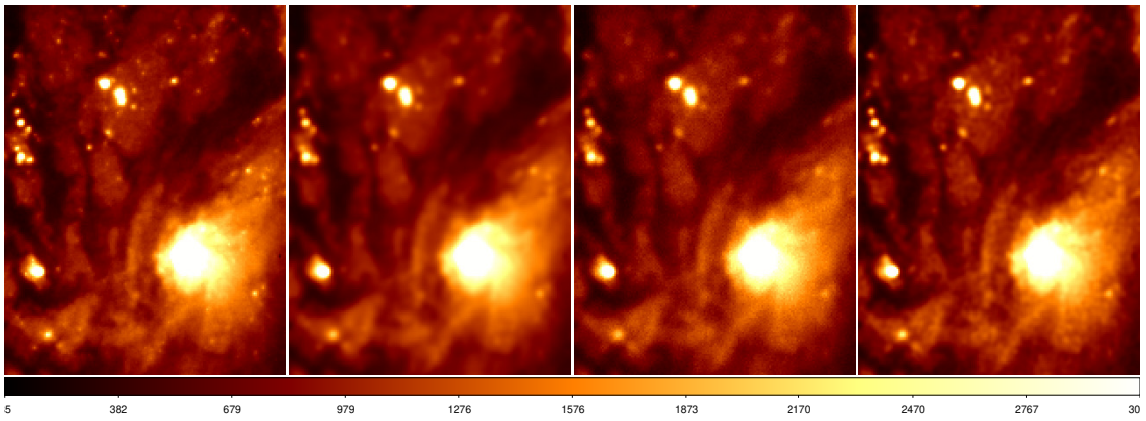

Figura 6.29: Evaluación del algoritmo de Richardson-Lucy en una imagen de M100 con la PSF2 y ruido de Poisson. De izquierda a derecha: Imagen original, imagen distorsionada, imagen distorsionada con ruido y reconstrucción.

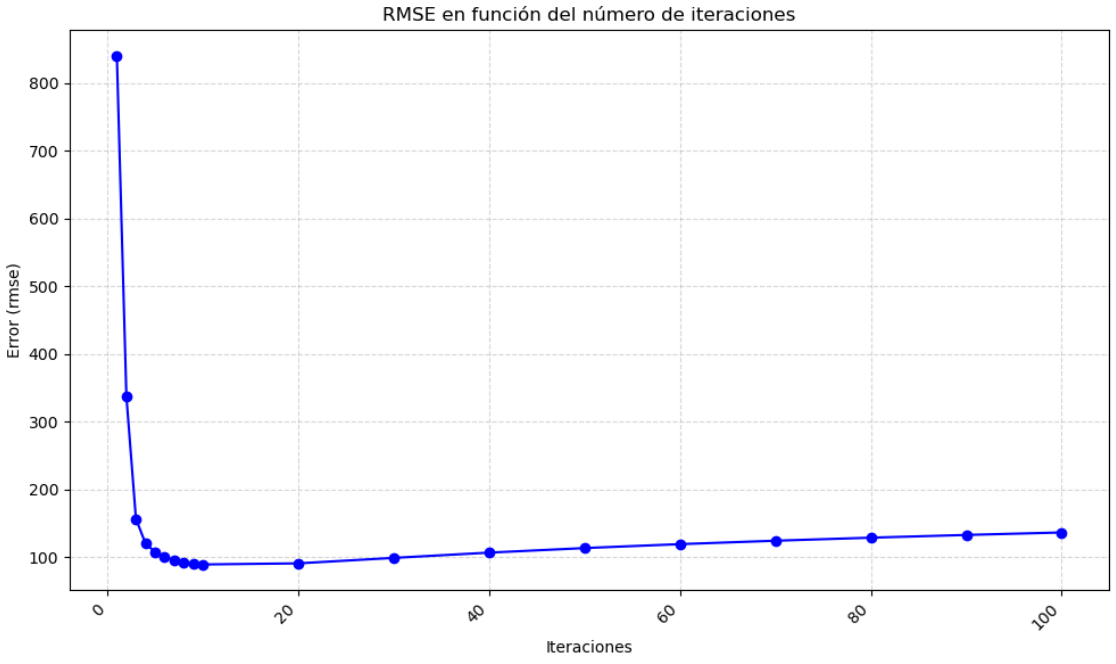

Figura 6.30: Evolución del RMSE entre la imagen original y la reconstrucción en función del número de iteraciones (imagen de Satruno con la PSF2 y ruido de Poisson) para el algoritmo de Richardson-Lucy.

6.1.4. Evaluación del algoritmo AWMLE

Por último, se va a evaluar el algoritmo más complejo de todos, AWMLE. Este algoritmo presenta más hiperparámetros que el resto, pero para simplificar su evaluación, se fijará el número de planos en 4, la tolerancia será suficientemente baja para que nunca se active y nunca pare la deconvolución, y σ se igualará a la desviación estándar del ruido gaussiano añadido. El parámetro alpha (α) se fijará a uno, ya que este parámetro solo afecta a la velocidad de convergencia, pero no al resultado en sí. De esta manera, la comparación de los resultados se centrará únicamente en función del número de iteraciones.

El algoritmo AWMLE está diseñado para manejar tanto ruido gaussiano como de Poisson. El componente de Poisson depende de la intensidad de cada píxel, por lo que no es ajustable, a diferencia del ruido gaussiano, cuya cantidad se ajustará al 0.1 % y al 0.01 % de la desviación estándar de la imagen distorsionada. Se realizarán experimentos con las dos imágenes (Saturno y M100), utilizando ambas PSF y ambos niveles de ruido gaussiano, lo que dará un total de 8 experimentos. Todos los experimentos incluirán ruido de Poisson.

Imagen de Saturno

El primer experimento se realiza utilizando la imagen de Saturno con la PSF1, que introduce un nivel bajo de distorsión, y se añade un bajo nivel de ruido gaussiano equivalente al 0.01 % de la desviación estándar de la imagen distorsionada (2.31) junto al ruido de Poisson. El objetivo es observar cómo el algoritmo AWMLE maneja estas condiciones relativamente sencillas. En la figura 6.31, se presenta la comparación entre la imagen original, la imagen distorsionada con ruido añadido, y las imágenes reconstruidas tras aplicar el algoritmo con 3 y 8 iteraciones. En este caso, puede ser interesante utilizar una imagen que no está en el valor óptimo de RMSE, ya que a cambio de tener algo más de ruido, muestra menor distorsión. En la figura 6.32, se puede observar la evolución del RMSE en función del número de iteraciones, destacando cómo el algoritmo mejora la calidad de la reconstrucción, hasta un punto en el que empieza a aumentar.

Para el caso de la imagen de Saturno con la primera PSF y un nivel más alto de ruido gaussiano, 0.1 % de la desviación estándar de la imagen original (23.16), además del ruido de Poisson, se observan resultados consistentes con una reducción progresiva del RMSE a medida que aumentan las iteraciones del algoritmo AWMLE como puede verse en la figura

6.34. Sin embargo, aunque el RMSE es decreciente, se encontró que la mejor reconstrucción visual se obtiene con 25 iteraciones, ya que esta imagen presenta un mejor balance entre reducción de distorsión y ruido, a pesar de que no coincida exactamente con el mínimo valor del RMSE. En la figura 6.33 se muestra la comparativa para este experimento.

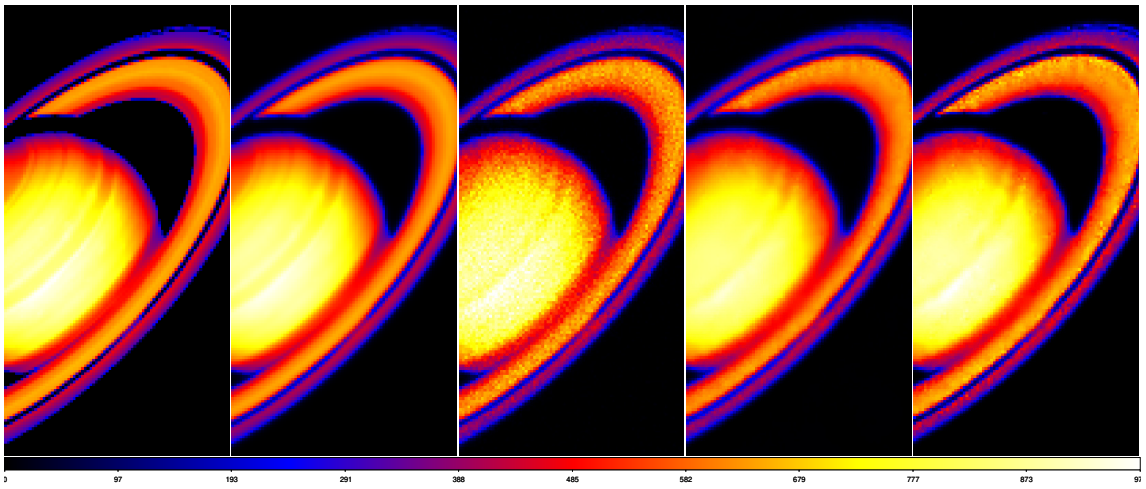

Figura 6.31: Comparación de la imagen de Saturno utilizando la PSF1 con ruido gaussiano (σ=2.31) y ruido de Poisson. De izquierda a derecha: imagen original, imagen distorsionada, imagen distorsionada con ruido añadido y reconstrucciones utilizando el algoritmo AWMLE con 3 y 8 iteraciones.

En el siguiente experimento, se utilizará la imagen de Saturno con la PSF2, que introduce un mayor nivel de distorsión en comparación con la primera, junto con un bajo nivel de ruido gaussiano equivalente al 0.01 % de la desviación estándar de la imagen original (2.29) y ruido de Poisson. En la figura 6.35 se presenta la comparación entre la imagen original, la imagen distorsionada con ruido añadido, y las imágenes reconstruidas tras aplicar el algoritmo AWMLE con 18 iteraciones. Como se observa, el algoritmo logra mejorar significativamente la calidad de la imagen reconstruida, aunque la distorsión inicial es mayor que en experimentos anteriores. La figura 6.36 muestra la evolución del RMSE en función del número de iteraciones, reflejando una tendencia decreciente del error hasta alcanzar un punto óptimo en el que se obtiene la mejor reconstrucción visual. A partir de ese punto, el RMSE comienza a aumentar.

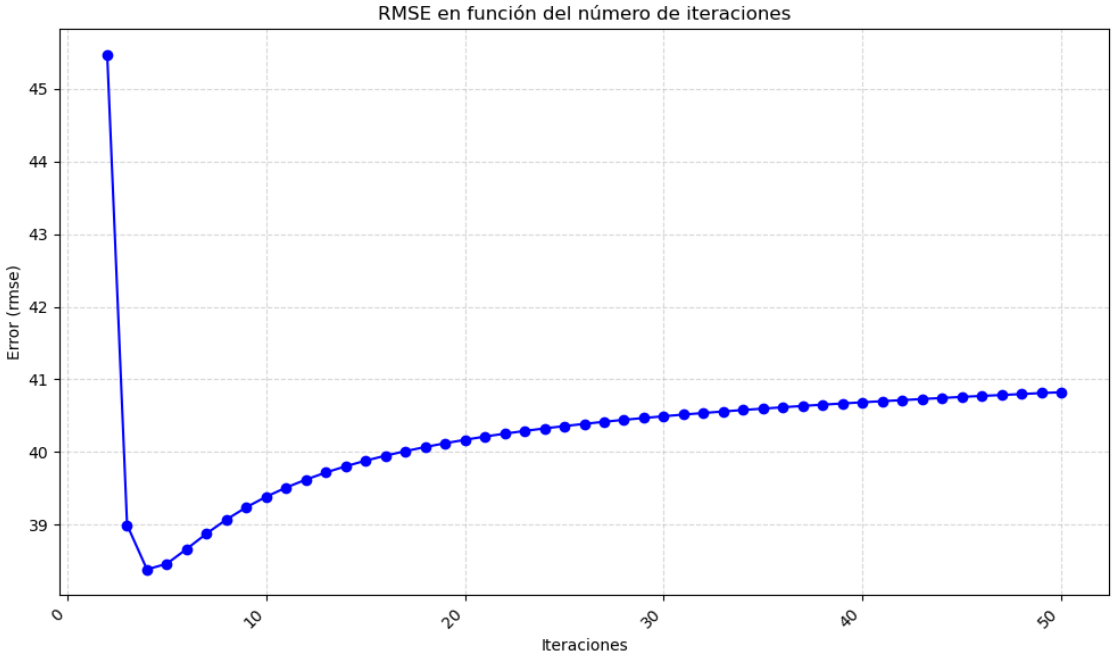

Figura 6.32: Evolución del RMSE en función del número de iteraciones del algoritmo AWMLE para la imagen de Saturno con la PSF1, ruido gaussiano (σ=2.31) y ruido de Poisson.

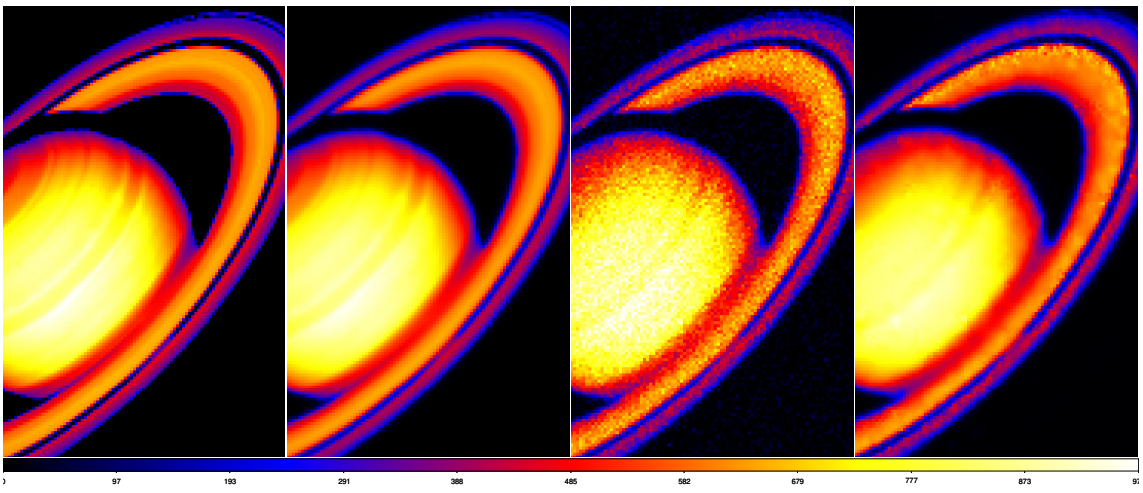

Figura 6.33: Comparación de la imagen de Saturno utilizando la PSF1 con ruido gaussiano (σ=23.16) y ruido de Poisson. De izquierda a derecha: imagen original, imagen distorsionada, imagen distorsionada con ruido añadido y reconstrucción utilizando el algoritmo AWMLE con 25 iteraciones.

