Facultad de Ciencias
Universidad Zaragoza

ANÁLISIS CUANTITATIVO Y TRATAMIENTO DE IMÁGENES PARA LA CARACTERIZACIÓN DE ESTRUCTURAS CRISTALINAS EN MICROGRAFÍAS DE MICROSCOPÍA ELECTRÓNICA CON RESOLUCIÓN ATÓMICA

Sergio Córdoba

Trabajo Fin de Máster en
"Modelización e investigación matemática, estadística y computación"

2014-2015

Dirigido por:

Dr. Ricardo Celorrio (UZ)
Dr. César Magén (UZ-INA)
Dr. Pedro Mateo (UZ)

Instituto Universitario de Investigación de Matemáticas y Aplicaciones
Universidad Zaragoza

Instituto Universitario de Investigación en Nanociencia de Aragón
Universidad Zaragoza

Agradecimientos

Es momento de agradecer a todos los que me han apoyado, empujado, alegrado, ayudado, motivado, criticado, consolado, animado, enseñado, derrotado, corregido, acompañado, abandonado, defraudado, sorprendido... porque todos, queriendo o sin querer, con consciencia o sin ella, han jugado un papel para que hoy esté escribiendo estas líneas.

Gracias a mis padres, hermana y amigos, por estar ahí de manera incondicional y constante.

Gracias a mis directores, porque el haber sido "víctima de su cooperación" (como me dijo uno de ellos en cierta ocasión), ha supuesto una suerte y privilegio para mí.

Gracias, en especial, a Ana y Andrea, porque su sonrisa es mi fuerza y, por supuesto, a Miriam, por acompañarme, contra viento y marea, en mi caminar.

Sergio Córdoba

"La naturaleza tiene perfecciones
para demostrar que
es imagen de Dios, e imperfecciones para
probar que sólo es una imagen"

Blaise Pascal

Contenido

Resumen

El objetivo de este trabajo es el desarrollo de una herramienta informática, concretamente un Plug-In para el software de tratamiento de imágenes *Digital Micrograph* (DM), que permitirá estimar posiciones y distancias entre las columnas de átomos representadas en micrografías de materiales obtenidas mediante técnicas de microscopía electrónica de transmisión en barrido (STEM). La medida de estas magnitudes permite caracterizar, de forma local, propiedades físicas del material, como es la ferroelectricidad, que están relacionadas con pequeñas variaciones de su estructura cristalina.

Se pretende modelizar estas micrografías usando una mixtura de normales de forma que cada una de sus componentes represente y modele una de las columnas de átomos visibles en la imagen. Usaremos el algoritmo de Levenberg-Marquardt para realizar el ajuste al modelo y obtener los parámetros de estas normales. En concreto, las medias, supondrán una estimación de la posición de cada columna, con lo que resultará inmediato medir distancias entre ellas. Se quiere además, que estas medidas alcancen precisiones picométricas, puesto que son desplazamientos atómicos de este orden los que, en algunos casos, motivan la aparición de ciertas propiedades.

El tarea principal del proyecto será pues la implementación computacional de este proceso, realizada con el lenguaje de programación, *R*, y el lenguaje de scripting propio de DM. Para ello, se deberá entender cómo se generan estas micrografías (capítulo 1), lo cual justifica la elección del modelo probabilista elegido. Posteriormente, se realizarán simulaciones de las mismas sobre las que probaremos y verificaremos el funcionamiento del algoritmo de ajuste (capítulo 2). Con esto, estaremos en disposición de usar dicho algoritmo para desarrollar el plug-in y probarlo en una micrografía STEM real (capítulo 3). Dichos códigos en R y en lenguaje de DM se presentan y explican en los apéndices.

PALABRAS CLAVE: Micrografía, TEM, STEM, mixtura de normales, modelización de la imagen, algoritmo de Levenberg-Marquardt, Digital Micrograph, R, estadística descriptiva, imagen digital.

Abstract

The main objective in this work is the development of a computational tool for the processing image software *Digital Micrograph* (DM), that will allow to estimate positions and distances between columns of atoms shown in micrographs of materials obtained by Scanning Transmission Electron Microscopy (STEM) techniques. The measurement of these magnitudes allows to characterize the local physical properties of the material, such as ferroelectricity, which are directly related to small variations of its crystal structure.

The intention is to model these micrographs using a mixture of gaussians so that, each one of its components represents one of those columns of atoms visible at image. We will use the Levenberg-Marquardt algorithm to fit the model, and to obtain the parameters for these gaussians. Specifically, their means will be an estimation to the position of columns of atoms, whereby it will be inmediate to measure distances between them. Also, we want that measurements of distances reach accuracies of order of picometres, because some properties are originated by atomic displacements of this order.

Thus, the main task in this work will be the computational implementation of this process using the computer language R, and the DM scripting language. For that purpose, we should understand how the experimental micrographs are generated (chapter 1) to justify the probabilistic model selected. Later, we will do micrographs simulations to test and verify the fit algorithm (chapter 2). Finally, we will use this algorithm to develope the plug-in and apply it to a real STEM micrograph (chapter 3). Both the R and DM codes are listed and described in the appendices.

KEY-WORDS: Micrograph, TEM, STEM, mixture of normal distributions, image modelling, Levenberg-Marquardt algorithm, Digital Micrograph, R, descriptive statistics.

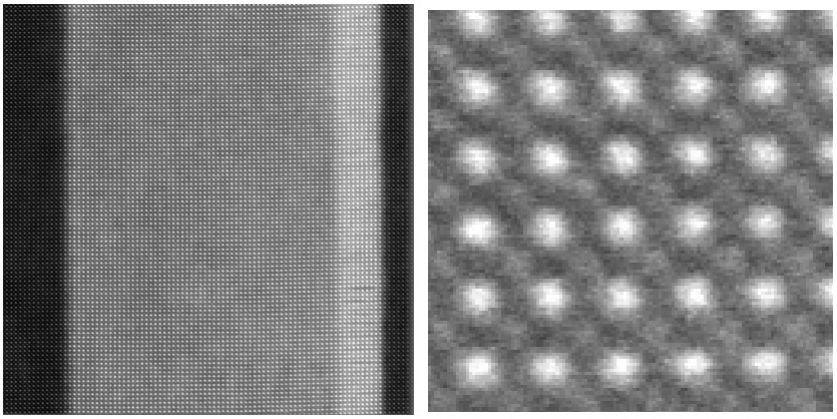
Capítulo 1

Introducción

En las últimas décadas el desarrollo de nuevas tecnologías y dispositivos, junto con la evolución en ciertas áreas científicas, como la biomedicina, ciencia de materiales o la nanociencia, ha dado lugar a que tengamos una capacidad e interés en la observación de los materiales a escala atómica como no se había tenido nunca. Una muestra de estas observaciones son las micrografías, que son imágenes digitales de cuerpos u objetos microscópicos, y en torno a las cuales se centra este trabajo.

Para conocer en detalle y controlar la materia en la nanoescala se requieren las más avanzadas técnicas de nanocaracterización, de las cuales sólo la microscopía TEM (*Transmission Electron Microscopy*) es capaz de proporcionar micrografías de la estructura cristalina de los materiales consiguiendo resoluciones espaciales inferiores a 1 angstrom. Además, se plantea el reto de analizar cuantitativamente las imágenes adquiridas (un ejemplo en la figura 1.1), con el fin de extraer magnitudes físicas con resolución atómica a partir de relaciones entre parámetros estructurales y químicos locales y propiedades físicas cuantificables macroscópicamente. Un caso muy interesante de aplicación del análisis cuantitativo de imágenes se da con los materiales ferroeléctricos, en los que la propia ferroelectricidad se deriva del desplazamiento relativo de distintos átomos con cargas eléctricas opuestas para formar localmente dipolos eléctricos. Habitualmente bastan pequeños desplazamientos atómicos, en el rango de los picómetros, para desencadenar este comportamiento. Así pues, no basta con la simple inspección de la imagen, si no que es necesario desarrollar modelos teóricos de su distribución de intensidades para posteriormente realizar la determinación de las posiciones atómicas.

Asumiremos un modelo probabilístico en el que consideraremos la micrografía como una realización de una distribución multimodal bivariante, en concreto una *mixtura de normales*, a partir de la cual estimaremos variables



(a) (b)

Figura 1.1: *a) Se presenta una micrografía real de tipo HAADF-STEM (de* 2048×2048 *pixeles) de una multicapa de un óxido cristalino, b) Detalle de micrografía de un área correspondiente con un óxido de lantano (columnas más brillantes) y manganeso (columnas menos brillantes). Se puede apreciar el pixelaje de la imagen, otro de los elementos que resultará clave en el trabajo.*

latentes asociadas con las posiciones de las columnas atómicas. Cabe notar que el modelo probabilista resulta coherente con la física del problema puesto que la micrografía es consecuencia directa de impactos de electrones sobre el detector.

1.1. Micrografías STEM

El TEM es una técnica de caracterización que genera imágenes digitales de alta resolución de un material. Se fundamenta en la dispersión de un haz de electrones de alta energía al incidir sobre una muestra delgada, del orden de unos 100 nm de espesor. Los electrones transmitidos a través de la muestra llegan a un detector que mide y procesa la señal que generan. A cada punto de la superficie de la muestra representada (a cada pixel[1]) se le asigna una intensidad proporcional[2] a la señal medida en este punto. Así pues se trata

[1]Obviaremos este concepto, que no resulta ajeno en la "Era Digital" que vivimos, pero dejaremos la definición que da la RAE: *Superficie homogénea más pequeña de las que componen una imagen, que se define por su brillo y color*

[2]Los electrones que impactan en el detector generan una señal eléctrica que se procesa siendo amplificada. Si el dispositivo encargado de esto opera linealmente, se da la proporcionalidad mencionada entre electrones detectados y el brillo asignado. Habitualmente se trata de que esto así sea y así lo consideraremos aquí.

de micrografías en blanco y negro. De esta forma se obtienen imágenes como la que podemos ver en la figura 1.1, donde las zonas con mayor intensidad se identifican con las columnas de átomos de la red.

Existen diversas técnicas de imagen TEM (figura 1.2), cuya diferencia radica en la forma en que se ilumina la muestra y se detectan los electrones transmitidos. En el modo STEM (Scanning Transmission Electron Microscopy), se forma una sonda electrónica sub-nanométrica que barre la superficie de la muestra y un detector anular recoge los electrones transmitidos dispersados a alto ángulo. Por la geometría de este detector, en puntos donde hay columnas de átomos, se recoge un mayor número de electrones dispersados a alto ángulo. A su vez, cuanto más pesados sean estos átomos, mayor es la intensidad en esos pixeles. Esta proporcionalidad directa entre el número de eventos y el brillo del pixel es clave en el posterior desarrollo. Esta técnica de imagen STEM se denomina HAADF (*High Angle Annular Dark Field*).

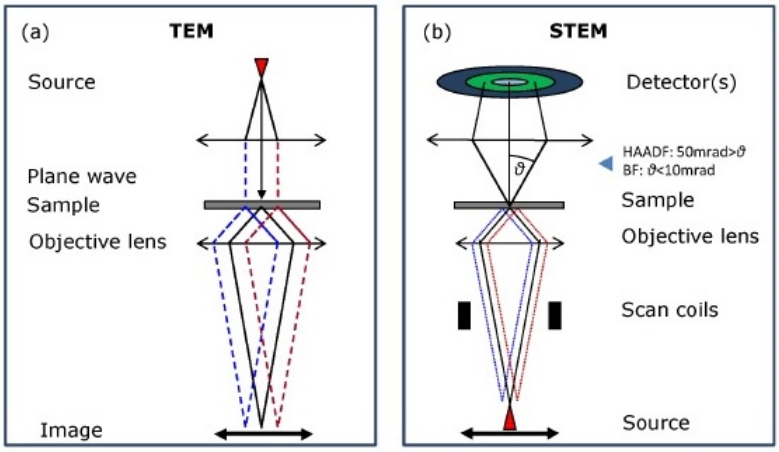


Figura 1.2: *Esquema de la formación de imágenes con dos técnicas TEM diferentes: TEM y STEM (figura obtenida de la referencia [1])*

1.2. Interpretación y tratamiento de la micrografía

La intensidad de cada pixel viene dada como un número entero y positivo, con lo que la "matriz de pixeles" que compone la imagen va ligada a una matriz de datos, con el mismo número de filas y columnas, de manera que cada dato de la segunda, corresponde a la intensidad del pixel que ocupe la misma posición en la primera.

Por otra parte, cada pixel es referenciado en la imagen por la columna y fila que ocupa, que pueden considerarse como coordenadas de posición discretas,

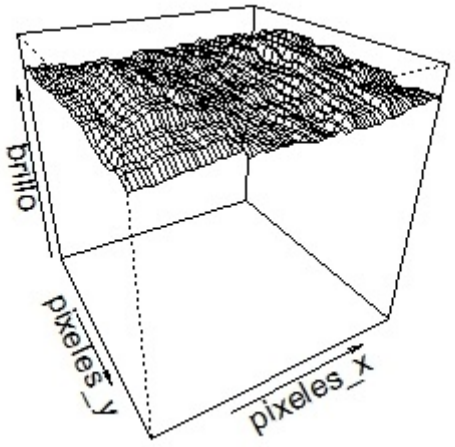
$$\begin{aligned} &x \equiv columna \in \{1, 2, 3...n_x\},\ con\ n_x = n^{\text{o}}\ de\ columnas, \\ &y \equiv fila \in \{1, 2, 3...n_y\},\ con\ n_y = n^{\text{o}}\ de\ filas. \end{aligned} \tag{1.1}$$

Cada pixel quedará caracterizado completamente por estas coordenadas de posición y su intensidad, pudiendo considerar esta última como una tercera coordenada perpendicular al plano de la imagen, de manera que se puede interpretar como un relieve sobre dicho plano. Puesto que este dato va asociado a un número de eventos detectados, pensaremos en la imagen como un histograma en tres dimensiones. Con esto, le podemos dar un tratamiento estadístico y modelizar la intensidad de la imagen mediante distribuciones de probabilidad.

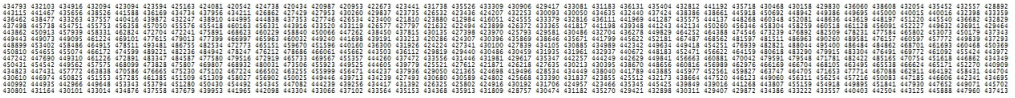
Si las coordenadas de posición (expresiones (1.1)) se consideran continuas y definidas de la siguiente manera,

$$\begin{aligned} &x \equiv coord.\ horizontal\ del\ centro\ del\ pixel \in (0, n_x), \\ &\qquad\qquad con\ n_x = n^{\text{o}}\ de\ columnas \\ &y \equiv coord.\ vertical\ del\ centro\ del\ pixel \in (0, n_y), \\ &\qquad\qquad con\ n_y = n^{\text{o}}\ de\ filas \end{aligned} \tag{1.2}$$

se tiene que cada pixel se representa como un punto en un espacio real de tres dimensiones, con coordenadas x, y (expresiones (1.2)) e intensidad. Estos puntos quedan contenidos en una superficie como la que se muestra en la figura 1.3, que ilustra la manera en la que vamos a interpretar la imagen, y sugiere la posibilidad de usar, para la modelización, distribuciones continuas y conocidas que nos otorguen cierta capacidad descriptiva.



(a) Representación de la matriz en b)



(b) Matriz de datos

Figura 1.3: *Región, de* 36×14 *pixeles (como las dimensiones de la matriz de datos), de la micrografía de la figura 1.1 representada con R. Esta representación, extraída de la matriz, corresponden a dos de las columnas de átomos más brillantes. Es de notar que no se tienen picos acusados, si no más bien, de poca altura y anchos superpuestos sobre una señal de fondo mucho mayor que la altura de dichos picos.*

Capítulo 2

Modelización de la imagen

Se asumirá un modelo probabilístico (no determinista) simple para la interacción entre el haz de electrones y el material cristalino que se emplea de espécimen. Según este la dispersión de los electrones debida a la interacción con una columna de átomos del cristal es una variable aleatoria bivariante que sigue una ley de distribución normal, y cuya media coincide con la proyección del eje central de la columna de átomos en el plano de la imagen.

La imagen asociada a la micrografía se basa en el recuento de los electrones que impactan en un escintilador para cada pixel de la imagen, por lo que el resultado del mismo aproximará la forma de la función de densidad. La radiación producida por el escintilador es amplificada por un fotomultiplicador, con lo que en la imagen adquirida se regula el contraste (amplificación o ganancia) y el brillo (desplazamiento, señal de fondo u *offset*) para operar en la zona lineal del amplificador que procesa la señal detectada.

2.1. Modelo matemático

En micrografías HAADF-STEM de resolución atómica de materiales cristalinos, como en la de la figura 1.1 b), correspondiente a un óxido de lantano y manganeso. Allí se aprecian zonas más o menos circulares y brillantes que corresponden a las columnas atómicas de lantano. Nótese que entre las zonas más destacadas tenemos otras, menos brillantes, correspondientes a columnas de manganeso. Este hecho puede corresponder con la vista de una de las caras de una celda BCC[1], con los átomos de lantano, en los vértices, más

[1]Estructura cristalina con celda unidad cúbica, siendo los átomos de los vértices y del centro del cubo (del inglés *Body Centered Cubic*), elementos químicos con distinto número atómico. Hay que decir que en realidad esta muestra no es de este tipo, pero es lo que visualmente se aprecia en la imagen.

destacados que el de manganeso del centro. Se propone modelizar cada una de estas regiones de la imagen mediante una distribución normal bivariante. Además, de las distribuciones con media y varianza conocidas, la normal maximiza la entropía, es decir, es la que introduce menos información a priori en el modelo.

La expresión explícita para una normal bivariante es

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \cdot \cdot \exp\left(-\frac{1}{2(1-\rho^2)}\left(\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{(\sigma_x\sigma_y)}\right)\right), \tag{2.1}$$

con lo que tenemos 5 parámetros (μ_x, media en x; μ_y, media en y; σ_x, desviación típica en x; σ_y, desviación típica en y y ρ, el coeficiente de correlación) que definirán la superficie de ajuste, identificando μ_x y μ_y con una estimación de la posición de la columna atómica en cuestión. Puesto que en una imagen encontramos muchas de ellas, se elige un modelo basado en una *Mixtura (o Mezcla) de Normales* (remitimos a [5] para más detalle).