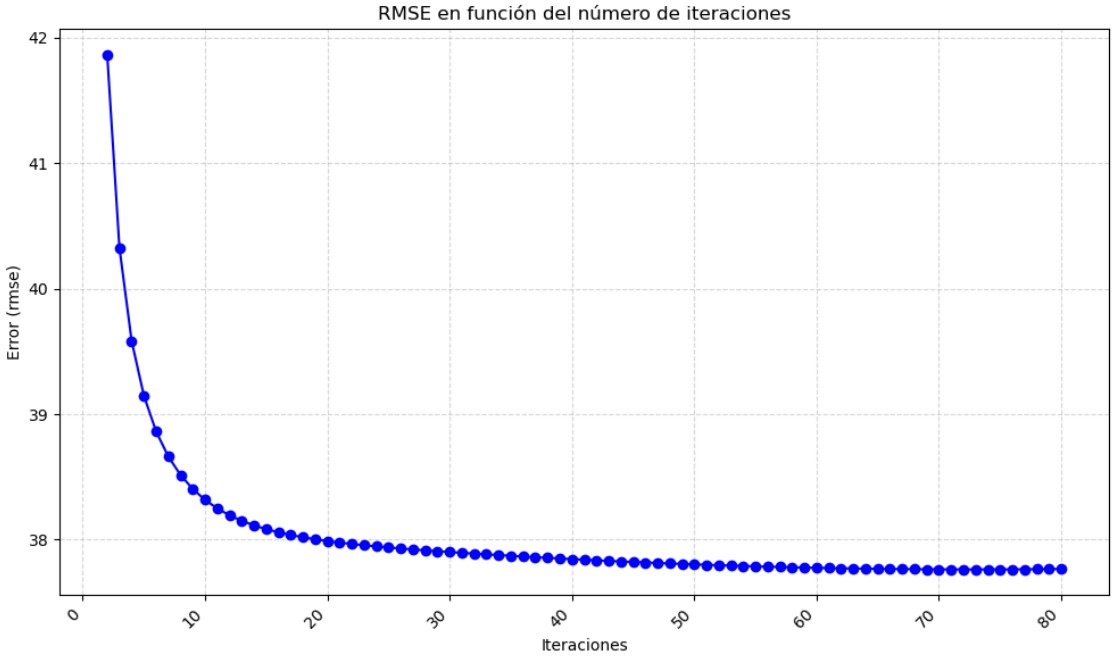

Figura 6.34: Evolución del RMSE en función del número de iteraciones del algoritmo AWMLE para la imagen de Saturno con la PSF1, ruido gaussiano (σ=23.16) y ruido de Poisson.

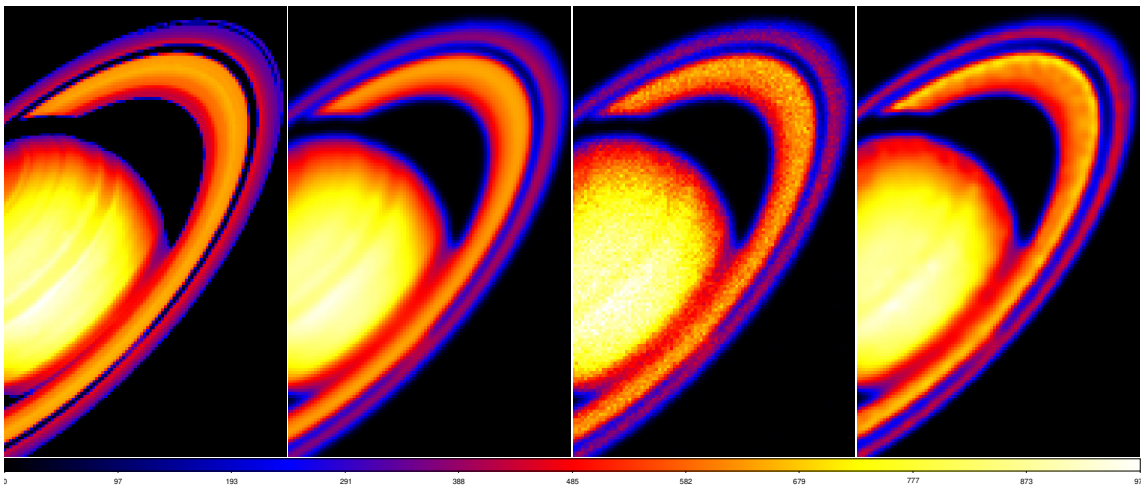

Figura 6.35: Comparación de la imagen de Saturno utilizando la PSF2 con ruido gaussiano ($\sigma = 2.29$) y ruido de Poisson. De izquierda a derecha: imagen original, imagen distorsionada, imagen distorsionada con ruido añadido y reconstrucción utilizando el algoritmo AWMLE con 18 iteraciones.

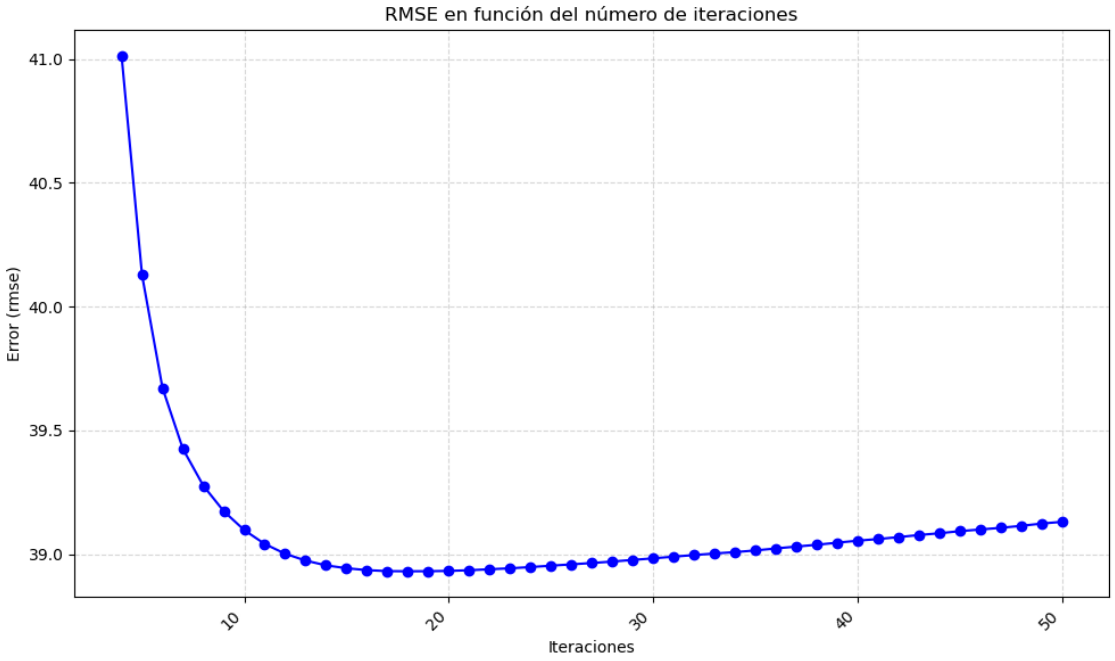

Figura 6.36: Evolución del RMSE en función del número de iteraciones del algoritmo AWMLE para la imagen de Saturno con la PSF2, ruido gaussiano ($\sigma = 2.29$) y ruido de Poisson.

En el último experimento con la imagen de Saturno, se utilizará la PSF2, que introduce un mayor nivel de distorsión, junto con un nivel alto de ruido gaussiano equivalente al 0.1 % de la desviación estándar de la imagen original (22.95) más el ruido de Poisson. En este caso, los resultados obtenidos no son tan buenos como en los experimentos anteriores. Aunque el algoritmo AWMLE logra reducir parte del ruido y mejorar la calidad de la imagen, la reconstrucción final no alcanza la calidad visual esperada. La figura 6.37 muestra la comparación entre las imágenes, y se puede apreciar que aún persisten elementos de ruido y distorsión. Además, en la figura 6.38, se observa que la gráfica del RMSE disminuye hasta un cierto punto, pero luego comienza a aumentar, lo que indica que un mayor número de iteraciones no necesariamente mejora la reconstrucción.

Imagen de M100

El siguiente experimento se realiza utilizando la imagen de M100 con la PSF1 junto con un bajo nivel de ruido gaussiano equivalente al 0.01 % de la desviación estándar de la imagen distorsionada (6.54) más el ruido de Poisson. En este escenario, el algoritmo AWMLE muestra un rendimiento sólido. Como se observa en la figura 6.39, la imagen

reconstruida tras aplicar el algoritmo presenta una calidad visual cercana a la original, con una notable reducción de la distorsión y el ruido. La gráfica del RMSE, mostrada en la figura 6.40, indica que el error disminuye rápidamente, alcanzando un mínimo después de 4 iteraciones.

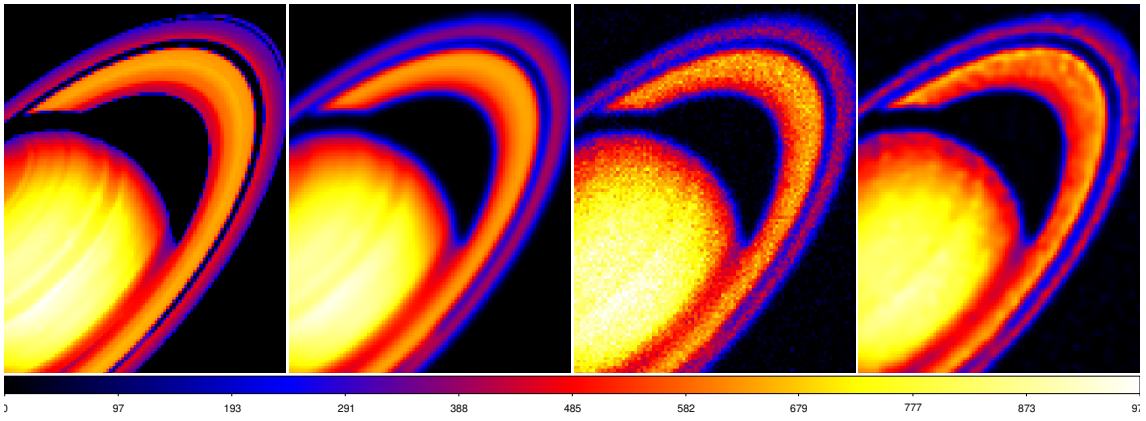

Figura 6.37: Comparación de la imagen de Saturno utilizando la PSF2 con ruido gaussiano ($\sigma = 22.95$) y ruido de Poisson. De izquierda a derecha: imagen original, imagen distorsionada, imagen distorsionada con ruido añadido y reconstrucción utilizando el algoritmo AWMLE con 9 iteraciones.

En el caso de la imagen de M100 con la PSF1 y un mayor nivel de ruido gaussiano, equivalente al 0.1 % de la desviación estándar de la imagen distorsionada ($\sigma = 65.4$) y ruido de Poission, el rendimiento del algoritmo AWMLE es significativamente peor. Como se muestra en la figura 6.41, algunos píxeles no se pueden reconstruir correctamente, lo que resulta en una imagen final con artefactos visibles y mayor distorsión. Debido a la ineficacia del algoritmo en este escenario, no se ha generado una gráfica del RMSE.

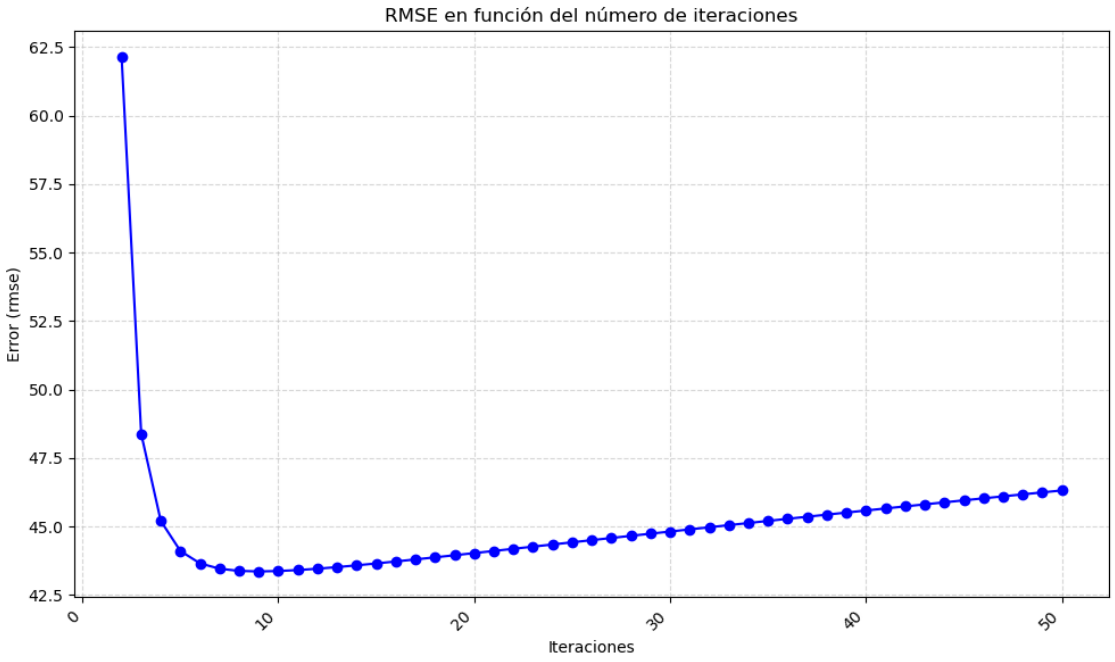

Figura 6.38: Evolución del RMSE en función del número de iteraciones del algoritmo AWMLE para la imagen de Saturno con la PSF2, ruido gaussiano ($\sigma = 22.95$) y ruido de Poisson.

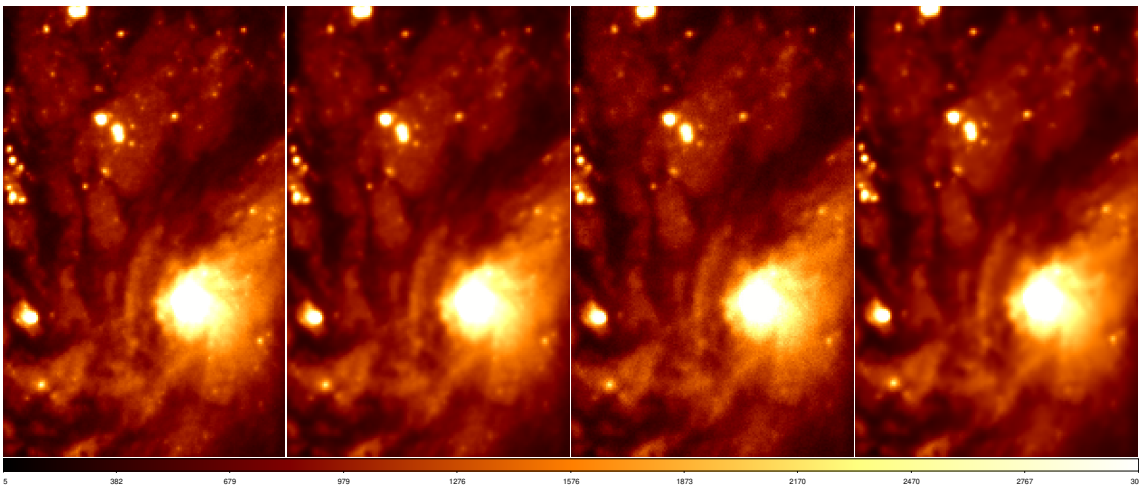

Figura 6.39: Comparación de la imagen de M100 utilizando la PSF1 con ruido gaussiano ($\sigma = 6.54$) y ruido de Poisson. De izquierda a derecha: imagen original, imagen distorsionada, imagen distorsionada con ruido añadido y reconstrucción utilizando el algoritmo AWMLE con 4 iteraciones.

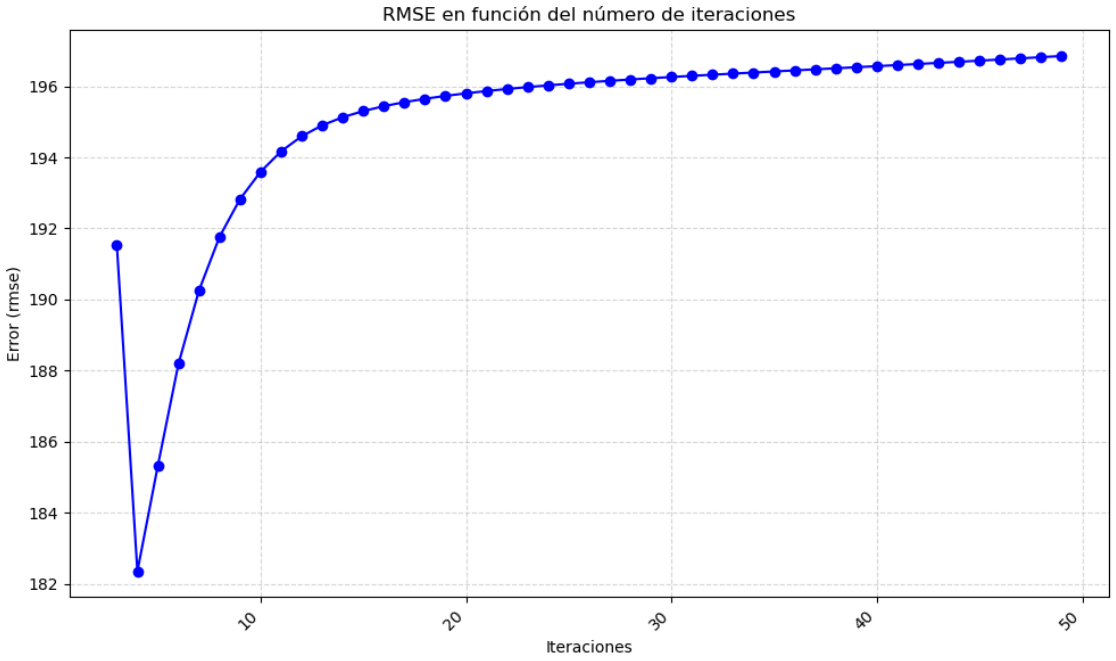

Figura 6.40: Evolución del RMSE en función del número de iteraciones del algoritmo AWMLE para la imagen de M100 con la PSF1, ruido gaussiano (σ = 22.95) y ruido de Poisson.

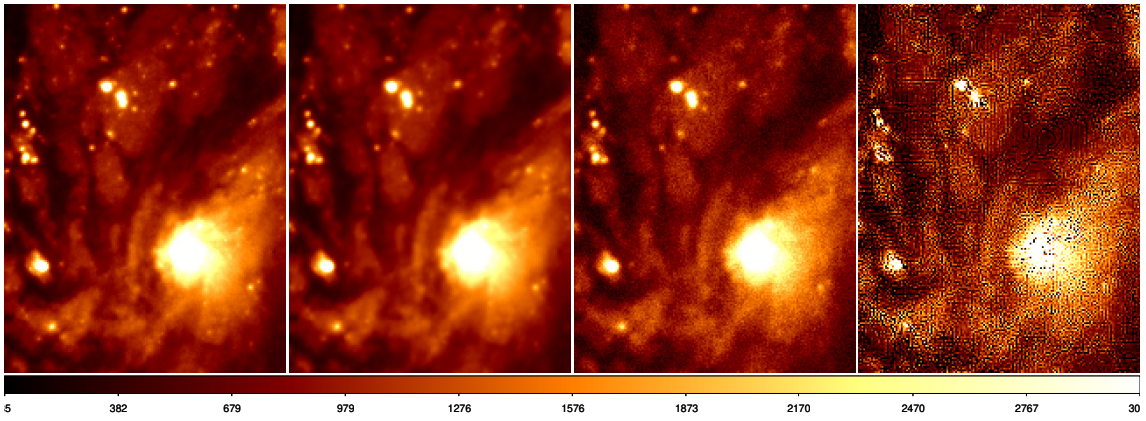

Figura 6.41: Comparación de la imagen de M100 utilizando la PSF1 con ruido gaussiano (σ = 65.4) y ruido de Poisson. De izquierda a derecha: imagen original, imagen distorsionada, imagen distorsionada con ruido añadido y reconstrucción (fallida) utilizando el algoritmo AWMLE

Para el experimento con la imagen de M100 utilizando la PSF2 y un bajo nivel de ruido gaussiano (0.01 % de la desviación estándar de la imagen distorsionada, 6.32) y ruido de Poission, se obtuvo un buen resultado. La figura 6.42 muestra que la calidad de la reconstrucción es aceptable, y la gráfica de RMSE en la figura 6.43 indica una disminución continua hasta alcanzar un mínimo en la séptima iteración. A partir de ese punto, el RMSE comienza a aumentar. Sin embargo, se utilizó la reconstrucción en 40 iteraciones ya que conseguía disminuir más la distorsión, aunque aumentase el ruido.

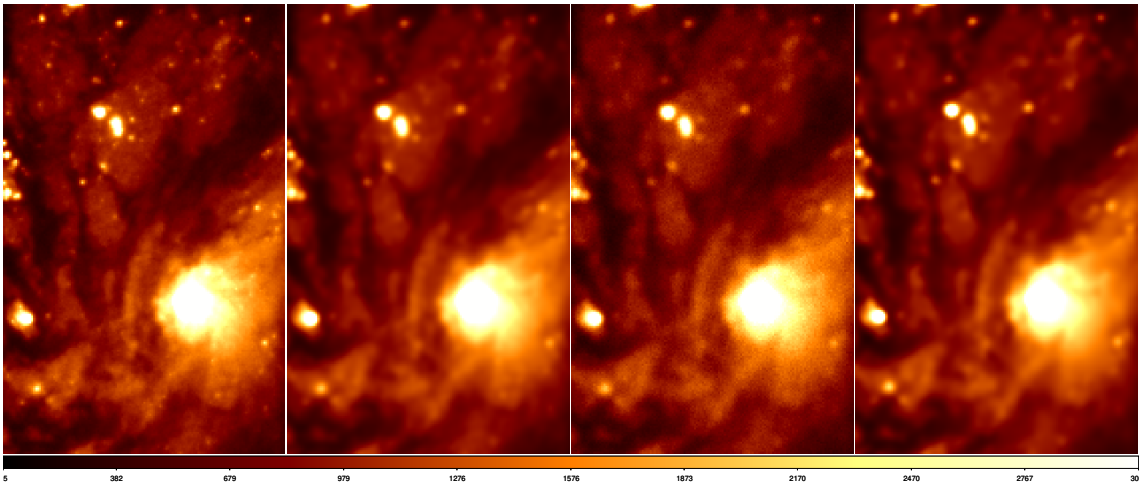

Figura 6.42: Comparación de la imagen de M100 utilizando la PSF2 con ruido gaussiano (σ = 6.32) y ruido de Poisson. De izquierda a derecha: imagen original, imagen distorsionada, imagen distorsionada con ruido añadido y reconstrucción utilizando el algoritmo AWMLE con 40 iteraciones.

En el caso de M100 utilizando la PSF2 con una mayor cantidad de ruido gaussiano (0.1 % de la desviación estándar de la imagen original, 63.2) y ruido de Poisson, el algoritmo AWMLE también mostró un buen rendimiento. A pesar del incremento en el nivel de ruido, la reconstrucción fue efectiva, como se puede ver en la figura 6.44. Como en el resto de casos, en la gráfica del RMSE (figura 6.45) se puede observar un mínimo, en este caso en 6 iteraciones a partir del cual empieza a aumentar el RMSE.

Los experimentos realizados con el algoritmo AWMLE han demostrado su efectividad en la reconstrucción de imágenes astronómicas bajo distintas condiciones de distorsión y niveles de ruido. En general, el algoritmo logra reducir significativamente la distorsión introducida por la PSF, especialmente en casos con bajo nivel de ruido. Sin embargo, el rendimiento depende del número de iteraciones y del nivel de ruido presente en la imagen.

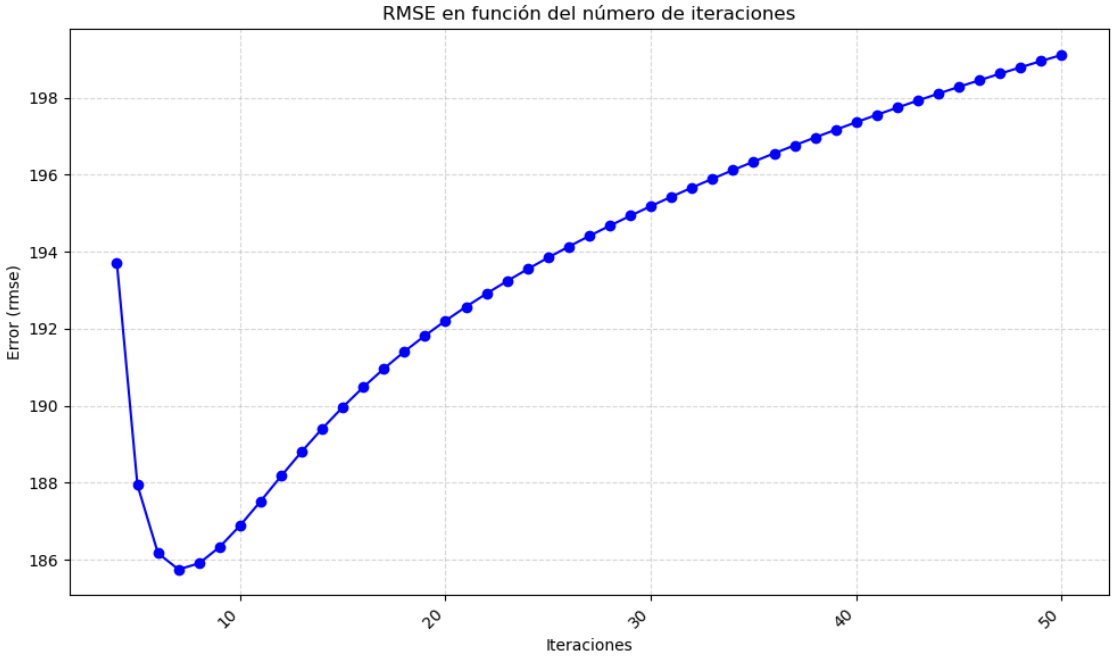

Figura 6.43: Evolución del RMSE en función del número de iteraciones del algoritmo AWMLE para la imagen de M100 con la PSF1, ruido gaussiano (σ = 6.32) y ruido de Poisson.

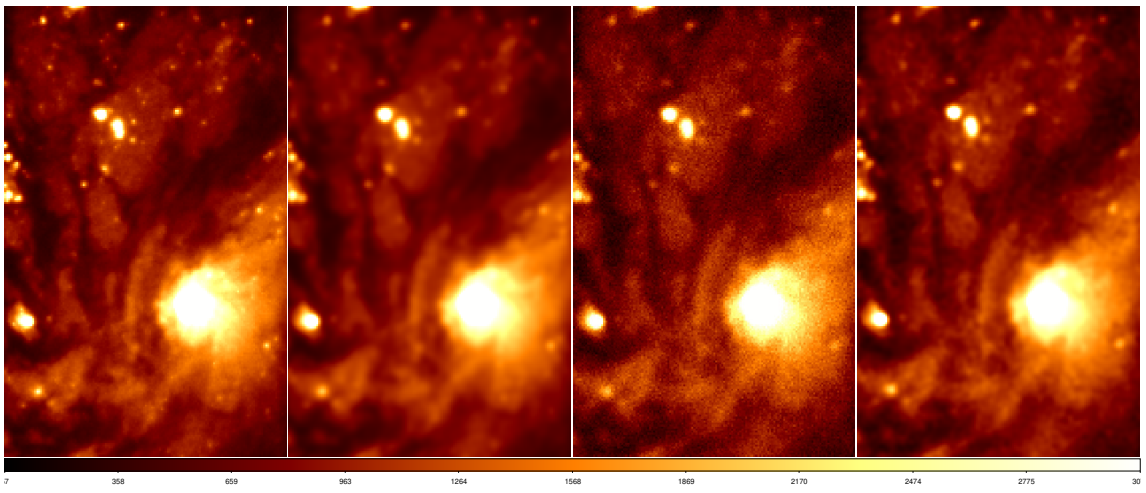

Figura 6.44: Comparación de la imagen de M100 utilizando la PSF2 con ruido gaussiano (σ = 63.2) y ruido de Poisson. De izquierda a derecha: imagen original, imagen distorsionada, imagen distorsionada con ruido añadido y reconstrucción utilizando el algoritmo AWMLE con 6 iteraciones.

En resumen, AWMLE es una herramienta útil para la deconvolución de imágenes, pero requiere una cuidadosa sintonización de sus parámetros y un análisis detallado de los resultados para maximizar su efectividad en diferentes contextos.

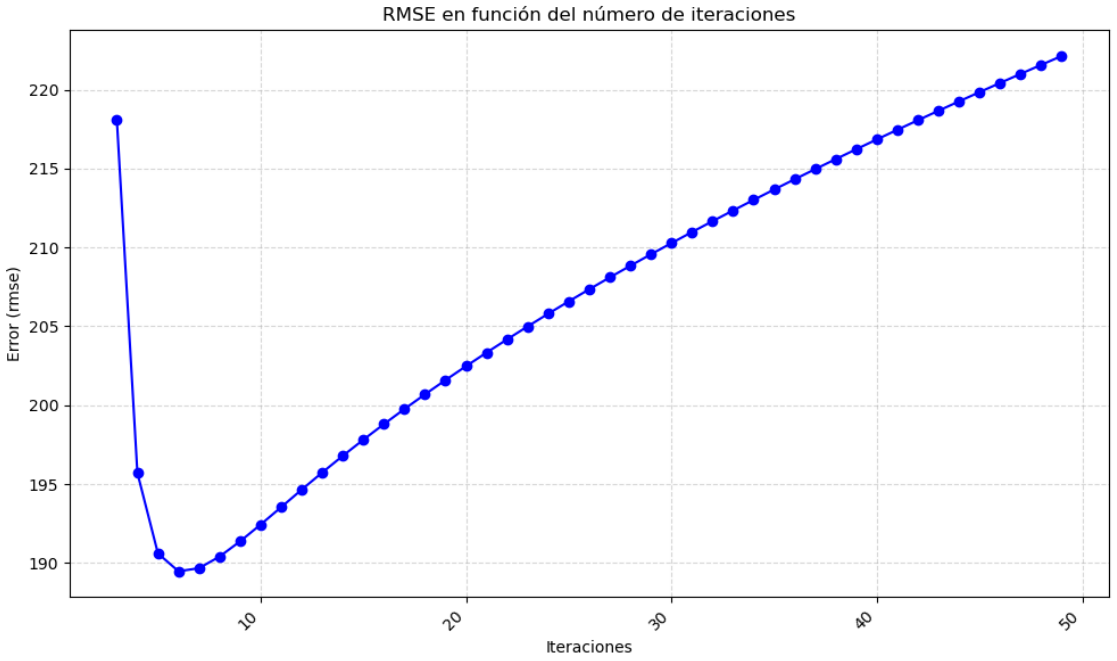

Figura 6.45: Evolución del RMSE en función del número de iteraciones del algoritmo AWMLE para la imagen de M100 con la PSF1, ruido gaussiano ($\sigma = 63.2$) y ruido de Poisson.

6.2. Comparación de algortimos con simulaciones realistas

6.2.1. Preparación de las imágenes

El objetivo de esta sección es obtener imágenes más realistas que permitan evaluar con precisión los algoritmos de deconvolución. Para ello, se detalla el procedimiento empleado en el procesamiento de imágenes generadas por las simulaciones cosmológicas de Illustris (Snyder y cols., 2017). El proceso comienza con la selección y recorte de una región específica de la imagen simulada, ajustando su orientación para asegurar su correcta alineación con los datos observacionales. Posteriormente, se aplica una convolución utilizando un kernel diseñado para emular efectos ópticos como la difracción de la luz en telescopios, lo que permite una representación más realista de las observaciones.