Una mixtura de normales es una variable aleatoria cuya función de densidad es una combinación lineal, de la forma

$$g(x,y) = \sum_{i=1}^{G} \pi_i f_i(x,y), \quad \text{con } \sum_{i=1}^{G} \pi_i = 1,\ \pi_i > 0,\ i = 1,\ldots,G \tag{2.2}$$

donde cada uno de los G términos de la sumatoria (cada f_i es una distribución normal que se denomina *componente* de la mixtura) modeliza una columnas de átomos e introduce otro parámetro, el π_i, que se denomina *peso* de la componente y que se asocia a su intensidad en la imagen.

Prueba de normalidad

En la figura 1.3 se puede ver la presencia de una señal de fondo u "offset" sobre el que se superpone el relieve que describe la intensidad de la imagen, y el cual se pretende modelizar como mixtura de normales. Esta constante, que corresponde a los mínimos de intensidad en la imagen, se fija en los ajustes experimentales realizados durante la adquisición de la imagen y, por tanto, se tiene en cualquier micrografía. Deberá de considerarse este fenómeno en el modelo.

Por otro lado, podemos representar una sección de esta superficie. Tomamos los datos, por ejemplo, de la séptima fila de la matriz de la figura

1.3 b), restamos su valor mínimo para anular el offset y tenemos un perfil como el de la figura 2.1. Se debe hacer notar que esto corresponde a una sección en el plano $(pixeles_x, brillo)$ y que pasa aproximadamente por el centro del eje $pixeles_y$ (con la notación de la figura 1.3). Si aplicamos el test de

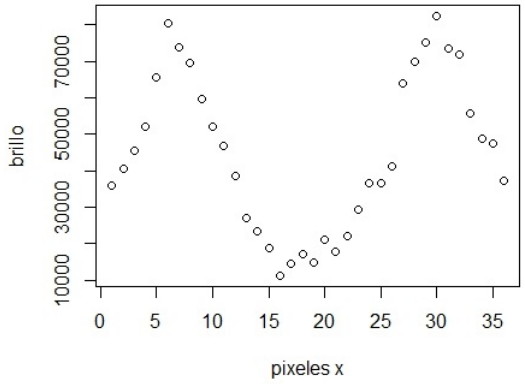


Figura 2.1: *Perfil de la superficie presentada en la figura 1.3 tomando los datos de la fila 7 de la matriz en la misma figura, habiendo restado el valor mínimo de la matriz para observar mejor el relieve.*

Shapiro-Wilk[2] (código en el apéndice E) a los picos que se observan, obtenemos p-valores mayores que la significancia, de 0,05 por defecto. En concreto, se obtiene 0,4246 en el primer pico y 0,08209 en el segundo con lo que se aceptaría la normalidad de ambas muestras.

Aunque la normalidad de este perfil no implica la normalidad de la superficie, se aporta como argumento a favor para el modelo de mixtura. Por otro lado, se realizó esta prueba para distintas secciones de los picos[3] y se obtuvo en todo caso resultados que apoyan su normalidad.

[2]Este test es recomendable para muestras pequeñas ($n < 50$ en muchos textos) por lo que calcularemos un vector de frecuencias proporcional al que se representa en la figura 2.1, y restringido a que la suma de sus elementos sea aproximadamente 50. Además, en R se aplica el test para un vector de observaciones (resultados de una medida), no de frecuencias. Esto requiere un nuevo vector que tenga tantas copias de la variable aleatoria (número de columna de pixeles, en este caso, como variable aleatoria de posición) como indique su frecuencia asociada. Este vector tendrá una longitud igual a la suma de frecuencias (aproximadamente 50, como proponíamos).

[3]En concreto, tomando las columnas 7 y 30 de la misma matriz (figura 1.3 b)) se obtienen p-valores de 0,5057 y 0,3387, respectivamente. Estas columnas corresponderían, aproximadamente, a las secciones perpendiculares a la de la figura 2.1 y que pasan por los máximos de cada pico.

2.1.1. Modelado del efecto del amplificador

Para operar en la zona lineal del amplificador que procesa la señal detectada, a la imagen $g(x, y)$ de (2.2) se ajusta el contraste y el brillo del detector. A efectos de nuestro modelo matemático, esto equivale a la introducción de una constante aditiva ,$\gamma_1 \in \mathbb{R}$, que modelice el offset, y otra multiplicativa, $\gamma_2 > 0$, para la amplificación: $F(x, y) = \gamma_2(g(x, y) + \gamma_1)$, es decir,

$$F(x, y) = \alpha + \sum_{i=1}^{G} w_i f_i(x, y), \tag{2.3}$$

siendo $w_i = \gamma_2 \cdot \pi_i > 0$ con $i = 1, \ldots, G$, los coeficientes de la nueva mixtura (nótese que ya no es una distribución de probabilidad) y $\alpha = \gamma_1 \cdot \gamma_2 \in \mathbb{R}$, el término independiente.

Finalmente consideraremos que las micrografías son muestreos en mallas regulares (cuadrícula) del modelo continuo $F(x, y)$ dado en (2.3).

En la siguiente sección vamos a generar imágenes sintéticas con este modelo, lo que además servirá para aclarar y ejemplificar lo visto hasta ahora.

2.2. Micrografías sintéticas

Cuando se hacen ajustes de micrografías reales no se tiene ningún referente para saber si se obtienen buenos resultados. Parece adecuado probar antes el algoritmo sobre imagenes sintéticas, generadas a partir del modelo propuesto en (2.3) y para las cuales hayamos asignado valores concretos a los parámetros del mismo. Para estas, propondremos unos parámetros iniciales ligeramente desviados y ejecutaremos el algoritmo para realizar un ajuste al mismo modelo. Los parámetros a los que converja como resultado serán un refinamiento de los parámetros iniciales, y deberán de ser muy próximos a aquellos con los que hayamos generado la imagen.

2.2.1. Proceso

Este proceso tendrá dos partes claramente diferenciadas:

- Por un lado, y en primer lugar, la creación de imágenes de prueba (en el apéndice B se presenta el código R y una descripción de las funciones que implementa).
- En segundo lugar la implementación y prueba del algoritmo de Levenberg-Marquardt que se propone para el ajuste (código y descripción del mismo en el apéndice C).

Generación de las micrografías sintéticas

La implementación de estas simulaciones seguirá los siguientes pasos,

- **PASO 1:** Asignación de valores a las medias, varianzas, coeficientes de correlación y pesos para definir el modelo que usaremos, considerando la suma de pesos normalizada, según el modelo dado en (2.2).

- **PASO 2:** Generación de una muestra (de tamaño $n = 5000$, predefinido por nosotros) de puntos en el plano, distribuidos según el modelo resultante en el PASO 1. Para tener muestras que sigan una mixtura se generan tantas muestras, de tamaño $\pi_i n$ (con $0 < \pi_i \leq 1$), como componentes tenga la mixtura[4], y siendo la suma de los π_i igual a 1. Si tomamos las observaciones de todas estas muestras como observaciones de una sola, tenemos otra muestra, de tamaño n y siguiendo la mixtura. Los π_i son precisamente el peso de la componente i, que define el número de puntos (observaciones) que corresponden a ella.

  Se realiza la representación gráfica de dicha muestra (n puntos en el plano) en rangos de los ejes de coordenadas definidos por el software. Denotamos los extremos de este rango como i_x y s_x para el eje horizontal (inferior y superior, respectivamente) y i_y y s_y para el vertical.

- **PASO 3:** Reescalaremos la muestra mediante las expresiones

$$\begin{aligned} x_{reescalado} &= n_x \cdot \frac{x - i_x}{s_x - i_x}, \\ y_{reescalado} &= n_y \cdot \frac{y - i_y}{s_y - i_y}, \end{aligned} \tag{2.4}$$

  para que quede contenida en el intervalo $0 < x < n_x$ en el eje horizontal, y en $0 < y < n_y$ en el vertical, donde n_x y n_y serán dos número positivos definidos por nosotros que representarán el número de columnas y filas de pixeles de la imagen sintética. Se dibuja una cuadrícula simulando el pixelaje (cada pixel tendrá un lado de longitud 1 según lo propuesto) y se reescalan tambien las medias (aplicando 2.4) y las desviaciones,

$$\begin{aligned} (\sigma_x)_{reescalado} &= n_x \cdot \frac{\sigma_x}{s_x - i_x} \\ (\sigma_y)_{reescalado} &= n_y \cdot \frac{\sigma_y}{s_y - i_y} \end{aligned} \tag{2.5}$$

[4]Aunque la elección del número de eventos de cada componente es determinista, para tamaños de muestra grandes la habitual aproximación probabilista da resultados similares.

para que sean coherentes con la nueva muestra reescalada. El coeficiente de correlación y peso son parámetros adimensionales con lo que no necesitan ser reescalados.

- **PASO 4:** Realizando el recuento de puntos de la muestra contenidos en cada una de las subdivisiones que nos da la cuadrícula, simularemos el brillo asociado a cada pixel de esta imagen sintética y la matriz de datos correspondiente.

- **PASO 5:** Se define y se suma una constante, α, que simula el offset, a los elementos de la matriz anterior, y en los casos que corresponda, el ruido.

- **PASO 6:** Escritura, en un fichero, de la matriz de datos que representa la micrografía: una matriz de $n_x \times n_y$.

Ajuste de las micrografías sintéticas

Se elige el *Algoritmo de Levenberg-Marquardt* como método de ajuste. Este es una generalización de la optimización por mínimos cuadrados, la cual consiste en obtener el vector de parámetros $\boldsymbol{\beta}$ que minimiza la expresión,

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} (z_i - F(x_i, y_i, \boldsymbol{\beta}))^2, \tag{2.6}$$

acorde con el caso bivariante que nos ocupa. En esta expresión m es el número de pixeles, que son los puntos que tenemos para realizar el ajuste; (x_i, y_i) son las coordenadas de posición de cada pixel, y z_i su brillo. La función F es el modelo propuesto en (2.3), y $\boldsymbol{\beta}$ los parámetros que lo definen (offset, pesos, medias, desviaciones y coeficientes de correlación de cada término de la mixtura).

El algoritmo trabaja siendo iniciado con un $\boldsymbol{\beta}_0$ que se propone. En cada iteración se resuelve la ecuación matricial (remitimos a la referencia [12] para más detalle)

$$(J^T J + \lambda I)\boldsymbol{\delta} = J^T[\boldsymbol{z} - F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\beta})], \tag{2.7}$$

siendo $\boldsymbol{\delta}$ el vector de incógnitas. Este nos da los incrementos para recalcular $\boldsymbol{\beta}$ en cada iteración, de manera que en la j-ésima tendremos un vector de parámetros dado por $\boldsymbol{\beta}_j = \boldsymbol{\beta}_{j-1} + \boldsymbol{\delta}_{j-1}$. Por otro lado, J es una matriz con m filas, siendo éstas el gradiente de F en cada uno de los m puntos.

La implementación del proceso de ajuste (código en el apéndice C) donde se usará este algoritmo seguirá los siguientes pasos:

- **PASO 1:** Lectura del fichero donde se tenga la matriz de datos que representa la micrografía que se quiere ajustar, ya sea sintética o real.
- **PASO 2:** Propuesta de los parámetros con los que se inicia el algoritmo. Cuanto más próximos sean estos a los reales más rápida será la convergencia.
- **PASO 3:** Procesado y preparación de la matriz de datos. Aquí, por una parte, restaremos a todos los elementos de matriz el valor del mínimo, que corresponde aproximadamente al offset. Por otra parte, dividiremos cada uno por la suma de todos, así quedará normalizada.
- **PASO 4:** Aplicamos el ajuste a un modelo como el de la expresión (2.3) con los puntos que resultan del PASO 3. Para esto se hará uso de la librería de R *lm.minpack* (referencia [23]), en concreto, de la función nlsLM() (remito nuevamente al código del apéndice C). Obtenemos los estimadores y datos ajustados.
- **PASO 5:** Realizaremos la operación inversa a la realizada en el PASO 3, sumando el mínimo valor de la matriz de datos original y multiplicando por la suma de todos sus elementos. Así recuperamos unos datos ajustados del mismo orden que los originales.
- **PASO 6:** Escribimos en ficheros los datos (matriz con las mismas dimensiones que la de datos originales) y estimadores ajustados.

2.2.2. Ejemplos en ausencia de ruido

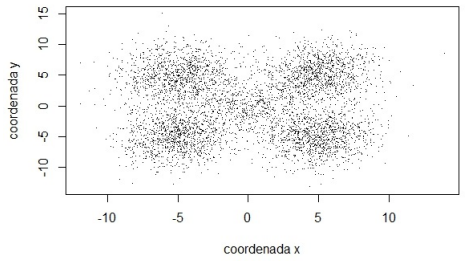
Presentaremos dos ejemplos que simulan estructuras atómicas similares a algunas que podemos encontrar en una micrografía real.

Ejemplo 1: micrografía de 25×20 con 5 normales en ausencia de ruido.

Aquí se pretende simular una de las celdas unidad que se presentan en la micrografía de la figura 1.1.

- **SIMULACIÓN:**

PASO 1-2: Generamos una estructura de este tipo con una muestra de 5000 puntos siguiendo una mixtura de 5 componentes normales, tal y como se puede ver en la figura 2.2.



(a) Muestra original de una mixtura de 5 componentes de parámetros dados en b).

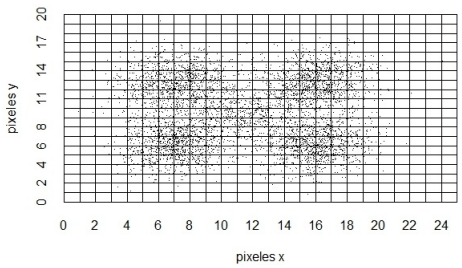
	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	-5	5	2	2.5	0.1	0.225
2	5	5	2	2.5	0.1	0.225
3	-5	-5	2	2.5	0.1	0.225
4	5	-5	2	2.5	0.1	0.225
5	0	0	2	2.0	-0.1	0.100

(b) Parámetros con los que se genera la muestra original

Figura 2.2: *Muestra original donde se observa el rango de representación que se establece por defecto y que aprovecharemos para reescalar la muestra.*

PASO 3: El resultado del reescalado de la muestra se puede ver en la figura 2.3.

PASO 4-5: Realizamos el conteo de puntos en el interior de cada elemento de la cuadrícula y obtenemos así la matriz de datos asociada a esta micrografía sintética (figura 2.4 b)). Se puede representar esto en 3 dimensiones considerando los elementos de matriz, como coordenada dependiente de la posición (x, y) dada por el punto en el centro del pixel (figura 2.4).

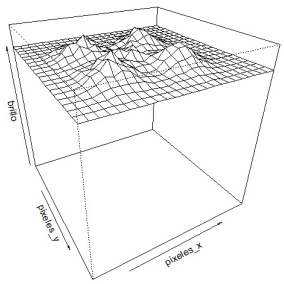


(a) Muestra reescalada con rejilla que simula el pixelaje de la imagen sintética. Contando el número de puntos contenidos en cada subdivisión de la cuadrícula simulamos el brillo de cada pixel.

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	7.113249	12.698193	1.78905	1.632691	0.1	0.225
2	16.058500	12.698193	1.78905	1.632691	0.1	0.225
3	7.113249	6.167427	1.78905	1.632691	0.1	0.225
4	16.058500	6.167427	1.78905	1.632691	0.1	0.225
5	11.585874	9.432810	1.78905	1.306153	-0.1	0.100

(b) Parámetros reescalados de manera que tenemos la muestra contenida en un rango coherente con las coordenadas discretas (filas y columnas) de los pixeles que componen una imagen.

Figura 2.3: *Muestra de la figura 2.2 tras haber sido reescalada.*



(a) Representación gráfica en 3 dimensiones de la muestra una vez discretizada, y donde la coordenada vertical nos da el número de puntos de la muestra original contenidos en cada uno de los elementos de la cuadrícula que simulan el pixelaje.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25
1	500	500	500	500	500	500	501	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500
2	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500
3	500	500	500	500	501	500	502	500	500	500	500	500	500	500	501	500	501	500	503	500	500	500	500	500	500
4	500	500	500	501	503	502	506	503	507	501	500	502	500	502	502	506	508	502	502	500	500	500	500	500	500
5	500	500	501	501	503	513	515	515	511	508	503	503	505	501	510	516	511	517	506	505	501	500	500	500	500
6	501	501	500	505	517	518	525	525	533	512	505	505	505	505	517	531	549	527	522	512	502	500	500	500	501
7	500	500	501	509	513	531	531	550	548	531	506	507	513	522	529	550	552	534	517	507	501	501	500	500	500
8	500	500	503	507	523	546	547	568	536	527	519	506	510	518	547	562	553	533	523	508	502	500	500	500	500
9	500	502	501	509	521	523	545	534	543	522	520	500	516	526	532	543	548	531	518	505	502	500	501	500	500
10	500	500	502	502	509	518	523	535	530	526	527	519	527	522	530	528	519	517	508	504	501	501	500	500	500
11	501	501	501	504	507	511	519	518	517	523	539	526	542	513	519	515	508	509	510	503	502	500	500	500	500
12	500	500	500	502	512	515	521	525	525	525	518	543	531	520	520	525	524	518	507	504	501	500	500	500	500
13	500	501	500	503	519	519	545	551	532	526	522	516	511	525	531	536	546	529	523	510	505	500	500	500	500
14	500	500	501	505	518	532	549	573	541	525	511	515	513	523	551	555	560	537	525	511	502	501	500	500	500
15	500	500	502	507	527	536	562	547	535	525	508	503	507	528	539	555	556	542	510	510	503	500	500	500	500
16	500	500	500	507	515	531	527	539	531	509	500	504	504	515	530	532	540	523	510	504	500	500	500	500	500
17	500	500	500	501	508	513	515	514	513	510	501	502	503	507	508	517	517	513	504	501	501	500	500	500	500
18	500	500	500	502	502	508	506	506	502	502	500	500	503	503	505	509	504	504	501	500	500	500	500	500	500
19	500	500	500	501	501	500	501	501	500	500	500	500	500	501	500	500	502	500	500	500	500	500	500	500	500
20	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	501	500	500	500	500	500	500	500	500	500

(b) Matriz que recoge el conteo de puntos de la muestra original más una constante aditiva (de 500 en este caso) que representa el offset que se tiene en imágenes reales.

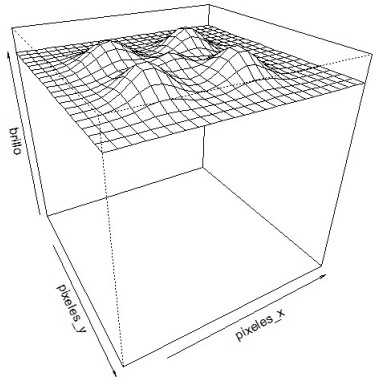
Figura 2.4: *Muestra de la figura 2.2 tras haber sido discretizada.*

- **AJUSTE:**

En un tiempo de ejecución de,

$$t_{ejecucion} = 1,10s \tag{2.8}$$

y con los parámetros iniciales que se muestran en la figura 2.5 b), se realiza el ajuste obteniendo los resultados de la figura 2.5, tanto la superficie ajustada como los parámetros con sus desviaciones típicas. Estos se pueden comparar con los de la figura 2.3, que son los que definían la imagen.