A continuación, las imágenes se reescalan a una resolución definida por una escala de píxeles específica, garantizando que la resolución sea adecuada para los análisis posteriores. Finalmente, se introduce ruido en la imagen según un límite de brillo superficial preestablecido, simulando las condiciones observacionales reales y permitiendo un análisis más realista. Las imágenes resultantes de este proceso se emplearán en estudios comparativos y análisis cuantitativos en astrofísica, facilitando su integración con datos observacionales reales y mejorando la precisión de los resultados obtenidos.

Durante los diferentes experimentos, se van a variar dos parámetros principales: el brillo superficial y la distorsión del kernel. Otras variables, como el centro y el tamaño del recorte o la resolución, permanecerán constantes. A continuación, se describen estas variables y sus unidades:

- **Ancho (*Width)***: Ancho del recorte en unidades de arco. Esta variable define el tamaño del recorte aplicado a la imagen y se mide en minutos de arco. Un grado equivale a 60 minutos de arco (1 hora).

- **Centro (*Center*)**: Coordenadas del centro del recorte en píxeles. Estas coordenadas indican el punto central del recorte aplicado a la imagen.

- **FWHM (*Full Width at Half Maximum*)**: Ancho del kernel medido en segundos de arco. Este parámetro define la extensión del kernel de distorsión utilizado en los experimentos. Un valor mayor indica un kernel más amplio y, por lo tanto, una mayor distorsión. Un grado equivale a 3600 segundos de arco. En los experimentos,

`fwhm` se ajusta para observar cómo afecta la distorsión en la calidad de la imagen reconstruida.

- **SB (*Surface Brightness*)**: Límite de brillo superficial en la imagen, medido en una escala que indica el nivel máximo de brillo permitido. El valor de `sb` determina el nivel de brillo superficial que se acepta durante el procesamiento de la imagen. En la práctica, un valor más alto de `sb` indica una mayor tolerancia al brillo, lo que puede llevar a una reducción en el ruido perceptible en áreas brillantes de la imagen. Sin embargo, esto puede comprometer la calidad en regiones de bajo brillo. En los experimentos, `sb` se ajusta para observar cómo el límite de brillo superficial impacta la reconstrucción de la imagen. La unidad de medida es el arco segundo cuadrado (arcsec^2).

- **CDELT (*Pixel Size*)**: Resolución final de la imagen en segundos de arco. Este parámetro indica el tamaño del píxel en la imagen reconstruida. Un valor menor corresponde a una mejor resolución, ya que cubre una menor área del cielo. En los experimentos, `cdelt` se mantiene constante para asegurar una resolución consistente durante el análisis.

En la figura 6.46 se ilustra el proceso de generación y ajuste de imágenes durante las pruebas.

- La primera imagen muestra el recorte de una pequeña parte de la imagen original que se utilizará a lo largo de las pruebas. Este recorte se selecciona para concentrar el análisis en una región específica.

- La segunda imagen presenta el recorte aplicado con un kernel de distorsión.

- La tercera imagen muestra el recorte con la nueva resolución aplicada. Este paso es crucial para ajustar la escala de píxeles y preparar la imagen para la comparación final.

- La cuarta imagen representa el resultado final después de aplicar el límite de brillo superficial.

- Finalmente, la última imagen adicional corresponde a la imagen sin distorsión ni ruido, escalada a la nueva resolución de píxeles. Aunque no se espera que sea un

resultado perfecto, esta imagen de referencia permite comparar la deconvolución con una imagen sin distorsiones ni ruido, facilitando una evaluación del rendimiento del proceso de deconvolución.

Las dos primeras imágenes tienen mayor resolución de píxeles, se visualizan del mismo tamaño en esta figura por razones de presentación. La imagen completa antes del recorte no se ha añadido al no aportar nada relevante.

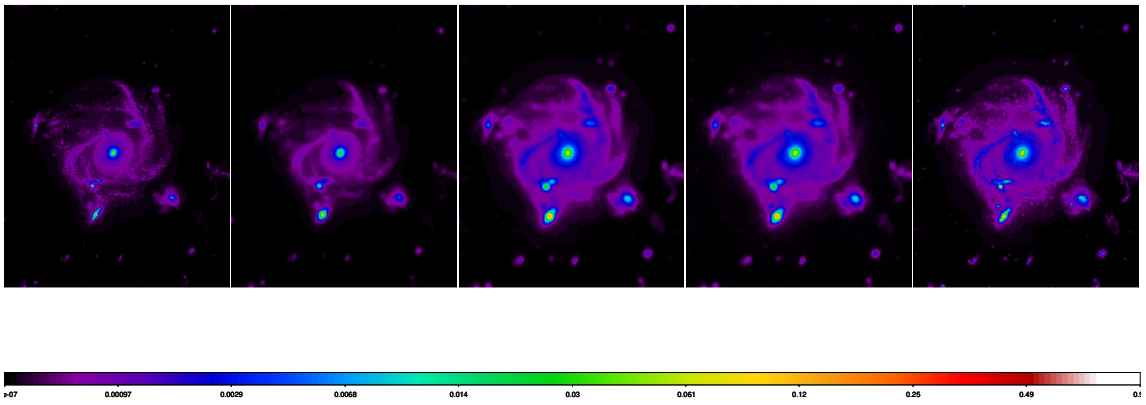

Figura 6.46: Proceso de preparación de la imagen, de izquierda a derecha: recorte de una pequeña parte de la imagen original, aplicación del kernel de distorsión, ajuste de la resolución, aplicación del límite de brillo superficial (usada como imagen detectada, I(x,y)), y una imagen de referencia sin distorsión ni ruido, escalada a la nueva resolución de píxeles para comparación (usada como imagen objetivo, O(x,y).

6.2.2. Efecto de la distorsión y el ruido

Uno de los desafíos principales al comparar algoritmos en este tipo de imágenes es la alta dimensionalidad de los parámetros involucrados. Aunque solo se varíen la distorsión del kernel y el límite de brillo superficial, se genera un espacio de experimentos considerablemente amplio. Para simplificar el análisis, se comenzará evaluando el algoritmo AWMLE, el cual, a nivel teórico, debería proporcionar los mejores resultados al ser el algoritmo que modela mejor la estadística típica en imágenes astronómicas, además de incorporar un análisis más complejo basado en la transformada wavelet. También se fijará el número de planos wavelet del algoritmo a 4, como en los experimentos anteriores. Inicialmente, se variará únicamente la distorsión del kernel. Una vez identificado el máximo nivel de

distorsión que permite una reconstrucción aceptable, se procederá a hacer lo mismo con el límite de brillo superficial. La imagen resultante de este proceso se utilizará posteriormente para evaluar la eficacia de los demás algoritmos.

Para ilustrar el impacto de la distorsión del kernel en la calidad de la imagen, se ha preparado la figura 6.47. La primera imagen corresponde a la versión sin distorsión, mientras que las imágenes siguientes muestran los distintos grados de distorsión, donde el parámetro FWHM se fija progresivamente en 0.15°, 0.25°, 0.5°, 0.75° y 1°. Este incremento gradual en FWHM permite visualizar cómo la calidad de la imagen se deteriora a medida que aumenta la distorsión del kernel.

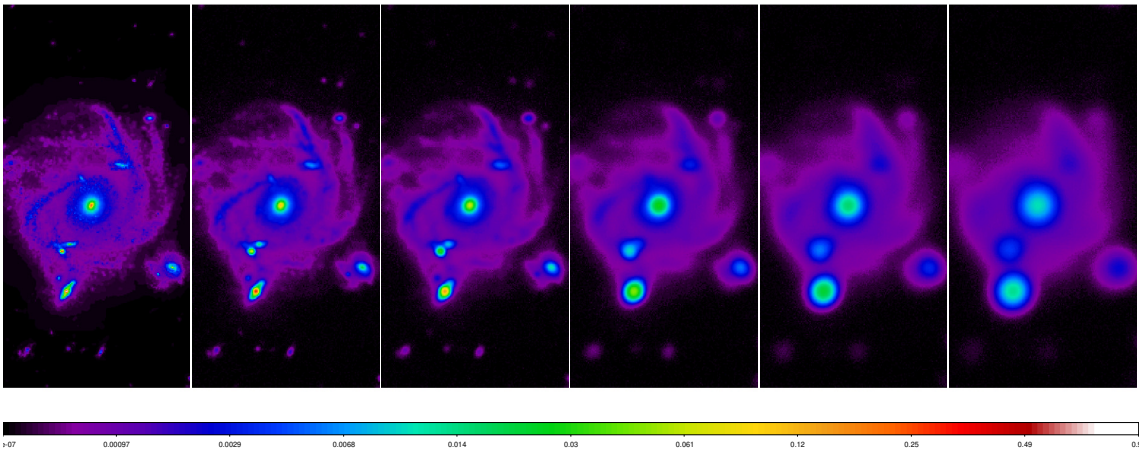

Figura 6.47: Efecto de la variación del kernel en la imagen de referencia. La primera imagen no tiene distorsión, mientras que las siguientes muestran niveles crecientes de FWHM: 0.15°, 0.25°, 0.5°, 0.75° y 1°.

Para no extender demasiado esta sección, se mostrarán únicamente los casos de FWHM de 0.25° y 0.5°, ya que en estos valores se produce el cambio de calidad más notable. En la figura 6.48, se observa el proceso de reconstrucción para FWHM = 0.25°, donde los detalles se preservan adecuadamente. Por otro lado, la figura 6.49 muestra la reconstrucción con FWHM = 0.5°, donde claramente los detalles comienzan a perderse.

Una vez fijado el kernel, se procederá a disminuir el límite de brillo superficial (SBL), lo que resultará en un aumento del ruido en la imagen. En los experimentos anteriores se utilizó un valor de SBL igual a 32 arcsec^2. Para esta sección, se reducirá gradualmente hasta alcanzar 27 arcsec^2, en decrementos de una unidad. En la figura 6.50 se presenta una comparativa entre la imagen original y los seis casos resultantes de variar el brillo superficial desde 32 arcsec^2 hasta 27 arcsec^2. El efecto en el ruido de la imagen sigue

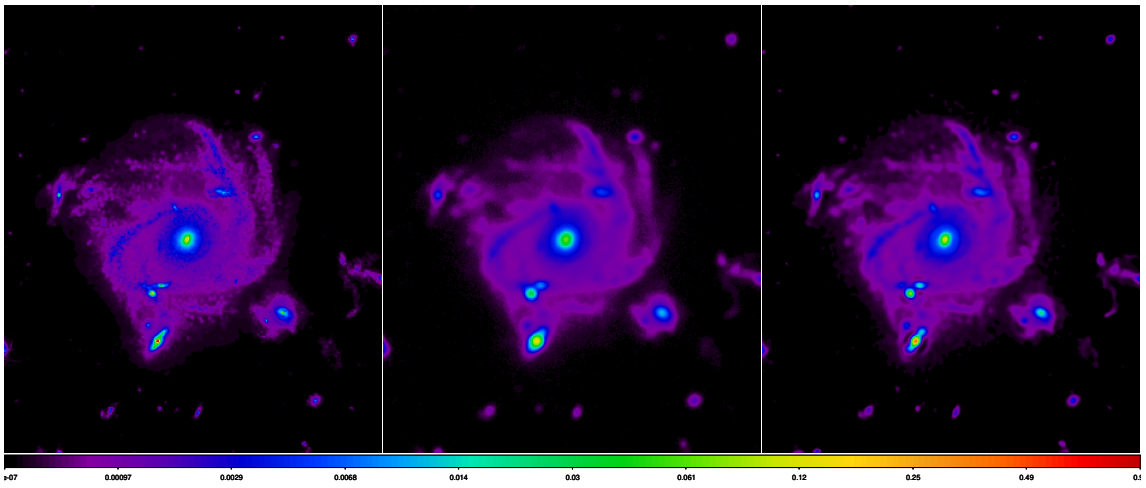

Figura 6.48: Secuencia del proceso de reconstrucción para FWHM = 0.25°: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE (7 iteraciones).

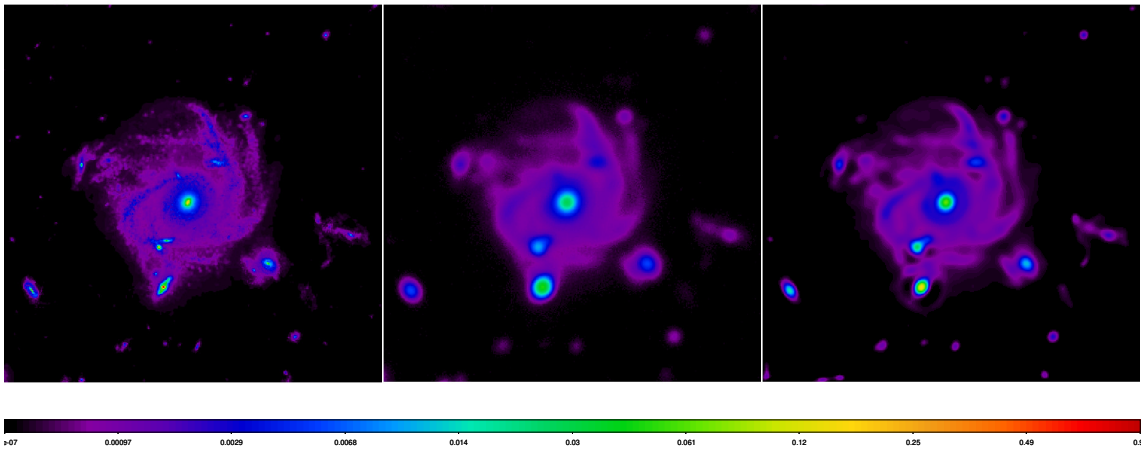

Figura 6.49: Secuencia del proceso de reconstrucción para FWHM = 0.5°: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE (8 iteraciones).

una escala logarítmica, lo que se refleja claramente en la evolución de la calidad de las imágenes.

Al igual que en el caso del kernel, se seleccionarán los dos casos más representativos: SBL = 29 arcsec^2 y SBL = 30 arcsec^2. En la figura 6.51, se puede observar cómo el contorno de la galaxia se desvanece y el ruido de fondo aumenta considerablemente. En contraste, en la figura 6.52, estos detalles aún son apreciables. Por lo tanto, se decide fijar el límite

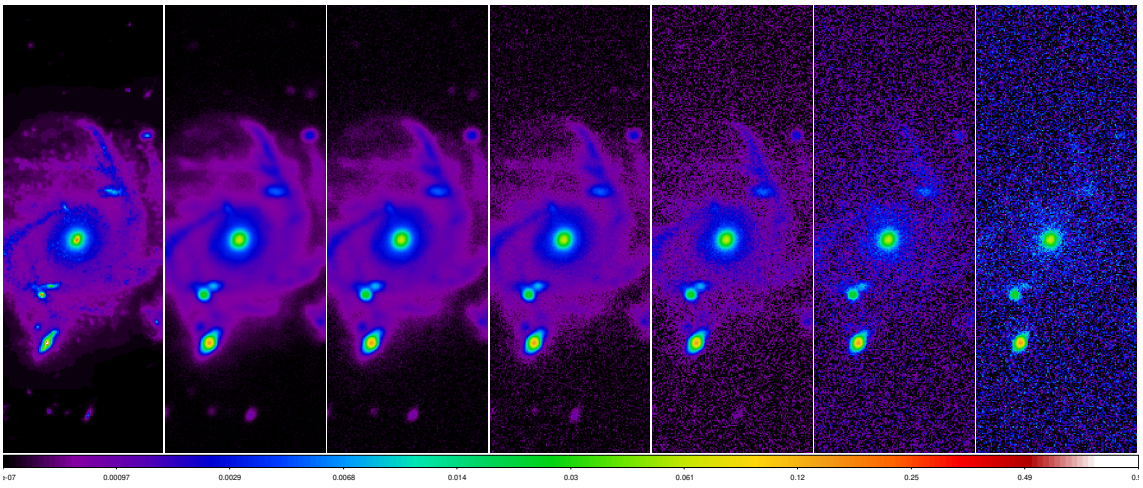

Figura 6.50: Efecto de la variación del límite de brillo superficial en la imagen de referencia. La primera imagen imagen es la original, mientras que las siguientes muestran niveles decrecientes de SBL: 32 arcsec^2, 31 arcsec^2, 30 arcsec^2, 29 arcsec^2, 28 arcsec^2, 27 arcsec^2

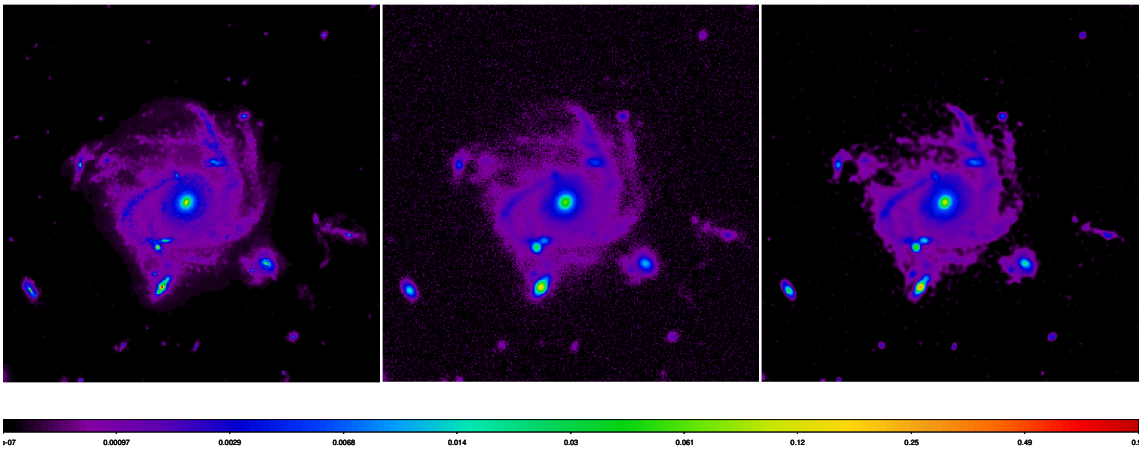

Figura 6.51: Secuencia del proceso de reconstrucción para SBL = 30 arcsec^2: Imagen original sin distorsión, Imagen distorsionada con ruido, Imagen resultante tras la deconvolución con el algoritmo AWMLE (6 iteraciones).

de brillo superficial en 30.

Aunque se ha omitido en el cuerpo principal, se ha optimizado el número de iteraciones y demás parámetros del algoritmo AWMLE para asegurar la mejor reconstrucción posible en cada caso. Además, en el Anexo B se presentan los experimentos adicionales para los casos que se omitieron anteriormente.

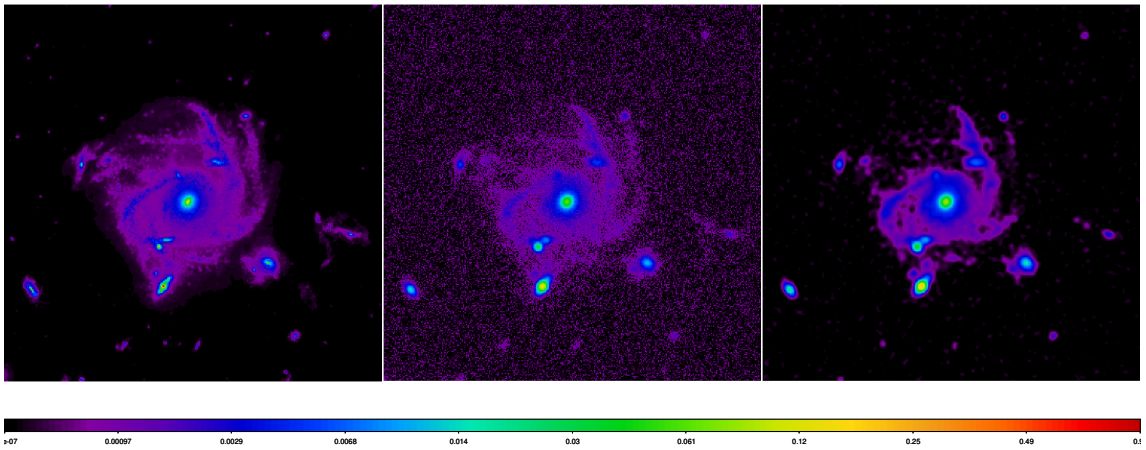

Figura 6.52: Secuencia del proceso de reconstrucción para SBL = 29 arcsec^2: Imagen original sin distorsión, Imagen distorsionada con ruido, Imagen resultante tras la deconvolución con el algoritmo AWMLE (4 iteraciones).

6.2.3. Evaluación Comparativa de Algoritmos de Deconvolución

En esta sección se procederá a comparar cómo se comportan los distintos algoritmos frente a una misma entrada. En la figura 6.53 se presenta la primera comparativa, utilizando los parámetros seleccionados anteriormente: SBL = 30 arcsec^2 y FWHM = 0.25°. De izquierda a derecha, se pueden observar la imagen de referencia, la imagen con ruido y distorsión, y los resultados de la deconvolución aplicados con los diferentes algoritmos: inversión directa, Tikhonov, RL, y AWMLE. Es evidente que el algoritmo de inversión directa genera principalmente ruido, mientras que los demás algoritmos logran reconstrucciones más precisas, con un RMSE óptimo similar de aproximadamente 1.2×10^{-4}.

Se procederá a aumentar tanto el límite de brillo superficial (SBL) como la distorsión (FWHM) para observar cómo evoluciona cada algoritmo en condiciones más desfavorables. Dado que el algoritmo de inversión directa no produjo buenos resultados en el caso más sencillo, se ha decidido excluirlo de los experimentos restantes. En la figura 6.54, se presenta el caso con SBL = 29.5 arcsec^2 y FWHM = 0.4°. Se puede observar cómo los resultados comienzan a divergir entre los algoritmos; por ejemplo, el algoritmo RL introduce más ruido y artefactos en el fondo. Se ha observado un aumento del RMSE hasta 3×10^{-4}, no hay una gran diferencia entre los diferentes algoritmos.

Sin embargo, no se ha observado una variación significativa en el RMSE en comparación con el experimento anterior.

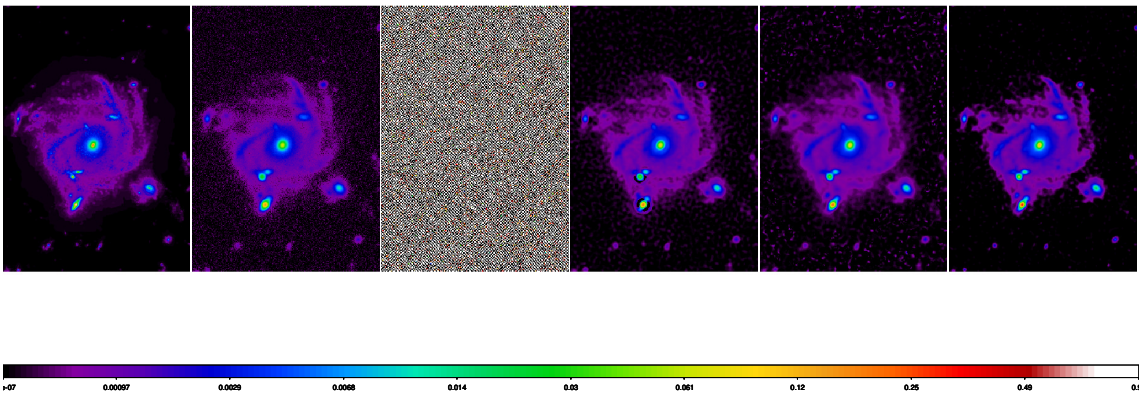

Figura 6.53: Comparativa de los resultados de diferentes algoritmos aplicados a la imagen distorsionada con SBL = 30 arcsec^2 y FWHM = 0.25°. De izquierda a derecha: imagen de referencia, imagen distorsionada, y resultados de la inversión directa, Tikhonov ($\lambda = 0.1$), RL (12 iteraciones) y AWMLE (4 iteraciones).

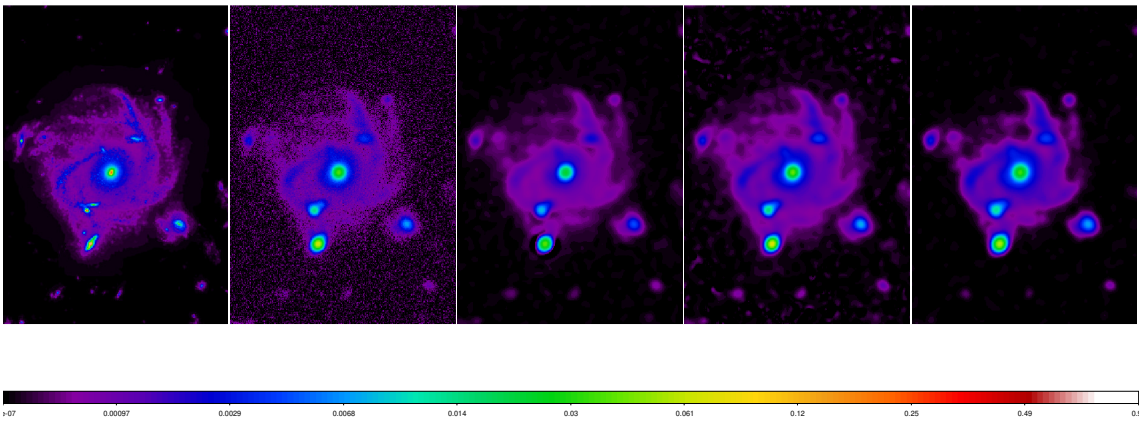

Figura 6.54: Comparativa de los resultados de diferentes algoritmos aplicados a la imagen distorsionada con SBL = 29.5 arcsec^2 y FWHM = 0.4°. De izquierda a derecha: imagen de referencia, imagen distorsionada, y resultados de Tikhonov ($\lambda = 0.1$), RL (11 iteraciones) y AWMLE (5 iteraciones).

Para el experimento con SBL = 29 arcsec^2 y FWHM = 0.6°, se observa que el RMSE permanece constante en torno a 3×10^{-4}, similar a los casos anteriores. Sin embargo, los resultados entre los algoritmos difieren aún más. El algoritmo RL introduce un mayor nivel de ruido en el fondo de la imagen, mientras que Tikhonov produce una reconstrucción más

difuminada. Por otro lado, el algoritmo AWMLE ha sido significativamente más agresivo, llegando a eliminar parte del contorno de la estructura original. Esto resalta las diferencias en la manera en que cada algoritmo maneja niveles más altos de distorsión y ruido. En la figura 6.55 puede verse la comparativa de los algoritmos.

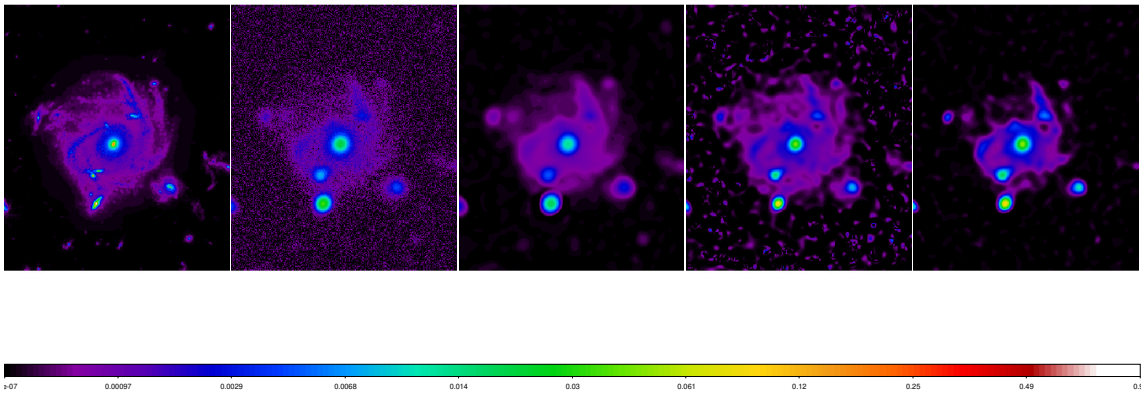

Figura 6.55: Comparativa de los resultados de diferentes algoritmos aplicados a la imagen distorsionada con SBL = 29 arcsec^2 y FWHM = 0.6°. De izquierda a derecha: imagen de referencia, imagen distorsionada, y resultados de Tikhonov ($\lambda = 0.05$), RL (8 iteraciones) y AWMLE (6 iteraciones).

En la figura 6.56 se muestra el caso con SBL = 28 arcsec^2 y FWHM = 0.75°. Aunque se observa que ninguno de los algoritmos logra una reconstrucción óptima, el RMSE sigue siendo similar entre ellos, manteniéndose en niveles parejos. Este resultado indica que, aunque las métricas de error no varían significativamente, la calidad visual de la reconstrucción se deteriora notablemente bajo estas condiciones desfavorables.

6.2.4. Mejora del desempeño con NoiseChisel

En esta sección, se introduce el uso de NoiseChisel como una estrategia para mejorar el desempeño de los algoritmos de deconvolución en imágenes con ruido. NoiseChisel permite optimizar la calidad de la reconstrucción mediante la creación de una máscara que distingue entre zonas con datos útiles y áreas afectadas por ruido.

El proceso implica generar una máscara que resalta las regiones de la imagen que contienen datos relevantes y oculta las áreas dominadas por ruido. Esta máscara se multiplica por la imagen distorsionada y ruidosa, eliminando las zonas no informativas y centrando el

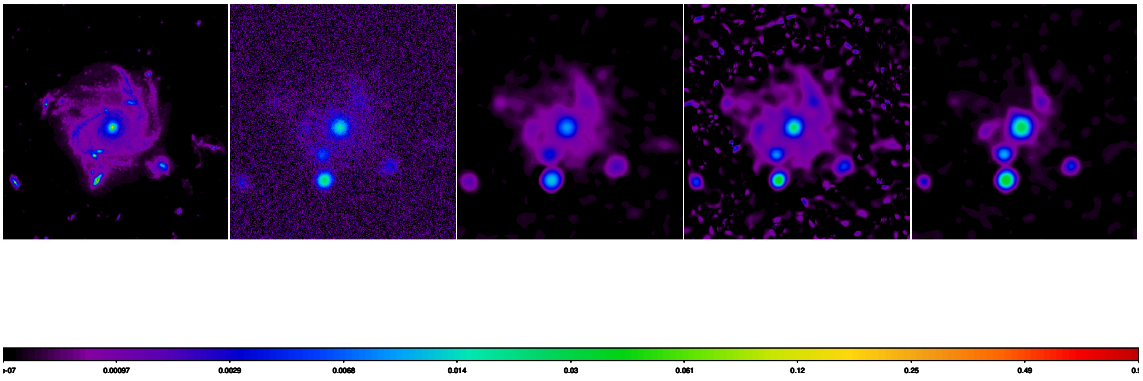

Figura 6.56: Comparativa de los resultados de diferentes algoritmos aplicados a la imagen distorsionada con SBL = 28 arcsec^2 y FWHM = 0.75°. De izquierda a derecha: imagen de referencia, imagen distorsionada, y resultados de Tikhonov ($\lambda = 0.001$), RL (6 iteraciones) y AWMLE (3 iteraciones).

análisis en las áreas con datos significativos. Cabe aclarar que esta máscara generada por NoiseChisel no tiene ninguna relación con la máscara generada por el algoritmo AWMLE.

El objetivo de esta técnica es reducir el impacto del ruido y mejorar la precisión de los algoritmos de deconvolución, ofreciendo así una forma eficaz de optimizar los resultados de las imágenes ruidosas y distorsionadas. En la figura 6.57 puede verse el proceso y el efecto final de aplicar el proceso descrito.