(a) Representación gráfica en 3 dimensiones de los datos ajustados.

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	6	14	1	1	0	0.2
2	17	14	1	1	0	0.2
3	6	5	1	1	0	0.2
4	17	5	1	1	0	0.2
5	12	10	1	1	0	0.1

(b) Estimadores propuestos para iniciar el algoritmo.

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	7.298221	12.604360	1.959985	1.741587	0.02459136	0.23498131
2	15.929183	12.741235	1.802053	1.735207	0.22408093	0.23137458
3	7.140056	6.166381	1.766673	1.671051	0.12786373	0.22518349
4	15.985678	6.109684	1.812084	1.611059	0.07205241	0.22524719
5	11.553358	9.193779	1.440268	1.204280	-0.10545390	0.07749682

(c) Estimadores ajustados para cada una de las 5 normales presentadas.

	error media_x	error media_y	error sigma_x	error sigma_y	error coefCorr	error peso
1	0.05558278	0.05188141	0.05615026	0.05385779	0.04300189	0.008046466
2	0.04799342	0.04725138	0.05038214	0.05025043	0.03757036	0.007142385
3	0.04849896	0.04744919	0.05060651	0.04958808	0.03990331	0.007266313
4	0.04871646	0.04323321	0.04965062	0.04434035	0.03977358	0.007023832
5	0.10414034	0.08077802	0.11193728	0.08300937	0.10629596	0.007545867

(d) Error de los estimadores ajustados para cada una de las 5 normales.

Figura 2.5: *Resultados del ajuste del Ejemplo 1.*

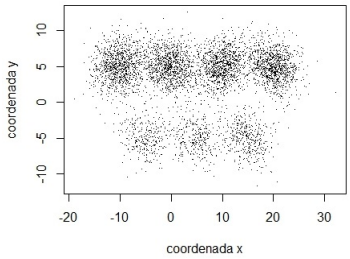
Podemos comparar los valores ajustados de la figura 2.5 c) con los originales dados en la figura 2.3 b) y observar desviaciones máximas entre parámetros de unos 0,4 pixeles (en estos valores se toma el pixel como unidad de longitud). Se observa además que las mayores diferencias se tienen en las desviaciones típicas (remitimos a la sección 2.2.5).

Ejemplo 2: micrografía con 7 componentes en una imagen de 40×20 y en ausencia de ruido

Ahora otro ejemplo con una estructura de átomos diferente.

- **SIMULACIÓN:**

PASO 1-2: Proponemos una distribución de puntos (5000, como en los ejemplos anteriores) como la de la figura 2.6.



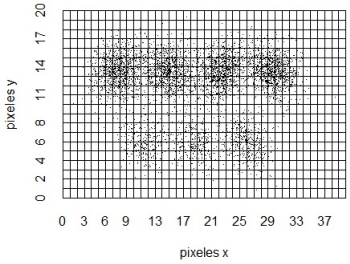
(a) Muestra original de una mixtura de 7 componentes de parámetros dados en b).

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	-10	5	2.5	2	0.10	0.20
2	0	5	2.5	2	-0.10	0.20
3	10	5	2.5	2	0.20	0.20
4	-5	-5	2.5	2	-0.15	0.06
5	5	-5	2.5	2	-0.20	0.06
6	20	5	2.5	2	-0.20	0.20
7	15	-5	2.5	2	-0.20	0.08

(b) Parámetros con los que se genera la muestra original.

Figura 2.6: *Muestra original del Ejemplo 2.*

PASO 3: La reescalamos tal y como se presenta en la figura 2.7.



(a) Muestra de la figura 2.6 reescalada en una rango de 40×20 y con parámetros, tambien reescalados, dados en b).

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	7.905104	13.518063	1.815027	1.526035	0.10	0.20
2	15.165214	13.518063	1.815027	1.526035	-0.10	0.20
3	22.425323	13.518063	1.815027	1.526035	0.20	0.20
4	11.535159	5.887885	1.815027	1.526035	-0.15	0.06
5	18.795268	5.887885	1.815027	1.526035	-0.20	0.06
6	29.685432	13.518063	1.815027	1.526035	-0.20	0.20
7	26.055378	5.887885	1.815027	1.526035	-0.20	0.08

(b) Parámetros con los que se genera la muestra original.

Figura 2.7: *Muestra reescalada del Ejemplo 2.*

PASO 4-5: La discretizamos en una imagen de 40×20 (representación en 3 dimensiones en la figura 2.8).

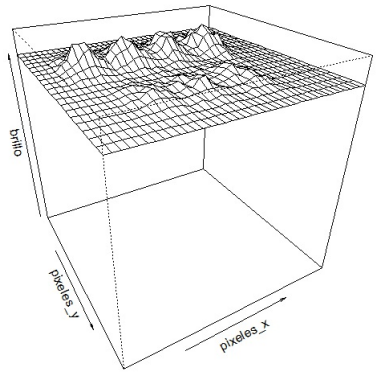


Figura 2.8: *Como en los ejemplos anteriores, presentamos la superficie que generan los puntos asociados a la posición y brillo de cada pixel, habiéndose obtenido mediante discretización de 2.7. La matriz de datos (así como la de pixeles) asociada, esta vez es de* 40×20.

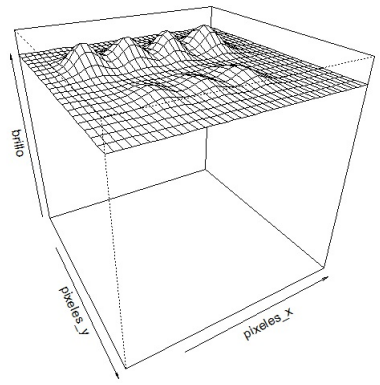
- **AJUSTE:**

En este caso el tiempo de ejecución del ajuste es,

$$t_{ejecucion} = 1,59s \tag{2.9}$$

habiendo propuesto como parámetros para iniciar el algoritmo los mostrados en la figura 2.9 b). Se obtienen los resultados de la figura 2.9, que como en el Ejemplo 1, podemos comparar con los parámetros con los que se propuso la imagen y que se presentan en la figura 2.7 b).

Comparando los valores ajustados, visibles en la figura 2.9 c), con los parámetros que generan la micrografía sintética (figura 2.7 b)), observamos desviaciones menores que 0,2 pixeles (tomando estos como unidad de longitud), aproximadamente. Remito a la sección 2.2.5 para más comentario.



(a) Representación gráfica en 3 dimensiones de los datos ajustados.

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	7	14	1	1	0	0.1
2	17	14	1	1	0	0.3
3	24	14	1	1	0	0.3
4	10	5	1	1	0	0.2
5	17	5	1	1	0	0.1
6	28	12	1	1	0	0.1
7	24	5	1	1	0	0.1

(b) Parámetros propuestos para iniciar el algoritmo.

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	7.871579	13.517765	1.837288	1.497046	0.03345973	0.19059976
2	15.026291	13.494808	1.966299	1.511100	-0.07337796	0.20599638
3	22.447924	13.533190	1.798688	1.581225	0.22953434	0.19577107
4	11.693788	5.820408	1.918649	1.459992	-0.06281338	0.06145987
5	18.989763	5.825222	1.640277	1.425117	-0.16688157	0.05551775
6	29.753636	13.471209	1.876860	1.450589	-0.14256596	0.19914788
7	25.926531	6.012040	1.806039	1.543363	-0.33958763	0.07783538

(c) Estimadores ajustados para cada una de las 7 normales presentadas en la simulación de l ejemplo 2.

	error media_x	error media_y	error sigma_x	error sigma_y	error coefCorr	error peso
1	0.04278336	0.03194915	0.04581745	0.03243442	0.03061416	0.004701043
2	0.04457420	0.03105355	0.05260168	0.03147873	0.02972705	0.005101413
3	0.04066844	0.03310268	0.04595376	0.03351795	0.02846025	0.004757214
4	0.13109819	0.09712796	0.13895694	0.09891049	0.09435642	0.004506503
5	0.11486650	0.09547205	0.12362509	0.09670951	0.09339427	0.004110645
6	0.03954524	0.02932789	0.04220171	0.02980973	0.02828654	0.004488062
7	0.09388662	0.07854493	0.09811861	0.08000115	0.06407243	0.004327022

(d) Error de los estimadores ajustados para cada una de las 7 normales.

Figura 2.9: *Resultados del ajuste del Ejemplo 2.*

2.2.3. Ejemplos que incluyen ruido

El propio fenómeno de detección de partículas lleva siempre asociado, a nivel experimental, un ruido que sigue una distribución de Poisson, cuya densidad de probabilidad viene dada por la expresión

$$p(n) = \frac{N^n \cdot e^{-N}}{n!}, \tag{2.10}$$

donde $p(n)$ es la probabilidad de obtener n como resultado en una realización, y N el promedio de todas las observaciones. En nuestro caso el recuento de puntos en cada región del mallado (esto es la simulación de la intensidad en un pixel).

Se plantea que tenemos dos ruidos que consideraremos independientes y que incluiremos,

- Debe de existir una perturbación (esta deberá ser de Poisson) que afecte a los picos, de forma que en pixeles más brillantes pueda haber más ruido. En lo relativo a la relación señal-ruido, el ruido de tipo Poisson o ruido de disparo, afecta más a los pixeles donde han caído pocos electrones.

- Debemos de tener en cuenta otro ruido que sea proporcional al nivel de offset. Es una realidad que al efectuar los ajustes de contraste que lo introducen, el ruido crece o decrece con este nivel.

En cuanto al primero, parece obvio que el ruido debe de ir asociado a cada pixel en particular, tomando como parámetro N el número de eventos en ausencia de ruido, así los pixeles más brillantes se verán afectados por una perturbación mayor. Para el segundo, se incluye un ruido uniforme tomando como promedio, N, el 2 % del offset que tengamos.

Para introducir la primera componente del ruido, sustituiremos el recuento de puntos en cada región del mallado por una realización de Poisson centrada en ese mismo recuento (que simula la intensidad del pixel). La segunda componente de ruido, relativa al offset, la incluimos sumando una realización de una uniforme con extremos en $\pm 2\,\%$ del offset.

Ejemplo 3: inclusión de ruido en el Ejemplo 1.

- **SIMULACIÓN:**

Si tomamos la imagen desarrollada en el Ejemplo 1 e introducimos el ruido, tenemos una imagen cuyo relieve es como el presentado en la figura 2.10.

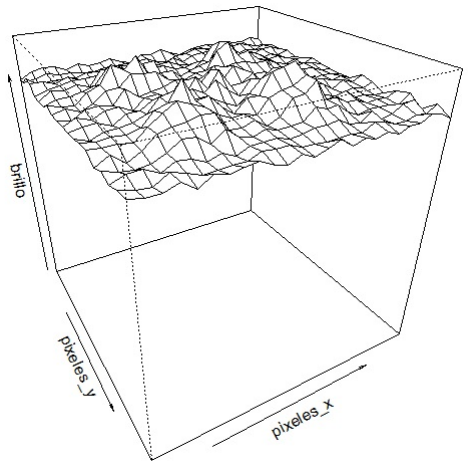


Figura 2.10: *Imagen del Ejemplo 1, habiendo introducido las dos componentes de ruido propuestas.*

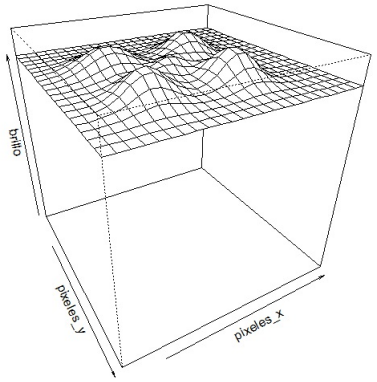
- **AJUSTE:**

En este caso el algoritmo se ejecuta en

$$t_{ejecucion} = 1{,}19\ s\ (20\ iteraciones) \tag{2.11}$$

siendo iniciado con los mismos parámetros que en el Ejemplo 1 y obteniendo los resultados mostrados en la figura 2.11.

Encontramos que las mayores desviaciones (comparando los parámetros ajustados presentados en la figura 2.11 c) con los originales, presentados en 2.3), como en los ejemplos en ausencia de ruido, las encontramos en una de las desviaciones típicas (en torno a 0,4 pixeles). En cuanto a las medias, notamos que tienen desviaciones inferiores a 0,2 pixeles (unidad de longitud). Remitimos a la sección 2.2.5.



(a) Representación gráfica en 3 dimensiones de los datos ajustados.

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	6	14	1	1	0	0.2
2	17	14	1	1	0	0.2
3	6	5	1	1	0	0.2
4	17	5	1	1	0	0.2
5	12	10	1	1	0	0.1

(b) Estimadores propuestos para iniciar el algoritmo.

	media x	media y	sigma x	sigma y	coef. corr	peso
1	7.158916	12.513642	2.218319	1.764587	-0.03963229	0.12871391
2	16.021940	12.851795	1.873632	1.716544	0.20853402	0.11534891
3	7.159776	6.109981	1.794832	1.648126	0.17812208	0.11042010
4	15.842126	6.201089	1.678797	1.603241	-0.04629839	0.11618267
5	11.627136	9.403266	1.429552	1.234245	0.47321718	0.03311062

(c) Estimadores ajustados para cada una de las 5 normales presentadas.

	error media x	error media y	error sigma x	error sigma y	error coef. corr	error peso
1	0.13545240	0.11256466	0.14420650	0.12144092	0.09026237	0.009463169
2	0.14257580	0.12669086	0.13685873	0.12154743	0.10235241	0.009676891
3	0.13421321	0.12029625	0.13042349	0.11671487	0.10136440	0.009220957
4	0.09037933	0.08910991	0.09418083	0.09439952	0.07864424	0.007171378
5	0.28845200	0.23854860	0.33329602	0.26353405	0.21598582	0.009465564

(d) Error de los estimadores ajustados para cada una de las 5 normales.

Figura 2.11: *Resultados del ajuste del Ejemplo 3.*

Ejemplo 4: inclusión de ruido en el Ejemplo 2.

- **SIMULACIÓN:**

Ahora introduciremos las dos componentes de ruido en la imagen del Ejemplo 2 y tenemos la imagen representada en la figura 2.12.

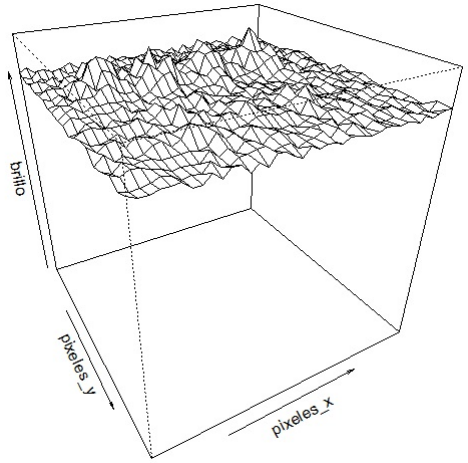


Figura 2.12: *Imagen del Ejemplo 2 habiendo introducido ruido.*

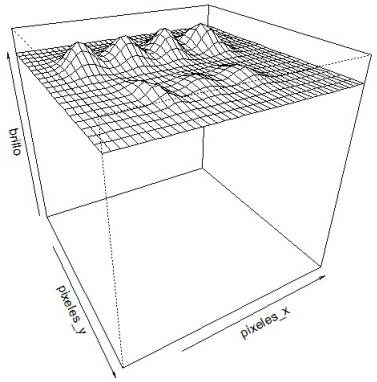
- **AJUSTE:**

En un tiempo de ejecución de,

$$t_{ejecucion} = 3{,}48\ s\ (28\ iteraciones) \tag{2.12}$$

tenemos los resultados presentados en la figura 2.13, iniciando el algoritmo con los parámetros de la figura 2.13 b).

Desviaciones de entre $0{,}1 - 0{,}3$ pixeles (tomando estos como unidad de longitud) de los parámetros ajustados (figura 2.13 c)) respecto de los que se habían propuesto para generar la muestra (figura 2.7). Remitimos a la sección 2.2.5.



(a) Representación gráfica en 3 dimensiones de los datos ajustados.

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	7	14	1	1	0	0.1
2	17	14	1	1	0	0.3
3	24	14	1	1	0	0.3
4	10	5	1	1	0	0.2
5	17	5	1	1	0	0.1
6	28	12	1	1	0	0.1
7	24	5	1	1	0	0.1

(b) Estimadores propuestos para iniciar el algoritmo.

	media x	media y	sigma x	sigma y	coef. corr	peso
1	7.781843	13.526638	1.796852	1.547989	0.15487085	0.07448554
2	15.139504	13.442586	1.919986	1.536830	0.02077757	0.08002337
3	22.619377	13.548352	1.738303	1.609652	0.30748145	0.07733525
4	11.866270	5.879409	1.323485	1.655072	-0.61504993	0.02090064
5	18.930922	5.729193	1.849554	1.340112	0.19388757	0.02251241
6	29.679186	13.396819	1.724137	1.439677	-0.37092850	0.06770285
7	26.176594	5.736395	1.750052	1.327229	-0.24022217	0.02601865

(c) Estimadores ajustados para cada una de las 7 normales presentadas.

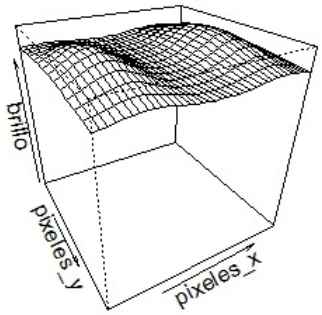
	error media x	error media y	error sigma x	error sigma y	error coef. corr	error peso
1	0.1112522	0.09146410	0.1177979	0.09298504	0.08224489	0.004896629
2	0.1158926	0.08760777	0.1302606	0.08902514	0.08213483	0.005203924
3	0.1026438	0.09057595	0.1126864	0.09165634	0.07277950	0.004863674
4	0.2224236	0.27645721	0.2261769	0.27932333	0.14681228	0.003659613
5	0.3528393	0.24694693	0.3799999	0.24999933	0.25381263	0.004538556
6	0.1057188	0.08611587	0.1113781	0.08718656	0.07337236	0.004392968
7	0.2782985	0.20370442	0.2934415	0.20608231	0.20581434	0.004358500

(d) Error de los estimadores ajustados para cada una de las 7 normales.