En la figura 6.58 se muestra el resultado de combinar el algoritmo Tikhonov con NoiseChisel. Se puede observar que, al aplicar NoiseChisel, parte del ruido de fondo ha sido significativamente reducido, lo que mejora el aspecto visual de la imagen reconstruida.

De igual manera, en la figura 6.59 se presenta la comparativa con el algoritmo Richardson-Lucy (RL). Aunque NoiseChisel elimina los artefactos de fondo, también introduce bordes más bruscos en comparación con la imagen original. Esta observación destaca la capacidad de NoiseChisel para reducir el ruido, pero también su tendencia a acentuar los bordes en el proceso.

Por último, en la figura 6.60 muestra el caso del algoritmo AWMLE combinado con NoiseChisel. En este caso, se observa una reducción del ruido de fondo, pero la diferencia visual no es tan pronunciada como en los casos anteriores.

Aunque el aspecto visual mejora con NoiseChisel, el RMSE no muestra una diferencia

significativa en ninguno de los tres casos, posiblemente debido a la eliminación de elementos del fondo que podrían ser parte de la imagen original.

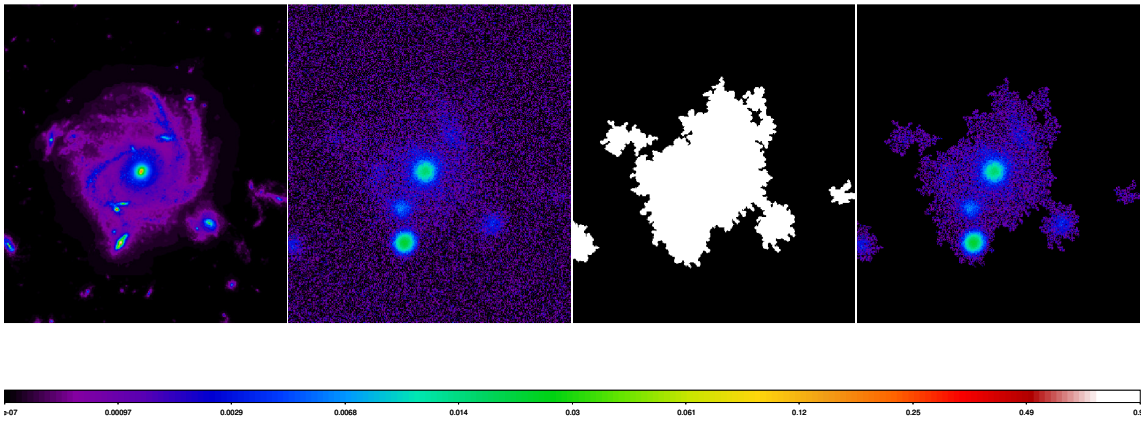

Figura 6.57: Proceso de aplicar una máscara generada con NoiseChisel: Imagen de referencia, imagen con ruido y distorsión, máscara generada por NoiseChisel (binaria), producto de la imagen distorsionada con la máscara.

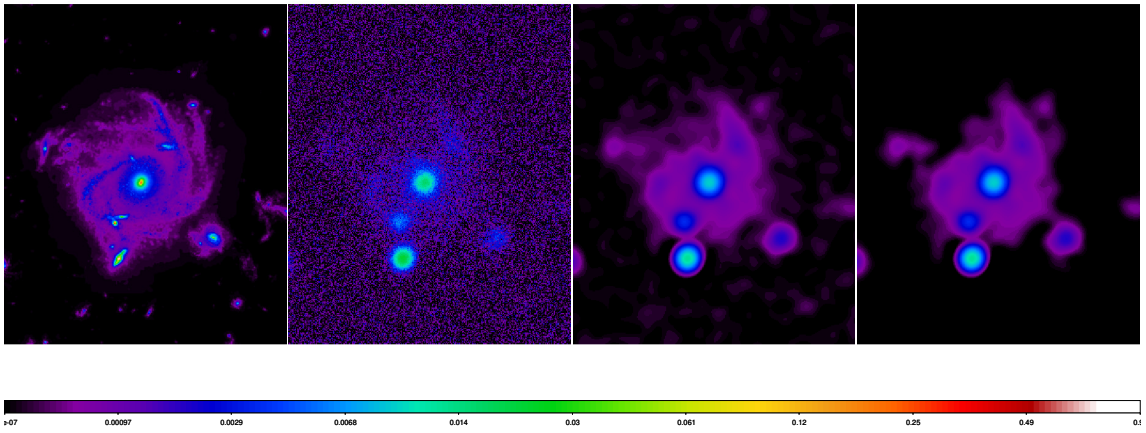

Figura 6.58: Comparativa del algoritmo Tikhonov con y sin NoiseChisel. De izquierda a derecha: imagen de referencia, imagen distorsionada con ruido, resultado del algoritmo Tikhonov, y resultado del algoritmo Tikhonov con NoiseChisel.

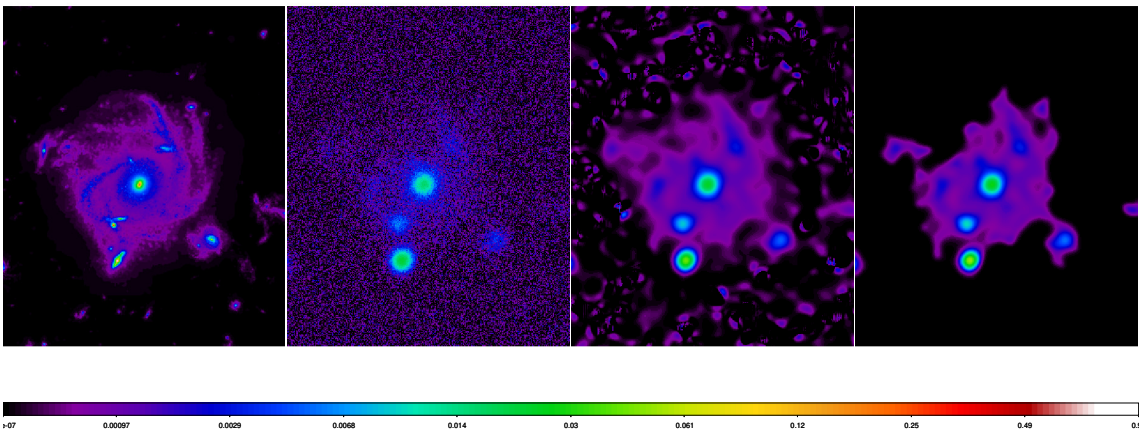

Figura 6.59: Comparativa del algoritmo Richardson-Lucy con y sin NoiseChisel. De izquierda a derecha: imagen de referencia, imagen distorsionada con ruido, resultado del algoritmo Richardson-Lucy, y resultado del algoritmo Richardson-Lucy con NoiseChisel.

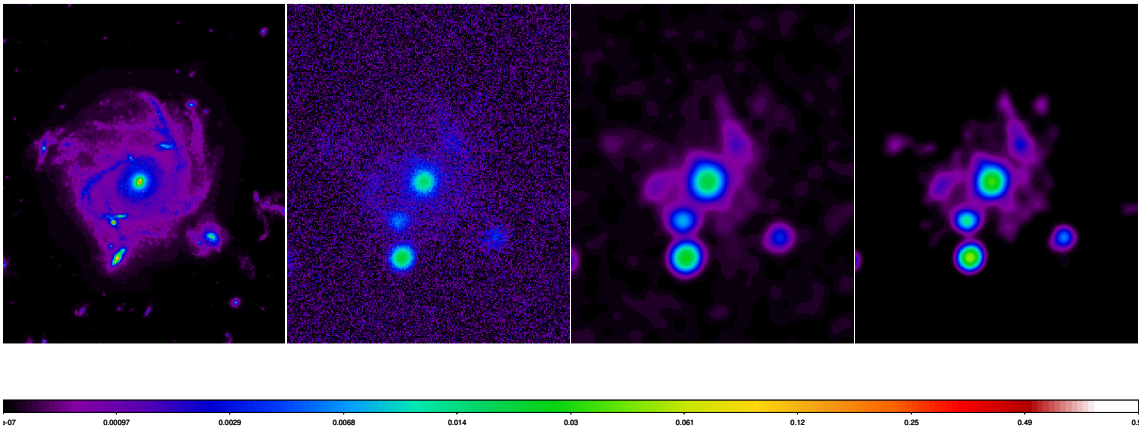

Figura 6.60: Comparativa del algoritmo AWMLE con y sin NoiseChisel. De izquierda a derecha: imagen de referencia, imagen distorsionada con ruido, resultado del algoritmo AWMLE, y resultado del algoritmo AWMLE con NoiseChisel.

7. Conclusiones y Trabajo Futuro

7.1. Resumen del Problema y Enfoque

Este trabajo ha abordado el problema de la deconvolución de imágenes astronómicas, un desafío crucial debido a la necesidad de mejorar la calidad de las imágenes capturadas mediante telescopios, las cuales suelen estar afectadas por distorsiones y ruido. El enfoque adoptado se centró en implementar y optimizar algoritmos de deconvolución dentro del marco del software Gnuastro, aprovechando su código libre y modularidad para facilitar futuras extensiones y mejoras.

7.2. Contribuciones y Resultados

El trabajo ha contribuido significativamente al desarrollo de Gnuastro, no solo mediante la implementación de diversos algoritmos de deconvolución, sino también a través del desarrollo de librerías de bajo nivel, como la transformada de Fourier y la transformada Wavelet. Estas herramientas han permitido obtener resultados satisfactorios y alineados con la teoría esperada. A continuación, se resumen las principales contribuciones y cómo se relacionan con los objetivos planteados:

1. **Implementación de Algoritmos de Deconvolución:** Se implementaron cuatro algoritmos principales: Inversión Directa, Tikhonov, Richardson-Lucy y AWMLE. Estos algoritmos se probaron exhaustivamente para garantizar su eficacia y precisión.

    - *Objetivo:* Proveer de herramientas robustas para la deconvolución de imágenes.
    - *Resultado:* Los algoritmos han demostrado ser efectivos, obteniendo resultados satisfactorios en términos de reducción de ruido y mejora de la calidad de la imagen.

2. **Desarrollo de Librerías de Bajo Nivel:** Se desarrollaron las librerías *fft.h*, *complex.h* y *wavelet.h* para soportar las operaciones de deconvolución.

    - *Objetivo:* Facilitar la implementación de los algoritmos de deconvolución y mejorar su rendimiento.

- *Resultado:* Estas librerías han mejorado la modularidad y eficiencia del software, permitiendo una implementación más fluida de los algoritmos.

3. **Creación de la Aplicación `astdeconvolve`:** Se desarrolló una aplicación de línea de comandos que integra los algoritmos de deconvolución con las funciones de validación y lectura/escritura de archivos de Gnuastro.

   - *Objetivo:* Proveer una herramienta accesible para los usuarios finales que permita aplicar los algoritmos de deconvolución de manera sencilla.
   - *Resultado:* `astdeconvolve` ha demostrado ser una herramienta útil y eficaz para los usuarios, facilitando la aplicación de los algoritmos de deconvolución en diversos escenarios.

7.3. Evaluación de los Resultados

El trabajo realizado ha cumplido en gran medida con los objetivos propuestos. Los algoritmos implementados han mostrado resultados satisfactorios y el desarrollo de librerías de apoyo ha mejorado significativamente la modularidad y eficiencia del software. Sin embargo, se reconoce que en situaciones reales de investigación, los algoritmos podrían necesitar optimizaciones adicionales para ser completamente efectivos.

7.4. Trabajo Futuro

Para mejorar y extender los resultados de este trabajo, se identifican varias líneas de investigación y desarrollo futuro:

1. **Optimización de Algoritmos:** Se recomienda optimizar aún más los algoritmos implementados, aprovechando el código libre de Gnuastro. Esto podría incluir la implementación de versiones más rápidas y eficientes.

2. **Integración de Algoritmos Avanzados:** Se podrían integrar algoritmos basados en *Machine Learning* y *Deep Learning* que han mostrado excelentes resultados en la deconvolución de imágenes (Makarkin y Bratashov, 2021).

3. **Soporte para Ejecución en GPU:** Desarrollar una librería que permita la ejecución de Gnuastro en GPU en lugar de CPU podría reducir significativamente los tiempos de ejecución.

4. **Desarrollo de Nuevos Módulos:** Futuras contribuciones podrían incluir nuevos módulos para optimizar aún más el rendimiento del software y ampliar sus capacidades. Además, librerias como la transformada de Fourier o Wavelet pueden acelerar nuevos módulos en el futuro.

5. **Desarrollo de scripts de alto nivel:** Combinando *astdeconvolve* con otros módulos (como se hizo con *astnoisechisel*) puede dar lugar a mejores resultados sin necesidad de crear nuevos algoritmos.

Estas perspectivas de futuro abren nuevas posibilidades para el campo de la astronomía y el procesamiento de imágenes, permitiendo a los investigadores obtener resultados más precisos y eficientes en sus estudios. La implementación y optimización de estos algoritmos no solo mejorarán la calidad de las imágenes astronómicas, sino que también facilitarán su análisis y la obtención de conclusiones científicas más robustas.

Por último, cabe añadir que, debido a motivos ajenos al trabajo, no se ha podido integrar esta aportación en Gnuastro a fecha de entrega, ya que no ha sido validada por un administrador. Por lo tanto, aún no está disponible. Se prevé que la integración se realizará antes de 2025 y que formará parte de la versión 1.0 de Gnuastro.

8. Bibliografía

Referencias

Akhlaghi, M. (2018, enero). *Gnuastro: GNU Astronomy Utilities.* Astrophysics Source Code Library, record ascl:1801.009.

Akhlaghi, M. (2019, octubre). Separating Detection and Catalog Production. En M. Molinaro, K. Shortridge, y F. Pasian (Eds.), *Astronomical data analysis software and systems xxvi* (Vol. 521, p. 299). doi: 10.48550/arXiv.1611.06387

Akhlaghi, M., y Ichikawa, T. (2015, septiembre). Noise-based Detection and Segmentation of Nebulous Objects. , *220*(1), 1. doi: 10.1088/0067-0049/220/1/1

Baena-Gallé, R., y Gladysz, S. (2011). Estimation of differential photometry in adaptive optics observations with a wavelet-based maximum likelihood estimator. *Publications of the Astronomical Society of the Pacific*, *123*(905), 865.

Brassarote, G., Souza, E., y Monico, G. (2018, 05). Non-decimated wavelet transform for a shift-invariant analysis. *Tema (São Carlos)*, *19*, 93. doi: 10.5540/tema.2018.019.01.93

Chun-Lin, L. (2010). A tutorial of the wavelet transform. *NTUEE, Taiwan*, *21*(22), 2.

Corrado, E. M. (2005, Mar.). The importance of open access, open source, and open standards for libraries: Theme: Open access journals. *Issues in Science and Technology Librarianship*(42). Descargado de `https://journals.library.ualberta.ca/istl/index.php/istl/article/view/2002` doi: 10.29173/istl2002

Free Software Foundation. (2024). *The gnu operating system and the free software movement.* Descargado de `https://www.gnu.org/` (Accessed: 2024-09-05)

Galatsanos, N. P., y Katsaggelos, A. K. (1992). Methods for choosing the regularization parameter and estimating the noise variance in image restoration and their relation. *IEEE Transactions on image processing*, *1*(3), 322–336.

GNU Astronomy Utilities. (s.f.). *Gnuastro developing section.* `https://www.gnu.org/software/gnuastro/manual/html_node/Developing.html`. (Accedido: 2024-06-26)

Gnuastro Savannah Repository. (s.f.). *Gnuastro Savannah Repository.* https://www.gnu.org/savannah-checkouts/gnu/gnuastro/gnuastro.html.

Golub, G. H., Heath, M., y Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, *21*(2), 215–223.