Figura 2.13: *Resultados del ajuste del Ejemplo 4.*

2.2.4. Ejemplo con una región de una micrografía real

En el primer capítulo presentamos, en la figura 1.3, y para ilustrar algunas ideas, una pequeña región de la micografía de la figura 1.1, la cual comprendía sólamente dos átomos. Como un primer ensayo sobre una imagen real, y antes de pasar a la implementación del plug-in, probaremos el algoritmo de ajuste sobre ella obteniendo los resultados presentados en la figura 2.14.



(a) Representación gráfica en 3 dimensiones de los datos ajustados.

	media_x	media_y	sigma_x	sigma_y	coefCorr	peso
1	6	8	4	4	0	0.5
2	30	8	4	4	0	0.5

(b) Estimadores propuestos para iniciar el algoritmo.

	X1	X2	X3	X4	X5	X6
1	6.788868	6.205525	4.369697	3.653007	-0.1432107	0.3576805
2	29.319138	6.891211	4.456090	3.961834	-0.1443274	0.3930256

(c) Estimadores ajustados para cada una de las 7 normales presentadas.

	X1	X2	X3	X4	X5	X6
1	0.08107017	0.06636300	0.1246595	0.0998453	0.02617458	0.01800944
2	0.08035790	0.06964532	0.1261351	0.1052015	0.02552864	0.01951970

(d) Error de los estimadores ajustados para cada una de las 7 normales.

Figura 2.14: *Resultados del ajuste de la figura 1.3, siendo* $X1, X2, X3, X4, X5$ *y* $X6$ *la media en x, media en y, desviación en x, desviación en y, coeficiente de correlación y peso, respectivamente, como en casos anteriores.*

En este caso, se procedió de manera similar a como luego se hará sobre micrografías reales. Se proponen unos parámetros iniciales donde visualmente parecen estar los picos, teniendo en cuenta que las coordenadas de posición se definen según las expresiones dadas en 1.2 y que esta imagen tenía $36 \times$

14 pixeles. Con estos parámetros se lanza el algoritmo, y se obtienen los resultados del ajuste, los cuales suponen un refinamiento de los iniciales. Esta metodología es la que se aplicará e implementará en el siguiente capítulo para desarrollar el plug-in.

2.2.5. Análisis de resultados y comentarios

En vista de los resultados, que vienen dados tomando el propio pixel como unidad de longitud, observamos diferencias en el primer decimal, concretamente, y a "grosso modo", de entre 0,1 y 0,4 pixeles, teniendo los mayores errores en los ajustes de las desviaciones típicas. Bien es cierto que para las medias se podría decir que estas diferencias son menores que 0,2, lo que supone una estimación de la precisión máxima con la que podemos determinar el desplazamiento de una columna. Si consideramos que la micrografia de la figura 1.1, por ejemplo, está calibrada tal manera que cada pixel representa 0,017455 nm de la muestra, estaríamos hablando de errores en torno a $3-4$ pm, con lo que no tendrían porqué detectarse desplazamientos inferiores a esta distancia. Sí parece que garantiza la posibilidad de medir desplazamientos por encima de $5-6$ pm.

Por otro lado, es notable el aumento en el tiempo de ejecución en los ejemplos que incluyen ruido.

Se pueden hacer algunos comentarios y puntualizaciones respecto del tratamiento que damos a las micrografías y de los ejemplos propuestos,

- Es habitual en las técnicas de procesamiento de imagen, que el origen para enumerar filas y columnas de pixeles se encuentre en la esquina superior izquierda, de la misma manera en que se enumeran (indexan) los elementos de una matriz en matemáticas. Sin embargo, en ciertas representaciones gráficas (por ejemplo las figuras del tipo 2.2 o 2.3) encontramos el origen en la esquina inferior izquierda. Este cambio en el origen debe tenerse en cuenta cuando se programan determinados bloques del código.

- En gráficas del tipo de las de las figuras 2.14 a) o 2.13 a), cada pixel está representado por las intersecciones de las líneas de la cuadrícula (como un punto), no por las propias subdivisiones de esta.

- La elección de parámetros iniciales debe de hacerse, dentro de los posible, tratando de seleccionarlos próximos a los reales. Esto favorece la convergencia del algoritmo.

- Se ha podido ver en los ejemplos que se presentan unos errores asociados a los parámetros ajustados. Sin embargo no se ha comentado nada respecto de ellos. Esto se debe a que son errores que arroja el software R al hacer el ajuste, pero por desconocimiento del funcionamiento de la rutina que los devuelve, no se ha tratado con ellos.

Capítulo 3

Funcionamiento del Plug-In para DM

En el anterior capítulo se ha desarrollado y puesto a prueba un script que realiza el ajuste de imágenes sintéticas recibiendo la matriz de datos asociada y los estimadores iniciales. Tenemos ahora que hacer una herramienta para DM (código en el apéndice D), que obtenga esta matriz y estos estimadores, directamente de una micrografía que se visualice en este software, y que, tras el ajuste, nos muestre la información resultante.

Comentaremos el funcionamiento del plug-in probándolo sobre la micrografía de la figura 1.1. Así, además de ver los resultados y el efecto que tiene en la imagen, discutiremos el proceso de ejecución del plug-in y las opciones que nos ofrece.

3.1. Uso y resultados: posprocesado de micrografías reales

Cuando abrimos una micrografía en DM encontramos una pantalla como la que vemos en la figura 3.1, sobre la que podemos hacer un zoom y seleccionar una región de interés que se quiera analizar[1] (figura 3.2). Con esto, estamos en disposición de ejecutar el plug-in (menú "custom" en la barra de tareas de DM, una vez que se tenga instalado[2]) y se tendrá la pantalla de la figura 3.3. Si no se ha seleccionado previamente una región de interés, se tomará la micrografía completa como región a analizar. Hay que ser consciente

[1] No se entrará en cómo hacer estas operaciones. Se remite a la ayuda del propio software para ello. Aquí se pretende comentar sólo el funcionamiento del plug-in, no de DM.

[2] En el apéndice A se detalla este proceso.

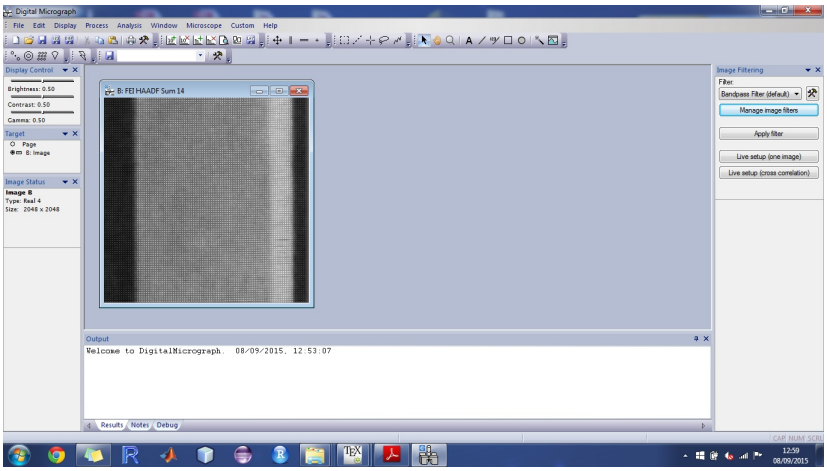


Figura 3.1: *Micrografía de la figura 1.1 visualizada en DM. Se muestra tambien el aspecto de la pantalla principal que tiene dicho programa.*

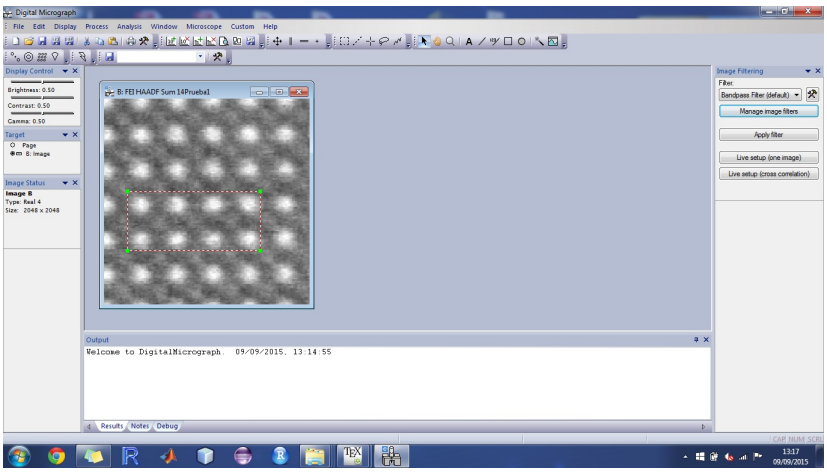


Figura 3.2: *Micrografía de la figura 1.1 habiendo seleccionado una región de interés.*

del elevado tiempo de cálculo que conllevaría dicho análisis (ver sección 3.2).

3.1.1. Asignación de parámetros iniciales

Resulta necesario dar unos valores a los parámetros que definen el modelo, para iniciar el proceso iterativo que nos llevará a un ajuste (refinamiento) de los mismos. Para esto se proponen dos funcionalidades diferentes, ambas basadas en la selección de los picos contenidos en la región de interés elegida.

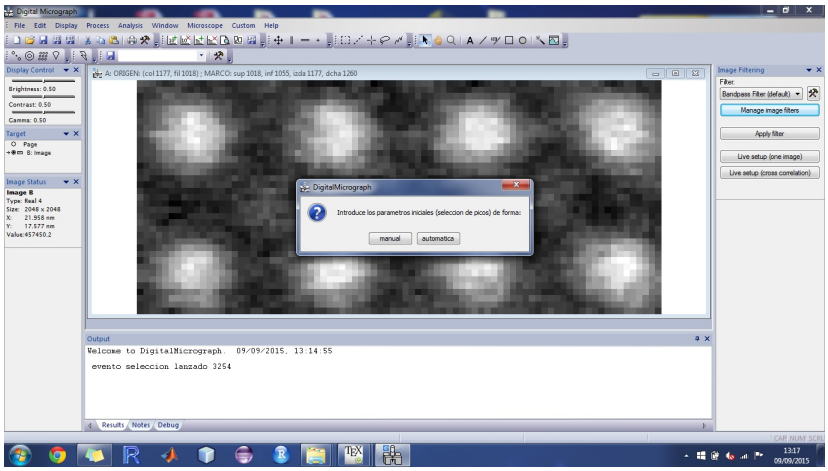


Figura 3.3: *Pantalla tras la ejecución del plug-in.*

Caso de uso: estimación "manual"

Con esta opción tendremos que seleccionar uno a uno los picos, usando regiones de interés rectangulares, hasta tener algo como lo que se muestra en la figura 3.4. Esta selección debe de hacerse con cierto cuidado, puesto que

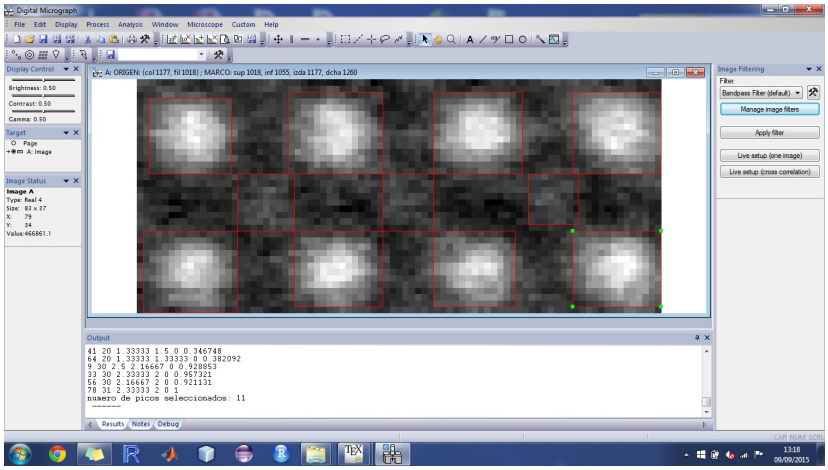


Figura 3.4: *Pantalla tras seleccionar los picos a ajustar (componentes de la mixtura que modeliza la región).*

los parámetros iniciales para cada componente de la mixtura se propondrán a partir de ella. Las medias se elegirán como las coordenadas del pixel más brillante dentro de cada región de interés, las desviaciones típicas como la cuarta parte de los lados del cuadro que encierra cada pico (considerando la

zona central de una normal con una distancia de 2 sigmas desde su centro), el coeficiente de correlación se fija a 0 en todos los casos, y el peso se asigna según la proporcionalidad entre la altura del pico, y la diferencia entre el máximo y mínimo brillo en la región a modelizar. De estas asignaciones, las desviaciones típicas son las que requieren una selección más cuidadosa, debiendo de hacerse de manera que se encuadre toda la zona brillante que corresponde al pico (remito a la figura 3.5 para una muestra).

Cuando ya se han seleccionado todos los picos, se debe de pulsar la tecla "escape" para finalizar el proceso, y se nos pide confirmación para lanzar el algoritmo de ajuste. Veremos como se abre la consola de Windows y al finalizar obtendremos la imagen original con un rombo azul sobre cada pico, y la imagen que generan los datos ajustados (figura 3.5)

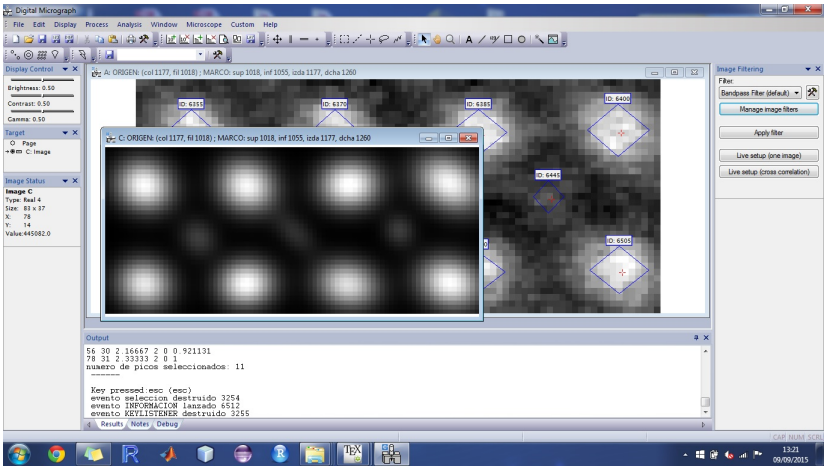


Figura 3.5: *Resultado tras el ajuste de la zona seleccionada en la figura 3.2.*

Caso de uso: estimación "automática"

Para presentar esta otra forma de asignar parámetros iniciales, tomaremos otra región de la imagen (3.6), lanzamos el plug-in, pero esta vez elegiremos la selección "automática". Con esta opción tendremos que seleccionar sólo un elemento (el más próximo a la esquina superior izquierda, que es el origen de coordenadas) de aquellos que tengan periodicidad e intensidad parecidas. Tras la selección de este pico, se nos pedirá la periodicidad con el siguiente, tanto en el eje horizontal como en el vertical. Este dato debe de introducirse en pixeles (como unidad de longitud), lo cual requiere contarlos sobre la imagen. En la figura 3.7, tenemos un ejemplo de como sería esta selección.

Confirmando el lanzamiento del algoritmo de ajuste obtenemos el resultado mostrado en la figura 3.8.

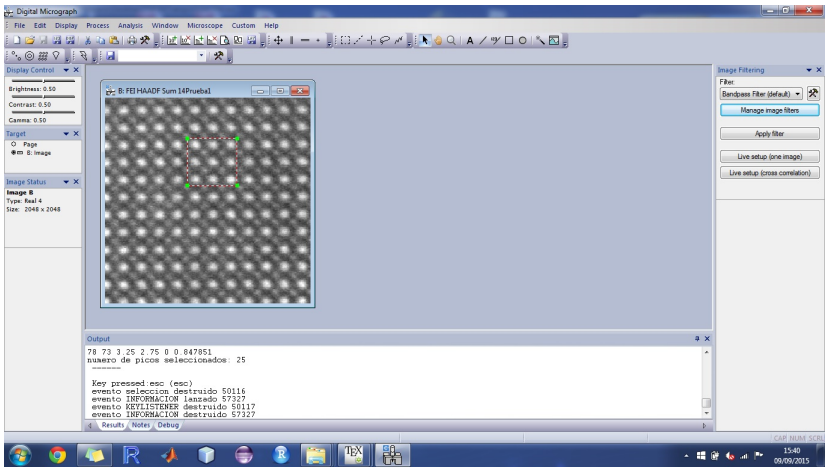


Figura 3.6: *Resultado tras el ajuste de la zona seleccionada en la figura 3.2.*

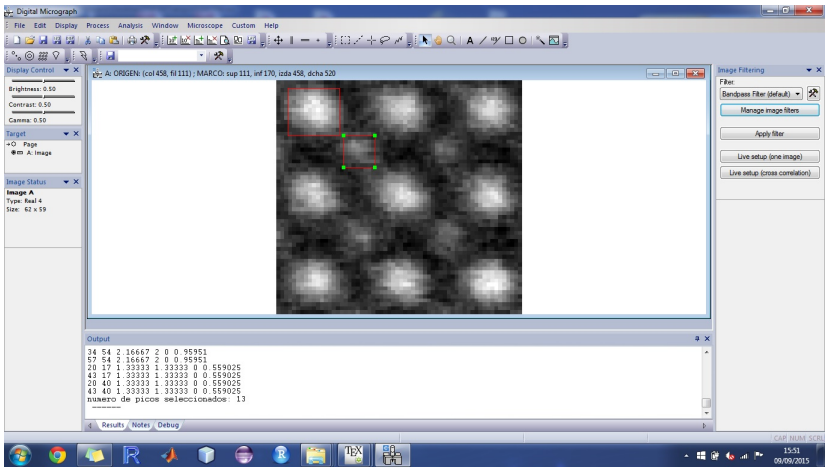


Figura 3.7: *Selección en el modo "automático" para la región de la figura 3.6. Se puede ver que se han consideraado dos conjuntos de elementos con periodicidad y características similares. Uno de ellos el de picos de mayor intensidad y otro el de los picos menos notables, se selecciona el que está máspróximo al origen y cuando procede se les introduce, en este caso, una periodicidad de 23 pixeles tanto en la coordenada horizontal como en la vertical. Este dato tambien de procurarse que sea una buena aproximación.*

En la siguiente sección, y sobre esta imagen resultante, vamos a ver como actuar para obtener distancias entre átomos y parámetros de ajuste.