Hasinoff, S. W. (2021). Photon, poisson noise. En K. Ikeuchi (Ed.), *Computer vision: A reference guide* (pp. 980–982). Cham: Springer International Publishing. Descargado de `https://doi.org/10.1007/978-3-030-63416-2_482` doi: 10.1007/978-3-030-63416-2_482

Hoffman, F. (1997). An introduction to Fourier theory.

Jefferies, S. M., y Christou, J. C. (1993). Restoration of astronomical images by iterative blind deconvolution. *Astrophysical Journal v. 415, p. 862*, *415*, 862.

Lucy, L. B. (1974). An iterative technique for the rectification of observed distributions. *Astronomical Journal*, *79*, 745.

Makarkin, M., y Bratashov, D. (2021). State-of-the-art approaches for image deconvolution problems, including modern deep learning architectures. *Micromachines*, *12*(12). Descargado de `https://www.mdpi.com/2072-666X/12/12/1558` doi: 10.3390/mi12121558

Murtagh, F., Starck, J.-L., y Pantin, E. (2002, 10). Deconvolution in astronomy: A review. *Publications of the Astronomical Society of the Pacific*, *114*. doi: 10.1086/342606

Palomares, F. G., Monsoriu, J. A., y Alemany, E. (2016). Aplicación de la convolución de matrices al filtrado de imágenes. *Modelling in Science Education and Learning*, *9*(1), 97–108.

Pence, W., Chiappetti, L., y Shaw, R. (2010, 12). Definition of the flexible image transport system (fits), version 3.0. *http://dx.doi.org/10.1051/0004-6361/201015362*, *524*. doi: 10.1051/0004-6361/201015362

Richardson, W. H. (1972). Bayesian-based iterative method of image restoration. *JoSA*, *62*(1), 55–59.

Snyder, G. F., Lotz, J. M., Rodriguez-Gomez, V., Guimarães, R. d. S., Torrey, P., y Hernquist, L. (2017, junio). Massive close pairs measure rapid galaxy assembly in mergers at high redshift. *mnras*, *468*(1).

Starck, J., Murtagh, F., y Bertero, M. (2011). *The starlet transform in astronomical data processing: Application to source detection and image deconvolution.* Springer New York, NY, USA.

Starck, J.-L., Murtagh, F. D., y Bijaoui, A. (1998). *Image processing and data analysis: The multiscale approach.* Cambridge University Press.

Thiebaut, E. (2006, enero). Introduction toimage reconstruction and inverse problems. En (Vol. 198, p. 397-422). doi: 10.1007/1-4020-3437-7_25

A. Documentación del código

En este anexo se detallarán a alto nivel las funciones implementadas en el trabajo. Se explicará qué tarea realiza cada función y se proporcionará su prototipo, permitiendo así una comprensión clara de su propósito y uso.

A.1. complex.h

A continuación se describirán las funciones desarrolladas para la librería de números complejos:

gal_complex_create_padding

Descripción: La función `gal_complex_create_padding` genera un relleno en el kernel (padding) y convierte tanto el kernel como la imagen astronómica de tipo real a complejo. También adapta la imagen para que tenga dimensiones impares.

Prototipo:

```
void gal_complex_create_padding(const gal_data_t *image,
const gal_data_t *kernel, gsl_complex_packed_array *outputimage,
gsl_complex_packed_array *outputkernel,size_t *xdim, size_t *ydim);
```

gal_complex_normalize

Descripción: La función `gal_complex_normalize` normaliza un array de números complejos. Esta operación se aplica sobre el vector de entrada (in place).

Prototipo:

```
void gal_complex_normalize(gsl_complex_packed_array inout, size_t size);
```

gal_complex_cumulative_sum

Descripción: La función `gal_complex_cumulative_sum` suma todos los elementos de un array.

Prototipo:

```
double gal_complex_cumulative_sum(gsl_complex_packed_array input, size_t size);
```

gal_complex_to_real

Descripción: La función `gal_complex_to_real` convierte un array de números complejos en una representación de números reales, aplicando una acción especificada por el usuario.

Prototipo:

```
double *gal_complex_to_real (gsl_complex_packed_array complexarray,
                             size_t size, complex_to_real action);
```

gal_complex_conjugate

Descripción: La función `gal_complex_conjugate` calcula el conjugado de un array de números complejos dado. El conjugado de un número complejo $a + bi$ es $a - bi$.

Prototipo:

```
gsl_complex_packed_array
gal_complex_conjugate (gsl_const_complex_packed_array input, size_t size);
```

gal_complex_multiply

Descripción: La función `gal_complex_multiply` realiza una multiplicación elemento a elemento de dos arrays de números complejos.

Prototipo:

```
gsl_complex_packed_array gal_complex_multiply (gsl_complex_packed_array first,
                                               gsl_complex_packed_array second,
                                               size_t size);
```

gal_complex_divide

Descripción: La función `gal_complex_divide` realiza una división elemento a elemento de dos arrays de números complejos. Si el divisor es menor que un valor mínimo dado, el resultado se establecerá en 0.

Prototipo:

```
gsl_complex_packed_array gal_complex_divide (gsl_complex_packed_array first,
                                             gsl_complex_packed_array second,
                                             size_t size, double minvalue);
```

gal_complex_scale

Descripción: La función `gal_complex_scale` multiplica cada elemento de un array por un valor dado. Esta operación se aplica sobre el vector de entrada (in place).

Prototipo:

```
void gal_complex_scale(gsl_complex_packed_array inout, double value,
size_t size);
```

gal_complex_add_scalar

Descripción: La función `gal_complex_add_scalar` suma un número escalar a cada uno de los elementos de un array de números complejos dado.

Prototipo:

```
gsl_complex_packed_array
gal_complex_add_scalar (gsl_const_complex_packed_array input, size_t size,
                        gsl_complex scalar);
```

gal_complex_power

Descripción: La función `gal_complex_power` eleva cada elemento de un array de números complejos a una potencia dada.

Prototipo:

```
gsl_complex_packed_array gal_complex_power (gsl_complex_packed_array input,
                                            double exponent, size_t size);
```

gal_complex_real_to_complex

Descripción: La función `gal_complex_real_to_complex` convierte un array de números reales (*double*) a *gsl_complex_packed_array*. **Prototipo**:

```
gsl_complex_packed_array gal_complex_real_to_complex (double *realarray,
                                                      size_t size);
```

gal_complex_substract

Descripción: La función `gal_complex_substract` realiza la resta entre el primer y el segundo array complejo. **Prototipo**:

```
gsl_complex_packed_array
gal_complex_substract (gsl_complex_packed_array first,
                       gsl_complex_packed_array second, size_t size);
```

A.2. fft.h

gal_fft_shift_center

Descripción: La función `gal_fft_shift_center` realiza un desplazamiento a lo largo de los ejes X e Y para mover el centro del kernel al píxel (0,0).

Prototipo:

```
void gal_fft_shift_center(gsl_complex_packed_array kernel, size_t *dim);
```

fft_init

Descripción: La función `fft_init` inicializa todas las estructuras necesarias para realizar una transformada de Fourier bidimensional (FFT). Es una función interna y no se puede usar fuera de la librería fft.

Prototipo:

```
void fft_init(fft_params **params,size_t numthreads,
size_t *dim, gsl_const_complex_packed_array input,
gsl_complex_packed_array output, gsl_fft_direction sign);
```

fft_free

Descripción: La función `fft_free` libera los recursos generados por la función `gal_fft_init`. Es una función interna y no se puede usar fuera de la librería fft.

Prototipo:

```
void fft_free(fft_params *params, size_t numthreads);
```

gal_fft_two_dimension_transformation

Descripción: La función `gal_fft_two_dimension_transformation` realiza una transformada de Fourier bidimensional (FFT) a una imagen de entrada. Trabaja en multi-hilo y permite tanto la transformada inversa como directa dependiendo del parámetros *sign*.

Prototipo:

```
gsl_complex_packed_array gal_fft_two_dimension_transformation (
    gsl_const_complex_packed_array input, size_t *dim, size_t numthreads,
    size_t minmapsize, gsl_fft_direction sign);
```

fft_one_direction

Descripción: La función `fft_one_direction` realiza una transformada de Fourier rápida (FFT) en una dirección dada de una cantidad determinada de columnas o filas. Está diseñada para ser multihilo. Es una función interna y no se puede usar fuera de la librería fft.

Prototipo:

```
void *fft_one_direction(void *p);
```

A.3. wavelet.h

gal_wavelet_no_decimate

Descripción: La función `gal_wavelet_no_decimate` realiza la transformada wavelet no decimada. El resultado de esta operación una lista del tipo *gal_data_t** y mediante el atributo *next* se puede acceder a los distintos planos (el último apunta a *NULL*).

Prototipo:

```
gal_data_t *gal_wavelet_no_decimate (const gal_data_t *image,
                                     uint8_t numberplanes, size_t numthreads,
                                     size_t minmapsize);
```

gal_wavelet_free

Descripción: La función `gal_wavelet_free` libera los recursos utilizados por `gal_wavelet_no_decima`

Prototipo:

```
void gal_wavelet_free (gal_data_t *wavelet);
```

gal_wavelet_add_padding

Descripción: La función `gal_wavelet_add_padding` ajusta el tamaño de una imagen de entrada añadiendo ceros.

Prototipo:

```
double *gal_wavelet_add_padding (double *input, size_t *inputsize,
                                 size_t *outputsize);
```

wavelet_init_father_function

Descripción: La función `wavelet_init_father_function` inicializa la función padre para la transformada wavelet. Uso interno.

Prototipo:

```
gal_data_t *wavelet_init_father_function (size_t minmapsize);
```

wavelet_expand_father_function

Descripción: La función `wavelet_expand_father_function` expande la función padre duplicando su tamaño en cada eje mediante interpolación. Uso interno.

Prototipo:

```
gal_data_t *wavelet_expand_father_function (gal_data_t *father, size_t factor,
                                            size_t minmapsize);
```

wavelet_substract

Descripción: La función `wavelet_substract` resta dos planos wavelet. Uso interno.

Prototipo:

```
double *wavelet_substract (double *first, double *second, size_t size);
```

A.4. deconvolution.h

gal_deconvolve_direct_inversion

Descripción: La función `gal_deconvolve_direct_inversion` realiza la deconvolución de inversión directa (simplemente divide la imagen por la PSF).

Prototipo:

```
gal_data_t *gal_deconvolve_direct_inversion (const gal_data_t *image,
                                             const gal_data_t *PSF,
                                             size_t numthreads,
                                             size_t minmapsize);
```

gal_deconvolve_tikhonov

Descripción: La función `gal_deconvolve_tikhonov` realiza la deconvolución según el algoritmo wiener-tikhonov. Requiere el parámetro lambda configurable por el usuario.

Prototipo:

```
gal_data_t *gal_deconvolve_tikhonov (const gal_data_t *image,
                                     const gal_data_t *PSF, double lambda,
                                     size_t numthreads, size_t minmapsize);
```

gal_deconvolve_richardson_lucy

Descripción: La función `gal_deconvolve_richardson_lucy` realiza la deconvolución según el algoritmo de Richardson-Lucy. Requiere el parámetro alpha (por defecto, 1) y el número de iteraciones (por defecto, 75) por el usuario.

Prototipo:

```
gal_data_t *gal_deconvolve_richardson_lucy (const gal_data_t *image,
                                            const gal_data_t *PSF,
                                            size_t iterations, double alpha,
                                            size_t minmapsize,
                                            size_t numthreads);
```

deconvolve_richardson_lucy_next_solution

Descripción: La función `deconvolve_richardson_lucy_next_solution` calcula la siguiente iteración del algoritmo de Richardson-Lucy. Uso interno.

Prototipo:

```
void deconvolve_richardson_lucy_calculate_next_solution (
    gsl_complex_packed_array *solution, gsl_complex_packed_array h,
    gsl_complex_packed_array h_f, gsl_complex_packed_array image, double alpha,
    size_t *dsize, size_t minmapsize, size_t numthreads);
```

deconvolve_richardson_lucy_init_solution

Descripción: La función `deconvolve_richardson_init_next_solution` calcula la solución inicial del algoritmo de Richardson-Lucy. Uso interno.

Prototipo:

```
gsl_complex_packed_array
deconvolve_richardson_lucy_init_solution (size_t size);
```

gal_deconvolve_AWMLE

Descripción: La función `gal_deconvolve_AWMLE` realiza la deconvolución de inversión según el algoritmo AWMLE. Requiere el parámetro alpha (por defecto, 1), el número de iteraciones (por defecto, 75), el número de planos, la tolerancia y la desviación estándar del ruido por el usuario. Depende de las funciones internas *deconvolve_AWMLE_update_object*, *deconvolve_calcule_AWMLE_noise_factor*, *deconvolve_AWMLE_estimate_p_prime*, *deconvolve_AWMLE_increment_correction_term* y *deconvolve_calcule_AWMLE_mask*.

Prototipo:

```
gal_data_t *gal_deconvolve_AWMLE (const gal_data_t *image,
                                  const gal_data_t *PSF, size_t iterations,
                                  size_t waves, double tolerance, double sigma,
                                  double alpha, size_t minmapsize,
                                  size_t numthreads, size_t *early);
```

B. Comparativas adicionales

B.1. Efecto del Kernel

A continuación se mostrarán los experimentos correspondientes a variar el FWHM de 0.15 a 1.

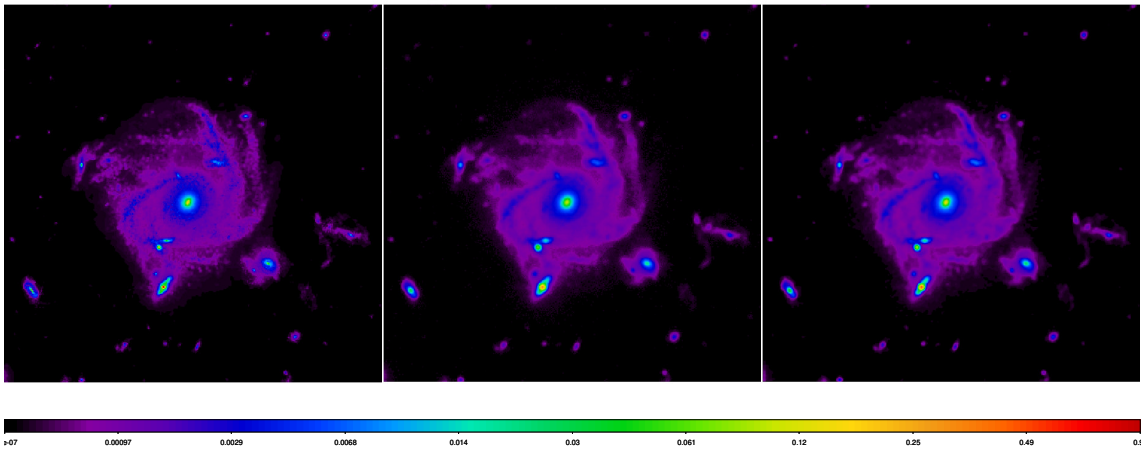

Figura B.1: Secuencia del proceso de reconstrucción para FWHM = 0.15: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

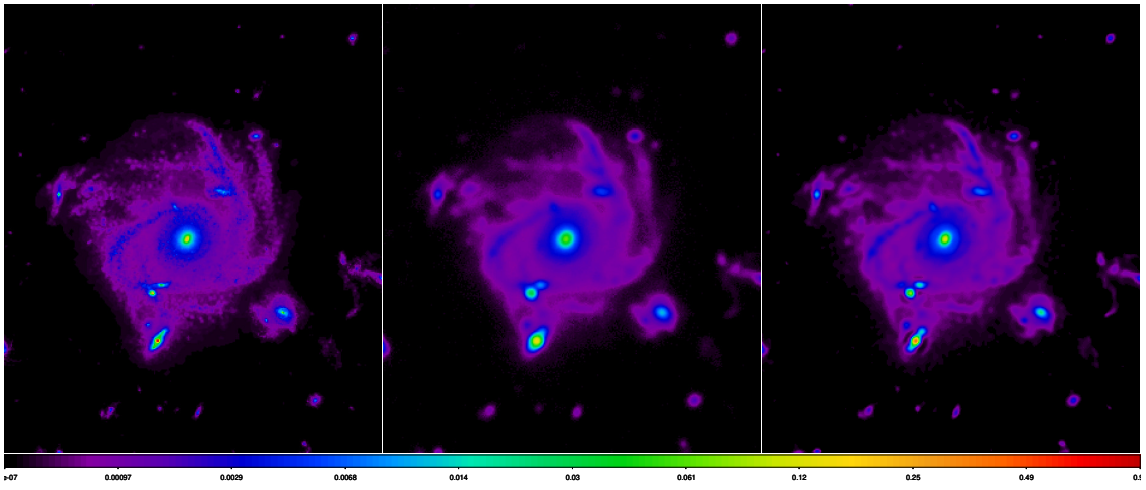

Figura B.2: Secuencia del proceso de reconstrucción para FWHM = 0.25: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

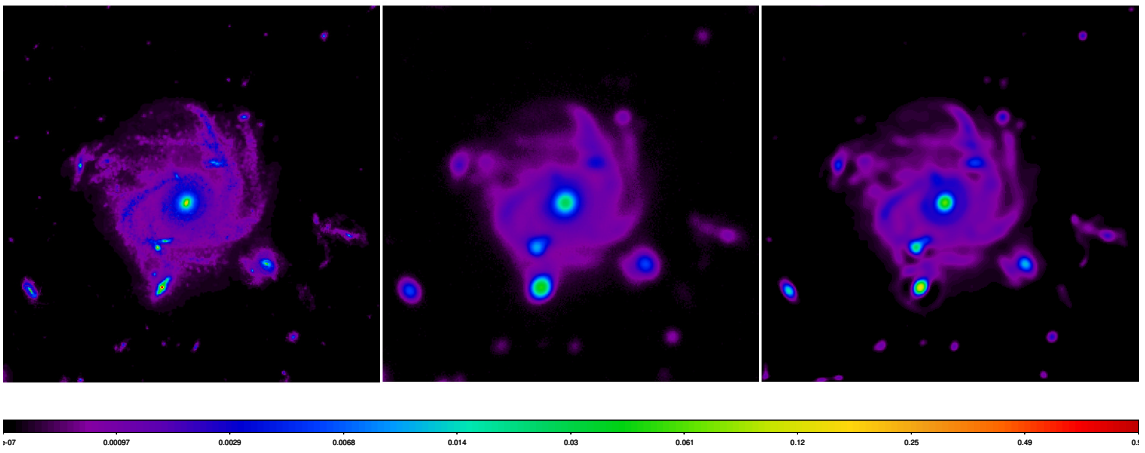

Figura B.3: Secuencia del proceso de reconstrucción para FWHM = 0.5: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

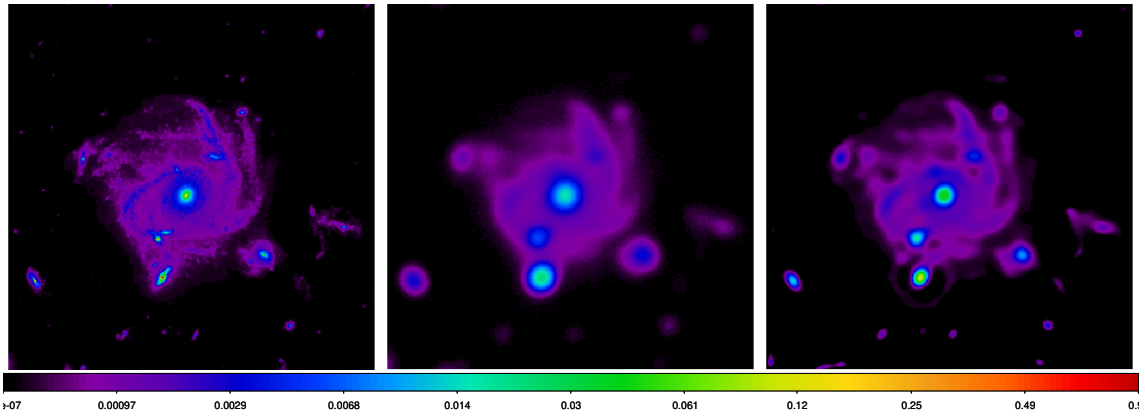

Figura B.4: Secuencia del proceso de reconstrucción para FWHM = 0.75: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

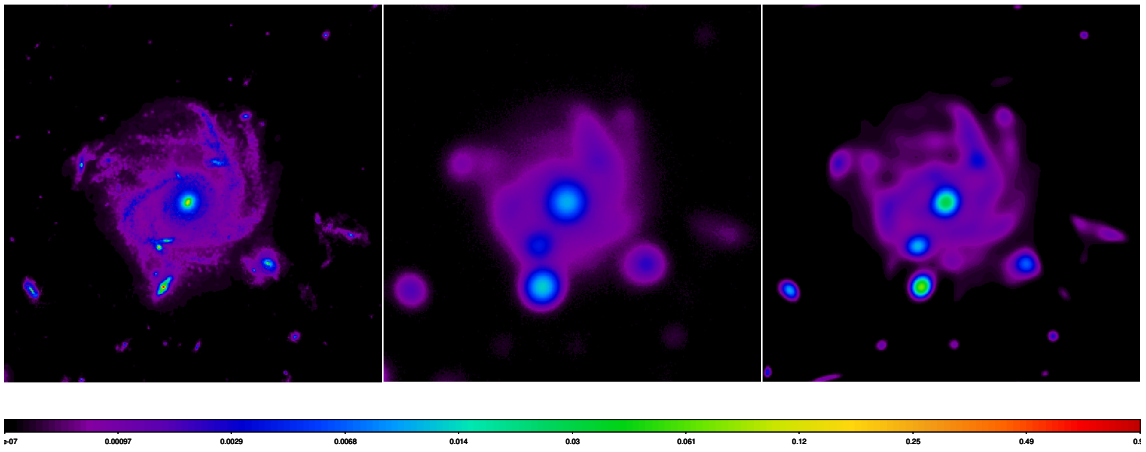

Figura B.5: Secuencia del proceso de reconstrucción para FWHM = 1: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

B.2. Efecto del límite de brillo superficial

A continuación se mostrarán los experimentos correspondientes a variar el SBL desde 31 arcsec^2 a 27 arcsec^2 tras fijar el FWHM a 0.25.

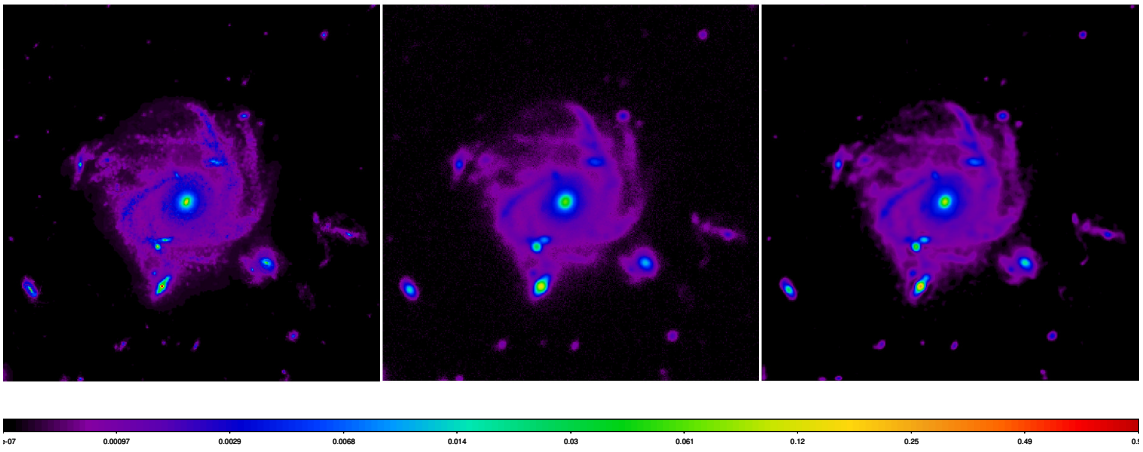

Figura B.6: Secuencia del proceso de reconstrucción para SBL = 31 arcsec^2: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

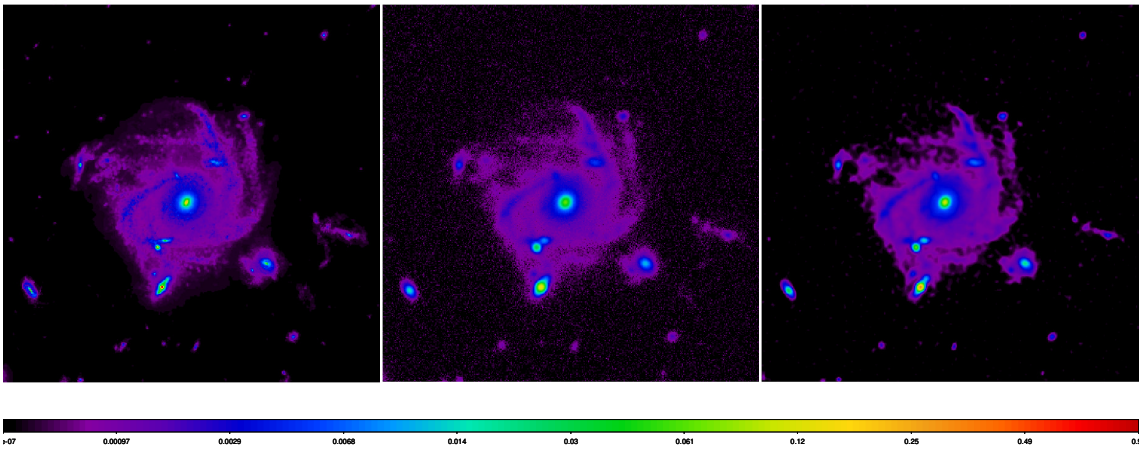

Figura B.7: Secuencia del proceso de reconstrucción para SBL = 30 arcsec^2: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

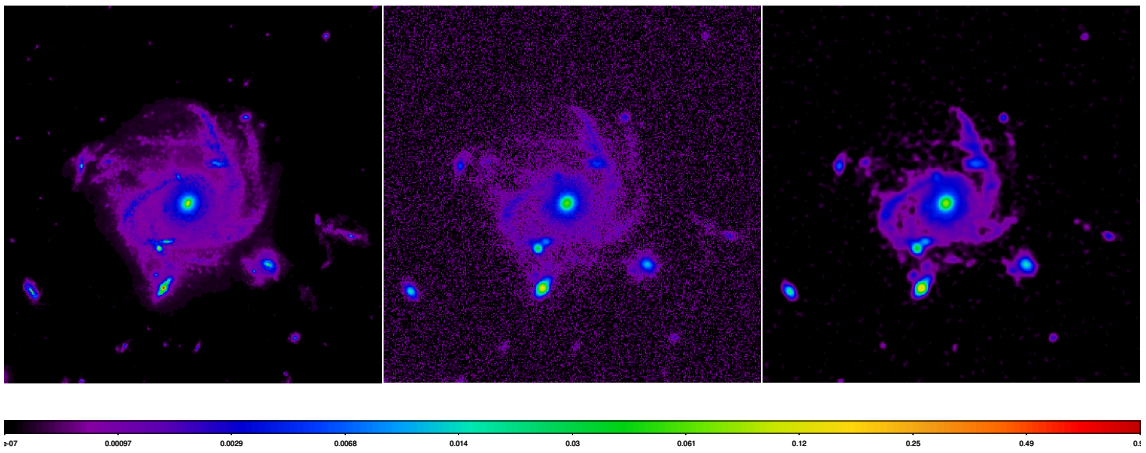

Figura B.8: Secuencia del proceso de reconstrucción para SBL = 29 arcsec^2: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

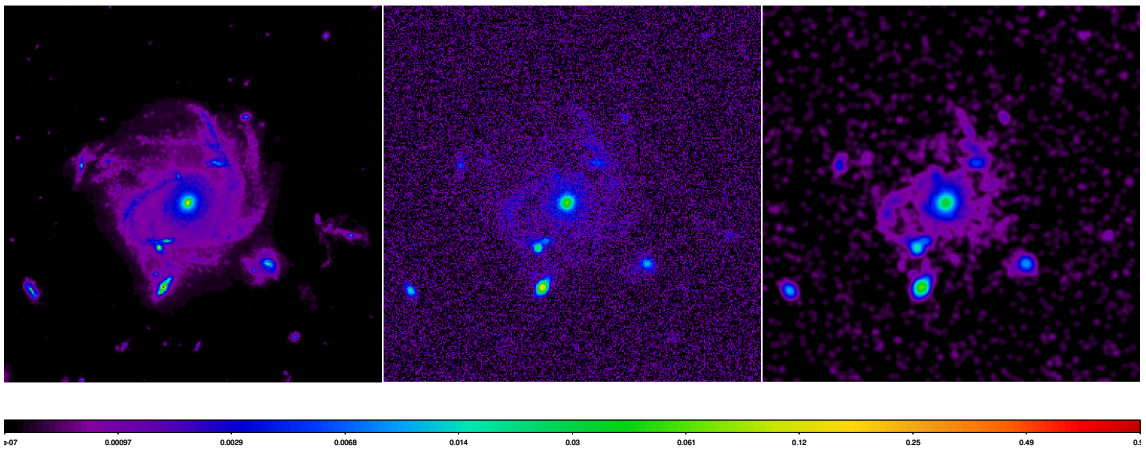

Figura B.9: Secuencia del proceso de reconstrucción para SBL = 28 arcsec^2: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.

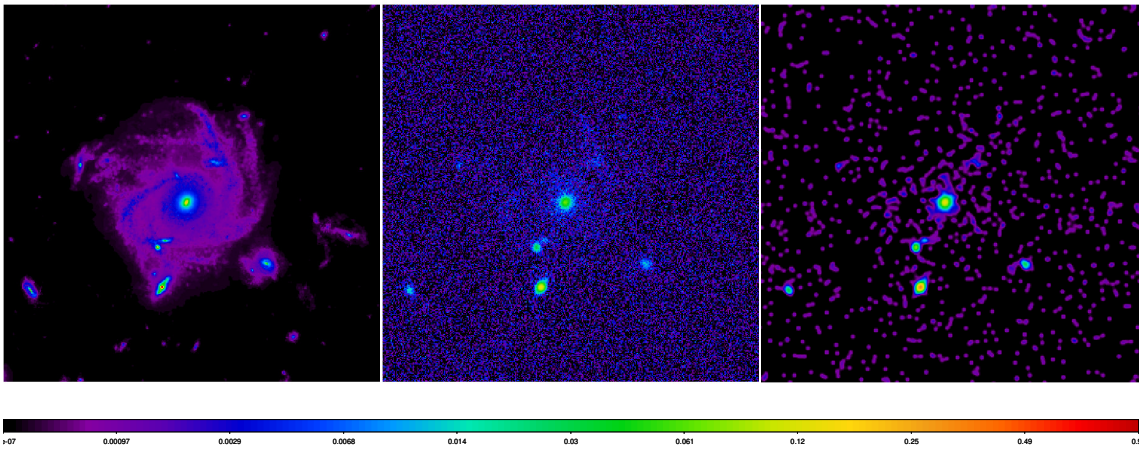

Figura B.10: Secuencia del proceso de reconstrucción para SBL = 27 arcsec^2: Imagen original sin distorsión, Imagen distorsionada con el kernel aplicado, Imagen resultante tras la deconvolución con el algoritmo AWMLE.