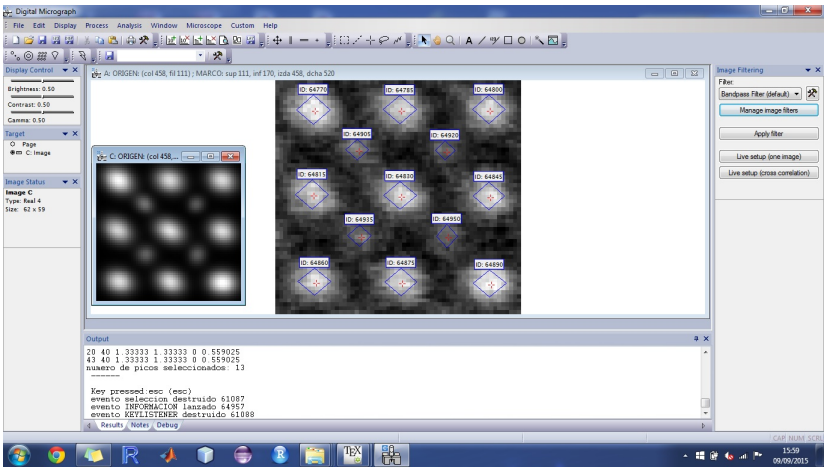


Figura 3.8: *Resultados del ajuste de la región seleccionada en la figura 3.6.*

3.1.2. Obtención de la información

Sobre la imagen producto del ajuste (figura 3.8 o 3.5) se generan objetos que asocian cada pico a una región de interés (rombo azul con tamaño de las diagonales igual a la desviación en x y en y, respectivamente), a los parámetros que definen su normal de ajuste y a un número identificador (que se visualiza en las etiquetas anexas en la parte superior de cada rombo).

Para interactuar con esta imagen, basta añadirle una región de interés tipo puntual o lineal. Si es tipo puntual, y la añadimos en el interior de uno de los rombos, aparecerá una etiqueta sobre dicho rombo con las medias[3] y el peso ajustados. Sobre fuera de los rombos, no tendrá efecto. Si la región de interés que añadimos es lineal, con extremos en el interior de dos rombos diferentes, nos da la distancia entre las medias correspondientes. En otro caso, veremos la linea etiquetada con la palabra "error". En la figura 3.9 vemos un ejemplo de estas utilidades.

Es necesario comentar, por una lado, que en la ventana de "output" tambien se presentan los resultados simultáneamente. Por otra parte, para los parámetros ajustados se usa el pixel como unidad de longitud, no así con las distancias, que se presentan en las unidades en las que esté calibrada la imagen original.

[3]Se presentan medias locales, esto es, respecto del origen de la región seleccionada para el ajuste, y globales, las cuales corresponden a medidas tomadas respecto del origen de la micrografía completa.

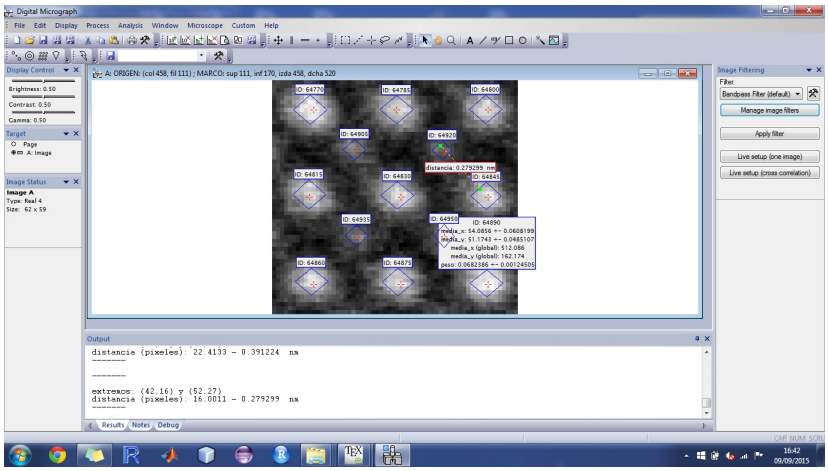


Figura 3.9: *Ejemplo de acción sobre la imagen resultante al añadirle una región de interés puntual o lineal.*

3.1.3. Conclusiones y resultados

Hagamos un breve análisis de la información obtenida. Lanzamos un nuevo ajuste, de una región algo mayor, usando el método "manual" de selección, que aunque más incómodo, nos da unos parámetros iniciales más precisos. Obtenemos así la imagen mostrada en la figura 3.10 donde se tiene una medida estimada, entre dos de los átomos de la parte central, de 0,390755 nm. Este valor es muy próximo al experimental de 0,390 nm.

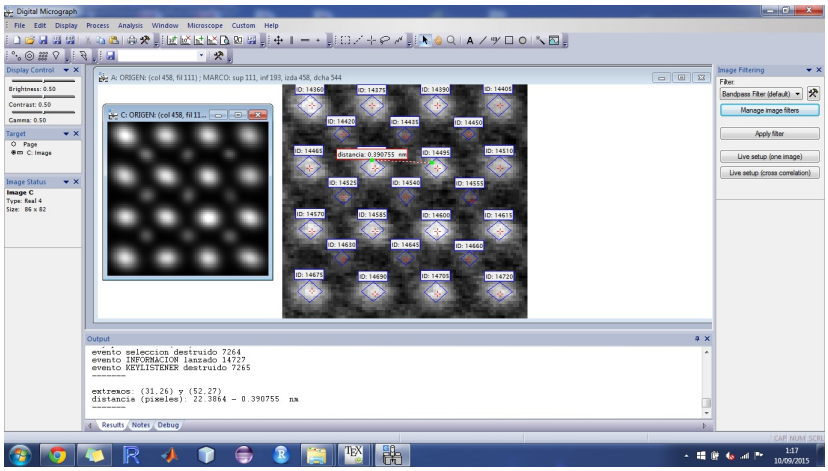
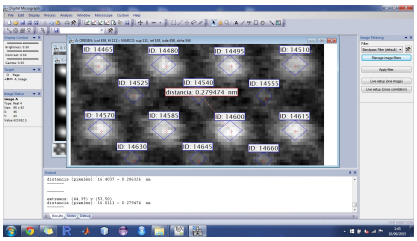


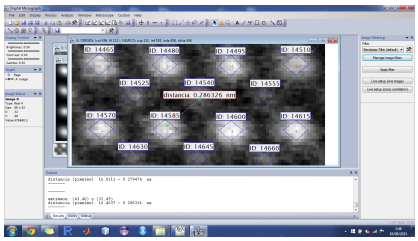
Figura 3.10: *Nuevo ajuste sobre el que se realizan pruebas de medida.*

Podemos probar tambien a tomar medidas, por ejemplo, entre los átomos

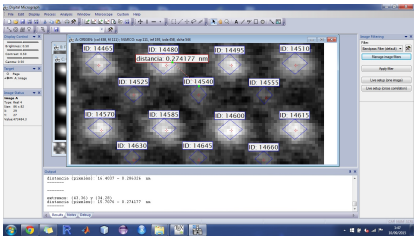
de los vértices y el central en la celda unidad que se ve en medio de la imagen (figura 3.11).



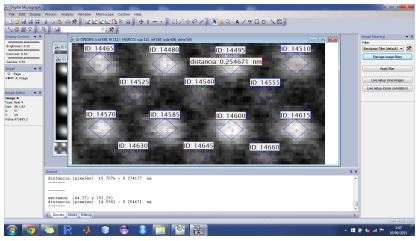
(a) Medida estimada de 279 pm.



(b) Medida estimada de 286 pm.



(c) Medida estimada de 274 pm.



(d) Medida estimada de 255 pm.

Figura 3.11: *Distancias entre los átomos de los vértices y el central en una celda unidad. Se deduce de las medidas una mayor proximidad entre el átomo central y el situado en la esquina superior derecha.*

3.2. Limitaciones actuales y propuesta de desarrollo

Se han observado algunos errores de ejecución o de ajuste en algunas situaciones, en general, en regiones mayores que las aquí tratadas, así como un elevado tiempo de cálculo en estos casos. Básicamente se diría que estos fallos vienen dados por dos cuestiones. La primera, por una mala aproximación de los parámetros iniciales, y la segunda, en casos en los que los picos se presentan demasiado difusos, o tenemos presencia de pixeles con intensidad notable en los bordes.

Si hablamos de la primera, el método de selección de picos que llamamos "manual", haciendo un buen encuadre de los mismos, ofrece mejores estimaciones para los parámetros, puesto que se realizan las asignaciones para cada pico en particular. Esto se traduce, además, en mayor velocidad de convergencia. En cuanto a la segunda, en algunas zonas encontramos que, los picos de menor intensidad, se ven demasiado planos y difusos llegando prácticamente a solaparse con sus vecinos. En estas situaciones se puede llegar a ajustes en los que las normales que los modelicen tengan grandes varianzas o altos coeficientes de correlación que no resultan acordes con lo esperado. Por otro lado, los pixeles en los bordes de la región a analizar pueden provocar deformaciones en el ajuste de las columnas de átomos más próximas, razón por la cual resulta conveniente realizar una buena selección de la región de interés, tratando de no incluir más pixeles de los necesarios, más aún si se observan con cierta intensidad.

Se desprende, incluso sólo de la lectura de este capítulo, que el proceso de selección de picos puede resultar algo tedioso. Se debe de hacer para cada uno, con cierto cuidado, y no admite corrección sobre la selección que se ha hecho, por lo que, ante una equivocación habría que repetir todo el proceso. Si además se ha utilizado la herramienta, es inmediato pensar en como mejorar esto. Con este fin, de hecho, se implementó el método de selección que denominamos "automático", pero aprovechar la periodicidad de la red no da buenas estimaciones iniciales. La solución que planteamos, y que supondría una evolución del plug-in es incluirle un detector de picos, adecuado para el caso, que fuera capaz de reconocerlos en la micrografía y asignar valores aproximados a los parámetros iniciales.

Apéndice A

Instrucciones para la instalación, detalles técnicos y de funcionamiento.

El plug-in ha sido desarrollado sobre Windows 7, usando la versión 3.0.1 del software de tratamiento de datos *R* (open source) y la 2.31.734.0 del software de procesamiento de imagen *Digital Micrograph* (desarrollado por la compañía *Gatan* y disponible en su web).

Está compuesto por cuatro scripts. Tres de ellos, contienen código del lenguaje de scripting de DM y en ellos se codifican las acciones sobre la imagen, como puede ser la obtención de la matriz de datos asociada, lectura de los datos ajustados, implementación de los "listeners" que controlan la interacción del usuario con la imagen... El cuarto script codifica, en lenguaje R, una implementación del algoritmo de Levenberg-Marquardt (función *nlsLM()* de la librería *lm.minpack*).

Los archivos mencionados se han denominado como:

- tfmAlgLMPlugIn_classListenerAccion.s (DM-script), pero nos referiremos a él como "classListener". En éste se implementa el listener que ejecuta las acciones para obtener la información en la imagen resultante tras la ejecución (acciones sobre los rombos que se generan).
- tfmAlgLMPlugIn_classMancha.s (DM-script), pero nos referiremos a él como "classMancha". Esta es la clase que genera los objetos que asocian los rombos, los parámetros de cada normal y la id que los identifica.
- tfmAlgLMPlugIn_mainV2.s (DM-script), pero nos referiremos a él como "main". Archivo principal de la herramienta.

- algoritmoLM_V5_DM_Definitivo.R (R-script), pero nos referiremos a él como "Ralgoritmo". Archivo en el que se procesa y se ajusta la imagen original.

De estos, es importante no cambiar el nombre del R-script porque el plug-in se refiere a él internamente mediante este nombre.

La instalación se lleva a cabo siguiendo estos pasos:

- Por un lado, es evidente que se tiene que tener instalado R, y con el paquete *lm.minpack* cargado, el cual no viene por defecto al instalar el software.

- En la carpeta donde se instala el software (habitualmente, como en tantos otros programas, es "C:\Archivos de programa\R") encontramos una carpeta llamada "\bin" que contiene un ejecutable llamado "Rscript". La ruta de este ejecutable debe de incluirse como variable de entorno del sistema operativo, puesto que es el que ejecuta R-scripts desde la consola y necesitamos tener útil esta función para que el plug-in pueda hacerlo.

- Debemos de tener en el escritorio una carpeta que denominaremos "plugInDM" que contenga el archivo "Ralgoritmo". Esta carpeta se utilizará además para el intercambio de datos entre scripts, vía archivos de texto plano. En ella, tras un ajuste quedarán cuatro ficheros que contienen la matriz de datos original, los estimadores iniciales, la matriz de datos ajustada y los estimadores ajustados. Estos ficheros serán borrados en la siguiente ejecución.

- Instalaremos los scripts "classMancha" y "classListener" como librería en DM (en el menú del software File>Install script file>as library).

- Instalaremos el script "main" como plug-in en DM (en el menú File>Install script)

En cuanto a detalles de funcionamiento debemos mencionar que se van viendo mensajes en la ventana de output de DM. Algunos son meramente informativos sobre la ejecución del código, pero otros son interesantes a nivel experimental. En concreto, cuando se realiza la selección de picos para asignar parámetros iniciales, podemos ir viendo los parámetros que se van estimando para cada pico (en el siguiente orden: mediax, mediay, sigmax, sigmay, coeficiente de correlación y peso) y el número de picos que se han

seleccionado. Esto resulta interesante puesto que no se admite la posibilidad de corrección de la región de interés que selecciona uno de ellos. Si se intenta modificar la misma, se añadiría como un nuevo pico. Por otro lado, cuando tras una ejecución visualizamos distancias y resultados, tenemos una información más amplia en la ventana de output.

Apéndice B

R-Script para generar imágenes

B.1. Descripción

Script que implementa las funciones que desarrollan las diversas etapas del proceso de generación de imágenes sintéticas. Se pueden ejecutar unas u otras funciones según lo que se quiera presentar.

- **Funciones "paramEjemploX()" y "paramManual()":** En estas funciones se definen explícitamente los parámetros y el número de pixeles usados, para generar las muestras de puntos de las que partimos para hacer las imágenes sintéticas. Puede resultar poco técnico y repetitivo codificarlo así, pero resulta cómodo para reproducir las mismas imágenes en cada ejecución. En el caso de querer asignar los parámetros en el momento, tenemos la función "paramManual()" que permite introducirlos por teclado.

- **Función "defineParametros()":** Aquí definimos otros parámetros que son comunes en los diversos ejemplos que propusimos. En concreto, el offset que aplicamos, el tamaño de la muestra de puntos, los nombres de los ficheros en los que escribiremos y el número de componentes de la mixtura. Éste último, no resulta necesario, pero se incluye por comodidad.

- **Función "generaMuestra()":** Con ésta se crea una tabla de datos con dos columnas y 5000 filas (tamaño de la muestra). Las columnas corresponden a coordenadas x e y (continuas) de dichos puntos. Además los representa y devuelve dicha tabla.

  Se puede observar que el peso de cada componente sirve para definir la proporción de esos 5000 puntos que se van a usar para representar cada componente de la mixtura.

- **Función "reescala()":** Bloque de código que recibe la tabla de datos obtenida en la ejecución de "generaMuestra()", y la reescala en un rango tal que $0 < x < numero\ de\ columnas\ de\ pixeles$ y $0 < y < numero\ de\ filas\ de\ pixeles$. En ésta se implementan las expresiones 2.4 y 2.5, que se usan tambien para reescalar de la misma forma los parámetros de la mixtura. Devuelve otra tabla de datos con las nuevas coordenadas de puntos, los estimadores reescalados y representa la muestra en el nuevo rango.

- **Función "discretizaMuestra()":** Se recibe la nueva tabla de datos y se discretiza la muestra contando los puntos que tenemos en cada división de la cuadrícula que representa el pixelaje (representada en la función anterior). Los resultados de este conteo se devuelven en una matriz con el mismo número de filas y columnas que la matriz de pixeles.

- **Función "plotea3d_V2()":** En estas líneas se recibe la matriz (de frecuencias) fruto de la discretización anterior y se representan sus elementos en la coordenada vertical. El plano horizontal representa el plano de la imagen con su ancho y alto como coordenadas continuas. Se tienen gráficos del tipo de los que encontramos, por ejemplo, en las figuras 1.3 o 2.4.

- **Función "introduceRuido()":** Según la imagen que queramos generar se llama a esta función para incluir el ruido correspondiente. Vemos claramente las dos componentes de ruido incluidas, una proporcional al offset y otra a los picos. Devuelve una nueva matriz que le añade el ruido a la obtenida en "discretizaMuestra()".

- **Función "escribeDatos()":** Cuando es necesario, o deseado, se pueden escribir en ficheros de texto plano (txt) la matriz de frecuencias (con o sin ruido, y con o sin offset) y los estimadores ya reescalados. Así tenemos los datos y parámetros de la simulación como los recibimos de las imágenes reales vía Digital Micrograph.

B.2. Código

```
###################################
# generar imagenes
# version: 'V5.4'.
###################################

library(MASS)#para funcion mvrnorm
```

```
##############################################
#funcion para definir parametros de muestra y de imagen
##############################################3
paramEjemplo1 <- function(){
  NPIXX <<- 25
  NPIXY <<- 20
  #dataframe para los estimadores, donde el número de filas
  #es el número de componentes de la mixtura
  estimadoresIni <<- data.frame(media_x=numeric(0),
                                media_y=numeric(0),
                                sigma_x=numeric(0),
                                sigma_y=numeric(0),
                                coefCorr=numeric(0),
                                peso=numeric(0))

  #### EJEMPLO 1######
  #definimos aqui los parámetros de cinco componentes
  estimadoresIni[1,1] <<- -5
  estimadoresIni[1,2] <<- 5
  estimadoresIni[1,3] <<- 2
  estimadoresIni[1,4] <<- 2.5
  estimadoresIni[1,5] <<- 0.1
  estimadoresIni[1,6] <<- 0.225

  estimadoresIni[2,1] <<- 5
  estimadoresIni[2,2] <<- 5
  estimadoresIni[2,3] <<- 2
  estimadoresIni[2,4] <<- 2.5
  estimadoresIni[2,5] <<- 0.1
  estimadoresIni[2,6] <<- 0.225

  estimadoresIni[3,1] <<- -5
  estimadoresIni[3,2] <<- -5
  estimadoresIni[3,3] <<- 2
  estimadoresIni[3,4] <<- 2.5
  estimadoresIni[3,5] <<- 0.1
  estimadoresIni[3,6] <<- 0.225

  estimadoresIni[4,1] <<- 5
  estimadoresIni[4,2] <<- -5
  estimadoresIni[4,3] <<- 2
  estimadoresIni[4,4] <<- 2.5
  estimadoresIni[4,5] <<- 0.1
  estimadoresIni[4,6] <<- 0.225

  estimadoresIni[5,1] <<- 0
  estimadoresIni[5,2] <<- 0
  estimadoresIni[5,3] <<- 2
  estimadoresIni[5,4] <<- 2
  estimadoresIni[5,5] <<- -0.1
  estimadoresIni[5,6] <<- 0.1
}

###################################
paramEjemplo2 <- function(){
  NPIXX <<- 40
  NPIXY <<- 20
  #dataframe para los estimadores, donde el número de filas
  #es el número de componentes de la mixtura
  estimadoresIni <<- data.frame(media_x=numeric(0),
                                media_y=numeric(0),
                                sigma_x=numeric(0),
```

```
                              sigma_y=numeric(0),
                              coefCorr=numeric(0),
                              peso=numeric(0))

  #### EJEMPLO 3######
  #definimos aqui los parámetros de cinco componentes
  estimadoresIni[1,1] <<- -10
  estimadoresIni[1,2] <<- 5
  estimadoresIni[1,3] <<- 2.5
  estimadoresIni[1,4] <<- 2
  estimadoresIni[1,5] <<- 0.1
  estimadoresIni[1,6] <<- 0.2

  estimadoresIni[2,1] <<- 0
  estimadoresIni[2,2] <<- 5
  estimadoresIni[2,3] <<- 2.5
  estimadoresIni[2,4] <<- 2
  estimadoresIni[2,5] <<- -0.1
  estimadoresIni[2,6] <<- 0.2

  estimadoresIni[3,1] <<- 10
  estimadoresIni[3,2] <<- 5
  estimadoresIni[3,3] <<- 2.5
  estimadoresIni[3,4] <<- 2
  estimadoresIni[3,5] <<- 0.2
  estimadoresIni[3,6] <<- 0.2

  estimadoresIni[4,1] <<- -5
  estimadoresIni[4,2] <<- -5
  estimadoresIni[4,3] <<- 2.5
  estimadoresIni[4,4] <<- 2
  estimadoresIni[4,5] <<- -0.15
  estimadoresIni[4,6] <<- 0.06

  estimadoresIni[5,1] <<- 5
  estimadoresIni[5,2] <<- -5
  estimadoresIni[5,3] <<- 2.5
  estimadoresIni[5,4] <<- 2
  estimadoresIni[5,5] <<- -0.2
  estimadoresIni[5,6] <<- 0.06

  estimadoresIni[6,1] <<- 20
  estimadoresIni[6,2] <<- 5
  estimadoresIni[6,3] <<- 2.5
  estimadoresIni[6,4] <<- 2
  estimadoresIni[6,5] <<- -0.2
  estimadoresIni[6,6] <<- 0.2

  estimadoresIni[7,1] <<- 15
  estimadoresIni[7,2] <<- -5
  estimadoresIni[7,3] <<- 2.5
  estimadoresIni[7,4] <<- 2
  estimadoresIni[7,5] <<- -0.2
  estimadoresIni[7,6] <<- 0.08
}
###################################
paramManual <- function(){
  #editamos/asignamos MANUALMENTE los estimadores/parámetros...
  estimadoresIni <<- data.frame(media_x=numeric(0),
                              media_y=numeric(0),
                              sigma_x=numeric(0),
                              sigma_y=numeric(0),
```

```
                                coefCorr=numeric(0),
                                peso=numeric(0))

  while(sum(estimadoresIni[,6])!=1){
    estimadoresIni <<- edit(estimadoresIni)
    if(sum(estimadoresIni[,6])!=1){
      print("la suma de pesos debe de ser 1. Vuelve a introducirlos.")
    }
  }

  #asignacion MANUAL  de parametros de la imagen digital
  nPixeles <- data.frame(nPixeles_x=numeric(0),nPixeles_y=numeric(0))
  nPixeles <- edit(nPixeles)
  NPIXX <<- as.numeric(nPixeles[1,1])#numero de columnas de pixeles
  NPIXY <<- as.numeric(nPixeles[1,2])#numero de filas de pixeles
}

####################################3
defineParametros <- function(){
  txtDatosOUT <<- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\datosOUT.txt"
  txtEstimadoresOUT <<- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\estimadoresOUT.txt"
  txtFactorCalidad <<- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\factorCalidadOUT.txt"

  #numero de normales(grupos) considerados
  G <<- nrow(estimadoresIni)

  #número de elementos para generar la muestra
  NMUESTRA <<- 5000

  #valor sobre el que se ‘‘montará’’ la mixtura...
  #...el nombre ‘‘offset’’ no me lo coge... está reservado para algo...
  offs <<- 500
}#...fin de la funcion ‘‘defineParametros’’


##########################################
#funcion para generar (y plotear) la muestra normal 2-d
##########################################
generaMuestra <- function(){
  #definimos variables
  mu <- matrix(nrow=G,ncol=2)
  sigmaArray <- array(dim=c(2,2,G))

  #‘‘llenamos’’ los arreglos
  i <- numeric(0)
  for(i in 1:G){
    mu[i,] <- c(estimadoresIni[i,1],estimadoresIni[i,2])
    sigmaArray[,,i] <- matrix(c(estimadoresIni[i,3]^2,
                                estimadoresIni[i,5]*estimadoresIni[i,4]*estimadoresIni[i,3],
                                estimadoresIni[i,5]*estimadoresIni[i,4]*estimadoresIni[i,3],
                                estimadoresIni[i,4]^2),2,byrow=T)
  }

  #genera muestra normal con 5000 puntos
  set.seed(1)
  muestra <- mvrnorm(n=NMUESTRA*estimadoresIni[1,6], mu[1,], sigmaArray[,,1])

  #...si hay más de una normal, entra en el if y añade las demás...
  if(G>1){
    i <- 0
    for(i in 2:G){
      set.seed(i)
```

```
      muestra2 <- mvrnorm(n=NMUESTRA*estimadoresIni[i,6],mu[i,],sigmaArray[,,i])
      muestra <- rbind(muestra,muestra2)
      #muestra <- rbind(muestra, mvrnorm(n=NMUESTRA*estimadoresIni[i,6],mu[i,],sigmaArray[,,i]))
    }
  }

  #ploteamos la muestra generada
  plot(muestra, pch='.', xlab="coordenada x", ylab="coordenada y")

  return(muestra)
}


#############################################################
#funcion para reescalar (y plotear) muestra y estimadores según el pixelaje
#############################################################
reescala <- function(muestraOriginal){

  #definimos y dimensionamos la nueva muestra reescalada...
  muestraRees <- matrix(nrow=nrow(muestraOriginal),ncol=ncol(muestraOriginal))

  #### REESCALAMOS LA MUESTRA ###################
  #definimos rangos en la imagen analogica usando los parametros del gráfico
  xmin <- par()$usr[1]
  xmax <- par()$usr[2]
  ymin <- par()$usr[3]
  ymax <- par()$usr[4]

  muestraRees[,1] <- NPIXX*( (muestraOriginal[,1]-xmin)/(xmax-xmin) )
  muestraRees[,2] <- NPIXY*( (muestraOriginal[,2]-ymin)/(ymax-ymin) )

  ##### PLOTEAMOS LA NUEVA MUESTRA REESCALADA ##############
  par(tck=1, lab=c(NPIXX,NPIXY,1), yaxp=c(0,NPIXY,1), xaxp=c(0,NPIXX,1))
  plot(muestraRees, pch='.', xlab="pixeles x", ylab="pixeles y")

  ##### REESCALAMOS ESTIMADORES ##################
  #definimos los estimadores reescalados haciendo una copia de los originales...
  estimadoresIniRees <- estimadoresIni

  #reescalamos la media y desviación en x
  estimadoresIniRees[,1] <- NPIXX*((estimadoresIni[,1]-xmin)/(xmax-xmin))
  estimadoresIniRees[,3] <- NPIXX*((estimadoresIni[,3])/(xmax-xmin))
  #reescalamos media y desviación en y
  estimadoresIniRees[,2] <- NPIXY*((estimadoresIni[,2]-ymin)/(ymax-ymin))
  estimadoresIniRees[,4] <- NPIXY*((estimadoresIni[,4])/(ymax-ymin))

  #cargamos muestra y estimadores reescalados en una lista para devolver
  reescalados <- list(muestraRees,estimadoresIniRees)

  return(reescalados)
}


################################################
#funcion para obtener las frecuencias absolutas
################################################
discretizaMuestra <- function(muestraReesc){

  frecAbsPorPix <- matrix(nrow=NPIXY,ncol=NPIXX)

  #bucle para recorrer filas/pixeles de la matriz
  for(fil in 1:(NPIXY)){
```

```
    #bucle para recorrer columnas/pixeles de la matriz
    for(col in 1:(NPIXX)){
      #las condiciones en el which, para el eje vertical, tienen en cuenta que
      #el extremo superior del rango en y coincide con la posicion (1,1) de la
      #matriz. Así gráfico y matriz tienen coherencia visual
      frecAbsPorPix[fil,col] <- length(which( muestraReesc[,1]>=(col-1)&
                                              muestraReesc[,1]<(col)&
                                              muestraReesc[,2]<(NPIXY-(fil-1))&
                                              muestraReesc[,2]>=(NPIXY-fil)
                                              ))
    }
  }


  return(frecAbsPorPix)
}


################################################
#funcion para graficar una matriz de frecuencias
################################################
plotea3d_V2 <- function(frecuencias){
  frecuencias <- as.matrix(frecuencias)

  pixeles_x <- seq(0.5,ncol(frecuencias)-0.5,by=1)
  pixeles_y <- seq(0.5,nrow(frecuencias)-0.5,by=1)

  eventos <- matrix(nrow=nrow(frecuencias),ncol=ncol(frecuencias))
  eventos <- frecuencias[pixeles_y+0.5,pixeles_x+0.5]

  persp(pixeles_y,pixeles_x,eventos, zlab="brillo", zlim=c(0,max(frecuencias)), phi=30, theta=60)

}


##############################
#funcion para introducir ruido
##############################
introduceRuido <- function(frecuencias){

  #ruido de fondo proporcional al offset. En este caso, el 2% de dicho offset
  ruidoFondo <- (2/100)*offs

  for(fil in 1:NPIXY){
    for(col in 1:NPIXX){
      #añadimos el ruido de poisson proporcional al brillo de cada pixel
      #y el asociado al offset
      lambda <- frecuencias[fil,col]
      frecuencias[fil,col] <- (rpois(1,lambda))+runif(1,min=-ruidoFondo,max=ruidoFondo)
    }
  }

  return(frecuencias)
}


################################
#funcion que escribe datos en txt
################################
escribeDatos <- function(frecuencias){
  write.table(frecuencias, txtDatosOUT)
  write.table(estimadoresIniReescalados, txtEstimadoresOUT)
```

```
}



########################################
################## MAIN ################
########################################

#DEFINIMOS PARAMETROS
#...definimos los ‘‘estimadoresIni’’ y pixelaje de uno u otro ejemplo...
#paramEjemplo1()
paramEjemplo2()
#paramManual()

#...y demás variables y constantes que usaremos...
defineParametros()


#GENERAMOS LA MUESTRA
muestra <- generaMuestra()

#REESCALAMOS MUESTRA Y ESTIMADORES ORIGINALES DE ACUERDO CON LOS PIXELES QUE TENEMOS
reesc <- reescala(muestra)
muestraReescalada <- as.data.frame(reesc[1])
estimadoresIniReescalados <- as.data.frame(reesc[2])

#DISCRETIZAMOS LA MUESTRA Y OBTENEMOS FRECUENCIAS DE APARICION POR PIXEL
matrizFrecuencias <- discretizaMuestra(muestraReescalada)
#...añadimos offset...
#...y ploteamos...
plotea3d_V2(matrizFrecuencias+offs)

#INTRODUCIMOS RUIDO
#introducimos ruido de poisson...
matrizFrecuenciasNoise <- introduceRuido(matrizFrecuencias)
plotea3d_V2(round(matrizFrecuenciasNoise+offs))


#ESCRIBIMOS DATOS Y PARAMETROS
escribeDatos(round(matrizFrecuenciasNoise+offs))
```

Apéndice C

R-Script para ajuste de imágenes sintéticas

C.1. Descripción

Script que implementa las funciones para desarrollar el ajuste de las imágenes sintéticas.

- **Funciones "estimIniEjemploX()" y "estimIniManual()":** Aquí se definen los parámetros con los que se inicia el algoritmo. En "estimIniEjemploX()" se tienen predefinidos los usados para los ejemplos descritos en el trabajo y si se quieren proponer en el momento, la función "estimIniManual()" nos ofrece la posibilidad. Se llama a una u otra función dependiendo de lo que se quiera.
- **Funcion "datosEntrada()":** En esta función se definen el nombre de los archivos de texto de donde leeremos la información, el offset, la matriz de datos y los parámetros con los que se ha generado la imagen sintética. Interesará acceder a estos para compararlos con los resultados del ajuste.
- **Funcion "convierteMatrizATabla()":** Se recibe una matriz y se reescribe como tabla ("dataframe") de tres columnas (coordenada x, coordenada y, brillo). Resulta necesaria porque R ejecuta ciertas funciones, como "nlsLM()" (implementa el algoritmo de Levenberg-Marquardt), para objetos de este tipo.
- **Funciones "procesaTablaEntrada()":** En este bloque se tiene como argumento de entrada una tabla, en concreto la generada en la función "convierteMatrizATabla()". Se procesan los datos restando el

offset (definido en "datosEntrada()" como el mínimo valor de la matriz de datos), y normalizándolos. Para esto último se procede dividiendo cada brillo por la suma de los mismos, con lo que se normaliza tal que la suma de todos los brillos sea igual a 1.

- **Funciones "algoritmoLMLibreria()":** Hacemos uso de la función "nlsLM()", de la librería "lm.minpack", para realizar el ajuste. Esta librería contiene diversas instrucciones para ajustes de funciones no lineales. Una de ellas, la mencionada, desarrolla el algoritmo de Levenberg-Marquardt.

  Este bloque de código actúa escribiendo, de manera dinámica, la fórmula que describe la mixtura de normales en cada caso e incuyendo otro offset como grado de libertad. Este será una corrección para el que lleve la imagen original. Con la fórmula completa se procede al ajuste y a la obtención de resultados, cargados en un lista que es la variable que devuelve (habiendo recibido la tabla de datos obtenida en "procesaTablaEntrada()").

- **Funciones "procesaTablaSalida()":** En esta función se recibe una tabla de datos (la obtenida tras el ajuste) y se devuelve la matriz de los datos ajustados. La tabla se procesa de manera inversa a como se hizo en "procesaTablaEntrada()", sumando el offset que teníamos en la matriz de datos orginal, y multiplicando por la suma de brillos (tambien de la matriz original). Así tenemos unos datos ajustados del mismo orden que los originales.

- **Funciones "datosSalida()":** Escribimos los datos de salida, en distintos ficheros, y de dos maneras distintas, como matrices y como columna (esto último por necesidades que surgieron cuando se escribió el script en Digital Micrograph)

- **Funciones "plotea3d_V2()":** Con esta función ploteamos una matriz como gráfico en tres dimensiones.

C.2. Código

```
##########################################
# script para algoritmo Levenberg-Marquardt
# version V3.4
##########################################

#librería que implementa la función nlsLM()
library(minpack.lm)
```

```
##################################
#FUNCIONES PARA LEER DEFINIR ESTIMADORES INICIALES Y OTROS PARAMETROS USADOS
##################################
estimIniEjemplo1 <- function(){
  #definimos estimadores iniciales para el EJEMPLO 1
  ESTIMADORES_INICIALES <<- data.frame(media_x=numeric(0),
                                       media_y=numeric(0),
                                       sigma_x=numeric(0),
                                       sigma_y=numeric(0),
                                       coefCorr=numeric(0),
                                       peso=numeric(0))
  ESTIMADORES_INICIALES[1,1] <<- 6
  ESTIMADORES_INICIALES[1,2] <<- 14
  ESTIMADORES_INICIALES[1,3] <<- 1
  ESTIMADORES_INICIALES[1,4] <<- 1
  ESTIMADORES_INICIALES[1,5] <<- 0
  ESTIMADORES_INICIALES[1,6] <<- 0.2

  ESTIMADORES_INICIALES[2,1] <<- 17
  ESTIMADORES_INICIALES[2,2] <<- 14
  ESTIMADORES_INICIALES[2,3] <<- 1
  ESTIMADORES_INICIALES[2,4] <<- 1
  ESTIMADORES_INICIALES[2,5] <<- 0
  ESTIMADORES_INICIALES[2,6] <<- 0.2

  ESTIMADORES_INICIALES[3,1] <<- 6
  ESTIMADORES_INICIALES[3,2] <<- 5
  ESTIMADORES_INICIALES[3,3] <<- 1
  ESTIMADORES_INICIALES[3,4] <<- 1
  ESTIMADORES_INICIALES[3,5] <<- 0
  ESTIMADORES_INICIALES[3,6] <<- 0.2

  ESTIMADORES_INICIALES[4,1] <<- 17
  ESTIMADORES_INICIALES[4,2] <<- 5
  ESTIMADORES_INICIALES[4,3] <<- 1
  ESTIMADORES_INICIALES[4,4] <<- 1
  ESTIMADORES_INICIALES[4,5] <<- 0
  ESTIMADORES_INICIALES[4,6] <<- 0.2

  ESTIMADORES_INICIALES[5,1] <<- 12
  ESTIMADORES_INICIALES[5,2] <<- 10
  ESTIMADORES_INICIALES[5,3] <<- 1
  ESTIMADORES_INICIALES[5,4] <<- 1
  ESTIMADORES_INICIALES[5,5] <<- 0
  ESTIMADORES_INICIALES[5,6] <<- 0.1
}
##################################################
estimIniEjemplo2 <- function(){
  #definimos estimadores iniciales para el EJEMPLO 2
  ESTIMADORES_INICIALES <<- data.frame(media_x=numeric(0),
                                       media_y=numeric(0),
                                       sigma_x=numeric(0),
                                       sigma_y=numeric(0),
                                       coefCorr=numeric(0),
                                       peso=numeric(0))
  ESTIMADORES_INICIALES[1,1] <<- 7
  ESTIMADORES_INICIALES[1,2] <<- 14
  ESTIMADORES_INICIALES[1,3] <<- 1
  ESTIMADORES_INICIALES[1,4] <<- 1
  ESTIMADORES_INICIALES[1,5] <<- 0
  ESTIMADORES_INICIALES[1,6] <<- 0.1
```

```
  ESTIMADORES_INICIALES[2,1] <<- 17
  ESTIMADORES_INICIALES[2,2] <<- 14
  ESTIMADORES_INICIALES[2,3] <<- 1
  ESTIMADORES_INICIALES[2,4] <<- 1
  ESTIMADORES_INICIALES[2,5] <<- 0
  ESTIMADORES_INICIALES[2,6] <<- 0.3

  ESTIMADORES_INICIALES[3,1] <<- 24
  ESTIMADORES_INICIALES[3,2] <<- 14
  ESTIMADORES_INICIALES[3,3] <<- 1
  ESTIMADORES_INICIALES[3,4] <<- 1
  ESTIMADORES_INICIALES[3,5] <<- 0
  ESTIMADORES_INICIALES[3,6] <<- 0.3

  ESTIMADORES_INICIALES[4,1] <<- 10
  ESTIMADORES_INICIALES[4,2] <<- 5
  ESTIMADORES_INICIALES[4,3] <<- 1
  ESTIMADORES_INICIALES[4,4] <<- 1
  ESTIMADORES_INICIALES[4,5] <<- 0
  ESTIMADORES_INICIALES[4,6] <<- 0.2

  ESTIMADORES_INICIALES[5,1] <<- 17
  ESTIMADORES_INICIALES[5,2] <<- 5
  ESTIMADORES_INICIALES[5,3] <<- 1
  ESTIMADORES_INICIALES[5,4] <<- 1
  ESTIMADORES_INICIALES[5,5] <<- 0
  ESTIMADORES_INICIALES[5,6] <<- 0.1

  ESTIMADORES_INICIALES[6,1] <<- 28
  ESTIMADORES_INICIALES[6,2] <<- 12
  ESTIMADORES_INICIALES[6,3] <<- 1
  ESTIMADORES_INICIALES[6,4] <<- 1
  ESTIMADORES_INICIALES[6,5] <<- 0
  ESTIMADORES_INICIALES[6,6] <<- 0.1

  ESTIMADORES_INICIALES[7,1] <<- 24
  ESTIMADORES_INICIALES[7,2] <<- 5
  ESTIMADORES_INICIALES[7,3] <<- 1
  ESTIMADORES_INICIALES[7,4] <<- 1
  ESTIMADORES_INICIALES[7,5] <<- 0
  ESTIMADORES_INICIALES[7,6] <<- 0.1
}
################################################
estimIniManual <- function(){
  ESTIMADORES_INICIALES <<- data.frame(media_x=numeric(0),
                                       media_y=numeric(0),
                                       sigma_x=numeric(0),
                                       sigma_y=numeric(0),
                                       coefCorr=numeric(0),
                                       peso=numeric(0))
  #editamos/asignamos los estimadores/parámetros iniciales...
  while(sum(ESTIMADORES_INICIALES$peso)!=1){
    ESTIMADORES_INICIALES <<- edit(ESTIMADORES_INICIALES)
    if(sum(ESTIMADORES_INICIALES$peso)!=1){
      print("la suma de pesos debe de ser 1. Vuelve a introducirlos.")
    }
  }
}
########################################################
datosEntrada <- function(){
  txtDatosIN <- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\datosOUT.txt"
  txtEstimadoresIni <- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\estimadoresOUT.txt"
```

```
  #variable para la matriz de datos
  DATOS_IN <<- read.table(txtDatosIN)
  #variable para los parámetros  que han generado la imagen sintética,
  #para poder compararlos con los que obtengamos como resultado del ajuste
  ESTIMADORES_REALES <<- read.table(txtEstimadoresIni)

  #definimos el offset
  offs <<- min(DATOS_IN)
}


##############################
#FUNCION PARA CONVERTIR LA MATRIZ DE DATOS EN UNA TABLA
################################
convierteMatrizATabla <- function(matrizImagen){
  #escribimos los datos en forma de tabla en vez de como matriz...
  tablaDatos <- data.frame(x=numeric(0),y=numeric(0),z=numeric(0))

  #...los cogemos y organizamos de la matriz ‘‘frecuencias’’...
  for(col in 1:ncol(matrizImagen)){
    for(fil in 1:nrow(matrizImagen)){
      tablaDatos <- rbind(tablaDatos,c(col-0.5,fil-0.5,matrizImagen[nrow(matrizImagen)-fil+1,col]))
    }
  }
  names(tablaDatos) <- c("x","y","z")
  return(tablaDatos)
}


#####################################
#FUNCION PARA RESTAR EL OFFSET Y NORMALIZAR LOS DATOS (TABLA)
########################################
procesaTablaEntrada <- function(tabla){
  #restamos el offset a los datos...
  tabla[,3] <- tabla[,3]-offs

  #Normalizamos
  sumaDatosEntrada <<- sum(tabla[,3])
  tabla[,3] <- tabla[,3]/sumaDatosEntrada

  return(tabla)
}


##############################################
#FUNCION PARA ALGORITMO DE LEVENBERG-MARQUARDT
##############################################
algoritmoLMLibreria <- function(tablaDatos){
  x <- tablaDatos$x
  y <- tablaDatos$y
  z <- tablaDatos$z

  #definimos variable para almacenar los paraametros ajustados
  estimParametros <- ESTIMADORES_INICIALES
  #y el número de componentes de la mixtura
  nComp <- nrow(ESTIMADORES_INICIALES)
  cotaSup <- c()
  cotaInf <- c()

  for(componente in 1:nComp){
    valorMx <- estimParametros[componente,1]
```

```
  valorMy <- estimParametros[componente,2]
  valorSx <- estimParametros[componente,3]
  valorSy <- estimParametros[componente,4]
  valorRho <- estimParametros[componente,5]
  valorPeso <- estimParametros[componente,6]
  mx <- paste("mx",componente, sep="")
  my <- paste("my",componente, sep="")
  sx <- paste("sx",componente, sep="")
  sy <- paste("sy",componente, sep="")
  rho <- paste("rho",componente, sep="")
  peso <- paste("peso",componente, sep="")

  #la funcion expresada como cadena de caracteres
  funcionNormal <-
  paste("(",paste(peso),"*(1/(2*pi*",paste(sx,"*",sy),"*sqrt(1-",paste(rho),"^2)) )*( exp(
   (-1/(2*(1-",paste(rho),"^2)))*(((x-",paste(mx),")^2/",paste(sx),"^2)+
   ((y-",paste(my),")^2/",paste(sy),"^2)-(2*",paste(rho),"*(x-",paste(mx),")*
   (y-",paste(my),")/(",paste(sx,"*",sy),"))) ) ))" )


  ini <- list(mx=valorMx,my=valorMy,sx=valorSx,sy=valorSy,rho=valorRho,peso=valorPeso)
  names(ini)[1]<-paste(mx)
  names(ini)[2]<-paste(my)
  names(ini)[3]<-paste(sx)
  names(ini)[4]<-paste(sy)
  names(ini)[5]<-paste(rho)
  names(ini)[6]<-paste(peso)

  cotaSup <- c(cotaSup, valorMx+7,valorMy+7,valorSx+5,valorSy+5,0.99,5)
  cotaInf <- c(cotaInf,valorMx-7,valorMy-7,valorSx-5,valorSy-5,-0.99,0)

  if(componente==1){
    #generamos la primera componente de la mixtura mas un offset
    formula <- paste("z ~",paste("of"),"+",funcionNormal)

    inicio <- c(of=min(tablaDatos[,3]),ini)
    cotaSup <- c(min(tablaDatos[,3])+(max(tablaDatos[,3])/2), cotaSup)
    cotaInf <- c(min(tablaDatos[,3])-(max(tablaDatos[,3])/2), cotaInf)

  }
  if(componente>1){
    #sumamos las demás componentes al modelo completo
    formula <- paste(formula,"+",funcionNormal)
    inicio <- c(inicio,ini)
  }


}#fin del for

ajuste <- nlsLM(formula,
                data=tablaDatos,
                start=inicio,
                upper=cotaSup,
                lower=cotaInf,
                control=nls.lm.control(maxiter=70,nprint=0),#ptol, ftol, factor...
                trace=T
                )

print(summary(ajuste))

tablaDatosAjustados <- data.frame(tablaDatos$x,tablaDatos$y,predict(ajuste))
estimParametros <- matrix(coef(ajuste)[-1],nrow=nComp,byrow=T)
```

```
  offsAjustado <- coef(ajuste)[1]
  errorOffsAjustado <- summary(ajuste)$coefficients[1,2]
  errorEstimParametros <- matrix(summary(ajuste)$coefficients[-1,2],nrow=nComp,byrow=T)


  valoresAjuste <-
   list(tablaDatosAjustados,estimParametros,errorEstimParametros,offsAjustado,errorOffsAjustado)

  return(valoresAjuste)

}#fin de la funcion ‘‘algoritmoLMLibreria’’


##############################################
#FUNCION PARA PROCESAR LA TABLA DE DATOS DE SALIDA
##############################################
procesaTablaSalida <- function(tablaDatosAjuste){

  #‘‘desnormalizamos’’ los valores ajustados y sumamos el offset que
  # habíamos restado en ‘‘procesaTablaEntrada’’
  tablaDatosAjuste[,3] <- tablaDatosAjuste[,3]*sumaDatosEntrada
  tablaDatosAjuste[,3] <- round(tablaDatosAjuste[,3]+offs)

  #..escribimos la tabla como matriz...
  #...la definimos igual que la de los datos de entrada...
  matrizAjuste <- DATOS_IN

  for(col in 1:ncol(DATOS_IN)){

    for(fil in 1:nrow(DATOS_IN)){
      matrizAjuste[fil,col] <- tablaDatosAjuste[which(
       (tablaDatosAjuste[,1]==(col-0.5))&(tablaDatosAjuste[,2]==(nrow(DATOS_IN)-fil+0.5)) ),3]
    }

  }
  return(matrizAjuste)
}


##############################################
#funcion para escribir datos
##############################
datosSalida <- function(estimAjustados,datosAjustados,errorEstimAjustados){
  txtDatosOUT <- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\datosImgDM_IN.txt"
  txtEstimadoresOUT <- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\estimImgDM_IN.txt"
  txtErrorEstimadoresOUT <- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\errorEstimImgDM_IN.txt"

  write.table(estimAjustados,txtEstimadoresOUT, row.names=F, col.names=F)
  write.table(datosAjustados,txtDatosOUT, row.names=F, col.names=F)
  write.table(errorEstimAjustados,txtErrorEstimadoresOUT, row.names=F, col.names=F)

  #esribimos datos y estimadores en una columna, habiendolos tomado de la matriz por filas
  txtDatosOUTColumna <- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\ColumnaDatosImgDM_IN.txt"
  txtEstimadoresOUTColumna <- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\ColumnaEstimImgDM_IN.txt"
  txtErrorEstimadoresOUTColumna <- "C:\\Users\\Usuario\\Desktop\\txtsPruebasR\\ColumnaErrorEstimImgDM_IN.txt"
  vectorDatos <- numeric(0)
  vectorEstim <- numeric(0)
  vectorErrorEstim <- numeric(0)

  for(fil in 1:nrow(datosAjustados)){
    for(col in 1:ncol(datosAjustados)){
      vectorDatos <- c(vectorDatos,datosAjustados[fil,col])
```

```
    }
  }
  write.table(as.matrix(vectorDatos),txtDatosOUTColumna, row.names=F, col.names=F)

  for(fil in 1:nrow(estimAjustados)){
    for(col in 1:ncol(estimAjustados)){
      vectorEstim <- c(vectorEstim,estimAjustados[fil,col])
      vectorErrorEstim <- c(vectorErrorEstim,errorEstimAjustados[fil,col])
    }
  }
  write.table(as.matrix(vectorEstim),txtEstimadoresOUTColumna, row.names=F, col.names=F)
  write.table(as.matrix(vectorErrorEstim),txtErrorEstimadoresOUTColumna, row.names=F, col.names=F)
}


##########################################
#funcion para ploteaar en 3d una matriz
#####################################33
plotea3d_V2 <- function(frecuencias){
  frecuencias <- as.matrix(frecuencias)

  pixeles_x <- seq(1,ncol(frecuencias))
  pixeles_y <- seq(1,nrow(frecuencias))

  eventos <- matrix(nrow=nrow(frecuencias),ncol=ncol(frecuencias))
  eventos <- frecuencias[pixeles_y,pixeles_x]

  persp(pixeles_y,pixeles_x,eventos, zlab="brillo",zlim=c(0,max(frecuencias)), phi=30, theta=60)

}




#####################################################################
####  MAIN  #########################################################
#####################################################################

#RECIBO DATOS DE ENTRADA: estimadores iniciales, matriz de datos y tamanio de la imagen
datosEntrada()
estimIniEjemplo1()
#estimIniEjemplo2()
#estimIniManual()

#PONEMOS LA MATRIZ DE ENTRADA COMO TABLA
tablaDatosIn <- convierteMatrizATabla(DATOS_IN)

#PROCESAMOS LA TABLA
tablaDatosInProc <- procesaTablaEntrada(tablaDatosIn)

#EJECUTAMOS EL ALGORITMO pasando la tabla de datos y el gradiente ``grad'' (de UNA normal)
#iniciamos el cronómetro para medir el tiempo
t <- proc.time()
nuevosDatos <- algoritmoLMLibreria(tablaDatosInProc)
proc.time()-t

tablaDatosAjustados <- as.data.frame(nuevosDatos[1])
estimadoresAjustados <- as.data.frame(nuevosDatos[2])
errorEstimadoresAjustados <- as.data.frame(nuevosDatos[3])
offsetAjustado <- as.numeric(nuevosDatos[4])
errorOffsetAjustado <- as.numeric(nuevosDatos[5])
```

```
#PROCESAMOS LA TABLA DE DATOS AJUSTADOS Y LA ESCRIBIMOS COMO MATRIZ
DATOS_OUT <- procesaTablaSalida(tablaDatosAjustados)
#plotea3d_V2(DATOS_OUT)

#ESCRIBIMOS LOS DATOS AJUSTADOS...
datosSalida(estimadoresAjustados,DATOS_OUT,errorEstimadoresAjustados)
```

Apéndice D

Código para el Plug-In

Compuesto por cuatro ficheros, uno de ellos en R y tres en DM-scripting (clase "Mancha" y "Accion" instalados como librerías y "main" como plug-in). Remito al apéndice A, donde se detalla el proceso de instalación.

D.1. Descripción de los DM-script

- **"main":** Aquí se encuentra el núcleo y la mayor parte del código. Contiene las siguientes funciones y clases,
  - **Función "seleccionaROI()":** Función que captura la región de interés a analizar.
  - **Función "visualizaROI()":** Con esta, se amplía y muestra la región seleccionada en "seleccionaROI()".
  - **Función "escribeImg()":** Escribe en un fichero txt la matriz de datos correspondiente a la región seleccionada.
  - **Función "lanzaR()":** Aquí realizamos la llamada al R-script.
  - **Función "leeImg()":** Leemos la matriz de datos asociados (de un txt que genera el R-script) a la imagen ajustada.
  - **Función "generaManchas()":** Se leen los estimadores ajustados (de otro txt que genera el R-script) y se asignan a objetos de la clase "Mancha". Estos objetos, a su vez, se cargan en una lista.
  - **Función "borrarFicheros()":** Elimina algunos ficheros usados en la ejecución de la carpeta "plugInDM".
  - **Clase "Selección":** Esta clase es un listener que implementa las rutinas para la captura de parámetros iniciales por selección

de picos. Hay dos métodos, uno para la selección que llamamos "manual" y otro para la denominada como "automática".

- **Clase "AccionTeclado":** Clase que implementa el key-listener para finalizar la selección de picos al pulsar la tecla "escape". Si se pulsa otra tecla, lanza un aviso de que se debe de pulsar "escape" al finalizar.
- **Función "main()":** Como es habitual en programación, aquí tenemmos la secuencia de llamadas a las distintas funciones para realizar el proceso completo.

- **Clase "Mancha":** Clase que genera objetos en los que se asocian los picos de la imagen a los parámetros ajustados. Contiene, en el siguiente orden, las variables de clase, los habituales getters y setters (según la terminología usada por los desarrolladores Java) y el constructor. A continuación, dos funciones ("asignaROI()" y "asignaCentro()") que generan las regiones de interés romboidales y puntuales que se sitúan sobre cada pico en la imagen final tras una ejecución. Y por último, "creaEtiqueta()" es el método encargado de desplegar y recoger la etiqueta sobre cada rombo, mostrando la ID del pico o los parámetros de ajuste.

- **Clase "Accion":** Esta clase corresponde al listener que se activa sobre la imagen generada tras el ajuste. Implementa las funciones mediante las cuales, al añadir una región de interés puntual o lineal sobre los rombos, se obtienen parámetros de ajuste o distancias, respectivamente.

D.1.1. Código DM: "main"

```
/*
SCRIPT TFM - main
*/


/* ************************************
--- VARIABLES GLOBALES
********************************** */
//variables para la imagen original
image img
imageDisplay imgDisp

//variable para la ROI seleccionada
image imgROI
imageDisplay imgROIDisp

//variable para la ‘‘imgROI’’ ajustada
image imgROIAjustada
```

```
//...definimos las rutas para los archivos de texto que usaremos...
string archivoDatosOUT = "C:/Users/Usuario/Desktop/plugInDM/datosImgDM_OUT.txt"
string archivoEstimOUT = "C:/Users/Usuario/Desktop/plugInDM/estimImgDM_OUT.txt"
string archivoDatosIN = "C:/Users/Usuario/Desktop/plugInDM/datosImgDM_IN.txt"
string archivoEstimIN = "C:/Users/Usuario/Desktop/plugInDM/estimImgDM_IN.txt"
string archivoDatosINColumna = "C:/Users/Usuario/Desktop/plugInDM/ColumnaDatosImgDM_IN.txt"
string archivoEstimINColumna = "C:/Users/Usuario/Desktop/plugInDM/ColumnaEstimImgDM_IN.txt"
string archivoErrorEstimIN = "C:/Users/Usuario/Desktop/plugInDM/errorEstimImgDM_IN.txt"
string archivoErrorEstimINColumna = "C:/Users/Usuario/Desktop/plugInDM/ColumnaErrorEstimImgDM_IN.txt"
string archivoResultados = "C:/Users/Usuario/Desktop/plugInDM/estimadores_IN.txt"

// definimos el numero de normales como variable global
number nNormales = 0
// y el origen de la ROI que ajustaremos
number origenROI_x, origenROI_y
number marcoROISup, marcoROIInf, marcoROIIzda, marcoROIDcha
//creamos una lista para las manchas
object listaManchas = alloc(ObjectList)
//...y una variable para las selecciones para estimadores iniciales
string cadenaEstimadoresIni = ""

//ids para los listeners que usaremos
number idListenerEstimaciones
number idListenerResultados
/* ****************************************** */



/* ************************************
--- FUNCION PARA SELECCIONAR ROI DE INTERES
************************************* */
void seleccionaROI(){
//asignamos la ROI a la variable (tipo image) global.
//De ella obtendremos los datos de cada pixel para el procesado
imgROI = img[]

//variables para definir la ROI seleccionada y su limite/marco
ROI ROISelec = newROI()

if(imgDisp.imageDisplayCountROIs()==0 || imgDisp.imageDisplayCountROIs()>1){
if(imgDisp.imageDisplayCountROIs()==0){
number nPix_x, nPix_y
getSize(img,nPix_x,nPix_y)
ROISelec.ROISetRectangle(0,0,nPix_y-1,nPix_x-1)
}
if(imgDisp.imageDisplayCountROIs()>1){
okDialog("selecciona UNA sola ROI para el ajuste")
exit(0)
}
}
else{
ROISelec = imageDisplayGetROI(imgDisp,0)
}

ROISelec.ROIGetRectangle(marcoROISup,marcoROIIzda,marcoROIInf,marcoROIDcha)
ROISelec.ROIGetVertex(0,origenROI_x,origenROI_y)

//presentamos la ROI seleccionda como nueva imagen... */
showImage(imgROI)
imgROIDisp = imgROI.imagegetimagedisplay(0)
setname(imgROI, "ORIGEN: ("+"col "+origenROI_x+", fil "+origenROI_y+") "+"; MARCO: "
     +"sup "+marcoROISup+", inf "+marcoROIInf+", izda "+marcoROIIzda+", dcha "+marcoROIDcha)
```

```
}//cierre de ‘‘seleccionaROI’’
/* ************************************** */


/* ************************************************
--- FUNCION PARA VISUALIZAR IMAGEN AMPLIADA
********************************************** */
void visualizaROI(image imgLocal){
//tamanio de la zona de visualizacion
number pantallaAncho, pantallaAlto
//tamanio de la ventana
number ventanax, ventanay
//posicion de la ventana
number ventanaPosx=10, ventanaPosy=25

GetScreenSize(pantallaAncho, pantallaAlto )

showImage(imgLocal)

setWindowPosition(imgLocal,ventanaPosx, ventanaPosy)
setWindowSize(imgLocal,pantallaAncho-(2*ventanaPosx), pantallaAlto-(2*ventanaPosy))
}


/* ************************************
--- FUNCION PARA ESCRIBIR DATOS DE LA IMAGEN EN FICHERO
************************************** */
void escribeImg(image imgLocal){
number refArchivoEscritura = CreateFileForWriting( archivoDatosOUT )
number nPixX,nPixY
number nCol,nFil
getSize(imgLocal,nPixX,nPixY)
for(nFil=0;nFil<nPixY;nFil++){
for(nCol=0;nCol<nPixX;nCol++){
writeFile( refArchivoEscritura, ""+getPixel(imgLocal,nCol,nFil)+" " )
if(nCol==(nPixX-1)){
writeFile( refArchivoEscritura, "\n")
}
}
}

closeFile( refArchivoEscritura )
}//cierre de la funcion ‘‘escribeImg’’
/* ************************************** */



/* ****************************************
--- FUNCION PARA LANZAR EL SCRIPT EN R
************************************** */
void lanzaR(){
if(twoButtonDialog("Confirmar lanzamiento de algoritmo de ajuste.","continuar","cancelar")){
string llamada="cmd /c Rscript C:/Users/Usuario/Desktop/plugInDM/algoritmoLM_V5_DM_Definitivo.R"
launchExternalProcess(llamada)
}
else{
if(doesFileExist(archivoDatosOUT)==1){
deleteFile( archivoDatosOUT )
}
if(doesFileExist(archivoEstimOUT)==1){
deleteFile( archivoEstimOUT )
```

```
}
if(doesFileExist(archivoDatosIN)==1){
deleteFile( archivoDatosIN )
}
if(doesFileExist(archivoResultados)==1){
deleteFile( archivoResultados )
}
exit(0)
}
}//cierre de ‘‘lanzaR()’’
/* ************************************* */



/* ****************************************
--- FUNCION PARA LEER DATOS DE IMAGEN GENERADOS (CON R)
**************************************** */
void leeImg(){
//definimos dimensiones de la imagen ajustada: iguales que las de la ROI seleccionada
number sizex, sizey
getSize(imgROI,sizex,sizey)
image imgAjuste := RealImage("ORIGEN: ("+"col "+origenROI_x+", fil "+origenROI_y+") "+";
      MARCO: "+"sup "+marcoROISup+", inf "+marcoROIInf+", izda "+marcoROIIzda+", dcha
      "+marcoROIDcha,4,sizex,sizey)

//definimos la referencia del archivo del que leemos
number refArchivoLectura = OpenFileForReading( archivoDatosINColumna )
//variable donde cargaremos las lienas del txt que leemos
string linea = " "
number contadorFil=0, contadorCol=0, contador=1, valor

while(contador<=(sizex*sizey)){
readFileLine(refArchivoLectura,linea)
valor = linea.val()

setPixel(imgAjuste,contadorCol,contadorFil,valor)

if(contador%sizex==0){
contadorCol=0
contadorFil++//controla el numero y cambio de fila
}
else{
contadorCol++//controla el cambio de columna
}
contador++//controla el recorrido del txt
}

closeFile(refArchivoLectura)

imgROIAjustada := imgAjuste
} // fin de la funcion ‘‘leeFichero’’
/* ********************************************** */


/* ************************************************
--- FUNCION PARA GENERAR OBJETOS CLASE ‘‘MANCHA’’
************************************************ */
void generaManchas(imageDisplay displayLocal){

//img.selectImage()
//ChooseMenuItem( "File", "Import Data...", )
number refArchivoLectura = OpenFileForReading( archivoEstimINColumna )
```

```
number refArchivoLecturaErrores = OpenFileForReading( archivoErrorEstimINColumna )
string linea=""
string lineaError=""

string resultados = "ID     mediax     mediay     sigmax     sigmay     coefCorr     peso \n"

number cuentaLineas=1
number cuentaParam=1
number n=nNormales
number mx, my, sx, sy, rho, peso
number emx, emy, esx, esy, erho, epeso

//iniciamos bucle para recoger los parametros ajustados y asignarlos a objetos de la clase ‘‘Mancha’’
while(cuentaLineas<=(n*6)){
readFileLine(refArchivoLectura,linea)
readFileLine(refArchivoLecturaErrores,lineaError)

if(cuentaLineas%6==0){
cuentaParam=1
peso=linea.val()
epeso=lineaError.val()
//cada 6 lineas leidas creamos un objeto...
object mancha1 = alloc(Mancha).init(mx,my,sx,sy,rho,peso,emx,emy,esx,esy,erho,
         epeso,origenROI_x, origenROI_y)
resultados.stringAppend(scriptObjectGetID(mancha1)+"     "+mx+"     "+my+"     "+sx+"
         "+sy+"     "+rho+"     "+peso+"\n")
//...lo asignamos a la lista
listaManchas.AddObjectToList(mancha1)
//...y seteamos y graficamos su ROI asociada
mancha1.asignaROI(displayLocal)
mancha1.asignaROICentro(displayLocal)
}
else{
if(cuentaParam==1) mx=linea.val()
if(cuentaParam==1) emx=lineaError.val()
if(cuentaParam==2) my=linea.val()
if(cuentaParam==2) emy=lineaError.val()
if(cuentaParam==3) sx=linea.val()
if(cuentaParam==3) esx=lineaError.val()
if(cuentaParam==4) sy=linea.val()
if(cuentaParam==4) esy=lineaError.val()
if(cuentaParam==5) rho=linea.val()
if(cuentaParam==5) erho=lineaError.val()

cuentaParam++
}

cuentaLineas++

}

closeFile(refArchivoLectura)
closeFile(refArchivoLecturaErrores)

number refArchivoResultados = CreateFileForWriting( archivoResultados )
writeFile( refArchivoResultados, resultados)
closeFile( refArchivoResultados )
}


/* ***********************************************
--- FUNCION para borrar los txts usados durante la ejecucion
```

```
****************************************************** */
void borrarFicheros(){
deleteFile( archivoEstimIN )
deleteFile( archivoDatosINColumna )
deleteFile( archivoEstimINColumna )
deleteFile( archivoErrorEstimIN )
deleteFile( archivoErrorEstimINColumna )
}



/* ********************************
--- CLASE para listener para la seleccion de manchas y obtencion de parametros iniciales
********************************** ***/
class Seleccion : object{

void estimadores(Object self, Number e_fl, ImageDisplay disp, Number r_fl, Number r_fl2, ROI roiEstim){

image imgSeleccion = getFrontImage()[]

if(roiEstim.ROIIsRectangle()==1){
number t,b,r,l
number ox, oy
string linea = ""
number mediax, mediay
number peso=1
number sizex, sizey
getSize(imgROI,sizex,sizey)

number sumaOffset = sizex*sizey*min(imgROI)
number aproxOffset = min(imgROI)
number maxROI = max(imgROI)

roiEstim.ROISetVolatile(0)

roiEstim.ROIGetRectangle(t,l,b,r)
roiEstim.ROIGetVertex(0,ox,oy)

max(imgSeleccion, mediax, mediay)

peso = (max(imgSeleccion)-aproxOffset)/(maxROI-aproxOffset)

linea = ""+(ox+mediax+1)+" "+(oy+mediay+1)+" "+((r-l)/4)+" "+((b-t)/4)+" "+0+" "+peso+"\n"

cadenaEstimadoresIni.stringAppend(linea)
nNormales++

result("\n ----- \n"+cadenaEstimadoresIni+"numero de picos seleccionados: "+nNormales+"\n ------ \n")
}
else{

okDialog("la ROI debe ser Rectangular")
disp.imageDisplayDeleteROI(roiEstim)
}


}//fin ‘‘estimadores()’’

void estimadoresAutom(Object self, Number e_fl, ImageDisplay disp, Number r_fl, Number r_fl2, ROI roiEstim){

image imgSeleccion = getFrontImage()[]
```

```
if(roiEstim.ROIIsRectangle()==1){
number t,b,r,l
number ox, oy
string linea = ""
number mediax, mediay, mediax0, mediay0
number periodox, periodoy
number peso=1

number sumaOffset = (marcoROIInf-marcoROISup)*(marcoROIDcha-marcoROIIzda)*min(imgROI)
number aproxOffset = min(imgROI)
number maxROI = max(imgROI)

getNumber("periodo en x: ", 23, periodox)
getNumber("periodo en y: ", 23, periodoy)

roiEstim.ROISetVolatile(0)

roiEstim.ROIGetRectangle(t,l,b,r)
roiEstim.ROIGetVertex(0,ox,oy)

max(imgSeleccion, mediax0, mediay0)

peso = (max(imgSeleccion)-aproxOffset)/(maxROI-aproxOffset)

mediax = mediax0
mediay = mediay0
while((oy+mediay+1)<(marcoROIInf-marcoROISup)){
while((ox+mediax+1)<(marcoROIDcha-marcoROIIzda)){
linea = ""+(ox+mediax+1)+" "+(oy+mediay+1)+" "+((r-l)/4)+" "+((b-t)/4)+" "+0+" "+peso+"\n"

cadenaEstimadoresIni.stringAppend(linea)
nNormales++

mediax = mediax+periodox

}
mediax = mediax0
mediay = mediay+periodoy
}
result("\n ----- \n"+cadenaEstimadoresIni+"numero de picos seleccionados: "+nNormales+"\n ------ \n")
}
else{

okDialog("la ROI debe ser Rectangular")
disp.imageDisplayDeleteROI(roiEstim)
}

}//fin de ‘‘estimadoresAuto()’’


Seleccion(object self){
result("\n evento seleccion lanzado "+self.ScriptObjectGetID())
}
~Seleccion(object self){
result("\n evento seleccion destruido "+self.ScriptObjectGetID())
}

}//fin de la clase ‘‘seleccion’’
```

```
/* ************************************************
--- CLASE para el key-listener que finaliza la seleccion de picos
********************************************** */
Class AccionTeclado : object
{

Number SelfTOKEN

number escalaCalibracion, origenCalibracion
string unidadesCalibracion

void KeepToken(object self, number tok) SelfTOKEN = tok

void setCalibracion(object self, number escCal, number origCal, string udsCal){
escalaCalibracion = escCal
origenCalibracion = origCal
unidadesCalibracion = udsCal
}

number controlTecla(Object self, ImageDisplay disp, Object keydesc)
{
number b_keyhandled = 0
Result("\n Key pressed:"+keydesc.GetKeyDescriptor())
Result(" ("+keydesc.GetDescription()+")")

If ( keydesc.MatchesKeyDescriptor("esc"))
{
//...destruimos keylistener...
disp.ImageDisplayRemoveKeyHandler(SelfTOKEN)
//...destruimos listener para la seleccion de picos...
disp.imageDisplayRemoveEventListener(idListenerEstimaciones)

b_keyhandled = 1
//...eliminamos rois en el display...
while ( 0 < disp.ImageDisplayCountROIs() ){
ROI r = disp.ImageDisplayGetROI( 0 )
                disp.ImageDisplayDeleteROI( r )
}


//...escribimos en fichero los estimadores...
number ref = createFileForWriting(archivoEstimOUT)
writeFile(ref, cadenaEstimadoresIni )
closeFile(ref)

//////lanzamos script R...
lanzaR()
//////leemos los datos ajustados...
leeImg()
//////...y mostramos la imagen refinada...
showImage(imgROIAjustada)
//////... y generamos los objetos "mancha" sobre ella...
generaManchas(imgROIDisp)

object acc = alloc(Accion)
acc.setListaManchas(listaManchas)
acc.setCalibracion(escalaCalibracion, origenCalibracion, unidadesCalibracion)

number idListenerResultados = imgROIDisp.imageDisplayAddEventListener(acc,"roi_added,roi_end_track:muestraInfo")
borrarFicheros()
}
else{
```

```
okDialog("Para finalizar la seleccion pulsa escape")
}
Return b_keyhandled
}

AccionTeclado(object self){
result("\n evento KEYLISTENER lanzado "+self.ScriptObjectGetID())
okDialog("Cuando acabes las selecciones pulsa ESC")
}
~AccionTeclado(object self){
result("\n evento KEYLISTENER destruido "+self.ScriptObjectGetID())
}

}



////////////////////////////////////////////////
////////////////////////////////////////////////
///////////////// MAIN /////////////////////////
////////////////////////////////////////////////
void main(){
//////...borramos archivos de la anterior ejecucion...
if(doesFileExist(archivoDatosOUT)==1){
deleteFile( archivoDatosOUT )
}
if(doesFileExist(archivoEstimOUT)==1){
deleteFile( archivoEstimOUT )
}
if(doesFileExist(archivoDatosIN)==1){
deleteFile( archivoDatosIN )
}
if(doesFileExist(archivoResultados)==1){
deleteFile( archivoResultados )
}


//////asignamos a ‘‘img’’ la imagen en pantalla
img.getFrontImage()
imgDisp = img.imageGetImageDisplay(0)
//////...y tomamos la calibracion de la imagen original...
number escalaCalibracion, origenCalibracion
string unidadesCalibracion
img.imageGetDimensionCalibration(0, origenCalibracion, escalaCalibracion,unidadesCalibracion,1 )

//////asignamos a ‘‘imgROI’’ la ROI seleccionada...
//////...y la seteamos en el imageDisplay ‘‘imgROIDisp’’
seleccionaROI()
visualizaROI(imgROI)
//////escribimos en un txt los datos de la ROI
escribeImg( imgROI )


//////introducimos parametros iniciales...
//...objeto listener para seleccionar los picos...
object estimaciones = alloc(Seleccion)

if(twoButtonDialog("Introduce los parametros iniciales
                    (seleccion de picos) de forma:","manual","automatica")){
idListenerEstimaciones =
 imgROIDisp.imageDisplayAddEventListener(estimaciones,"roi_end_track:estimadores")
}
```

```
else{
okDialog("selecciona el primer elemento de cada conjunto
            (el mas proximo a la esquina superior izquierda)")
idListenerEstimaciones =
 imgROIDisp.imageDisplayAddEventListener(estimaciones,"roi_end_track:estimadoresAutom")
}

//...objeto key listener pra finalizar la seleccion de picos...
object controlFinal = Alloc(AccionTeclado)
controlFinal.setCalibracion(escalaCalibracion, origenCalibracion, unidadesCalibracion)
number idControlFinal = imgROIDisp.ImageDisplayAddKeyHandler( controlFinal, "controlTecla" )
controlFinal.setCalibracion(escalaCalibracion, origenCalibracion, unidadesCalibracion)
controlFinal.KeepToken(idControlFinal)

}//...fin del main()...

main()
```

D.1.2. Código DM: clase “Mancha”

```
/* ********************************************************
--- CLASE para objetos ‘‘Mancha’’: cada una de las normales
******************************************************** */
class Mancha : object{

//...variables de clase...
//...los parmetros de la normal...
number mu_x, mu_y, sigma_x, sigma_y, rho, peso/*, IDLista*/
//...y sus desviaciones...
number emu_x, emu_y, esigma_x, esigma_y, erho, epeso
number IDMancha, IDROIMancha
ROI ROIMancha
ROI ROICentro
number origenROI_x, origenROI_y

number cuentaClick


//...getters y setters...
void setCuentaClickIni(object self) cuentaClick=2
number getMux(object self) return mu_x
number getMuy(object self) return mu_y
ROI getROIMancha(object self) return ROIMancha

void setCuentaClick(object self) cuentaClick++


//...constructor
object init( object self, number mx, number my, number sx, number sy, number ro, number
 pes,number emx, number emy, number esx, number esy, number ero, number epes, number oROI_x,
  number oROI_y ){

mu_x = mx
mu_y = my
sigma_x = sx
sigma_y = sy
rho = ro
peso = pes
emu_x = emx
```

```
emu_y = emy
esigma_x = esx
esigma_y = esy
erho = ero
epeso = epes

origenROI_x = oROI_x
        origenROI_y = oROI_y

IDMancha = ScriptObjectGetID(self)

//...inicializamos variable ‘‘cuentaClick’’ al crear el objeto
setCuentaClickIni(self)

return self
}

void asignaROI(object self, imageDisplay displayLocal){
ROIMancha = newROI()
ROIMancha.ROIAddVertex(mu_x-sigma_x , mu_y)
ROIMancha.ROIAddVertex(mu_x , mu_y+sigma_y)
ROIMancha.ROIAddVertex(mu_x+sigma_x , mu_y)
ROIMancha.ROIAddVertex(mu_x , mu_y-sigma_y)
ROIMancha.ROISetColor(0,0,1)
ROISetIsClosed(ROIMancha,1)
ROISetVolatile(ROIMancha,0)

ROIMancha.ROISetLabel("ID: "+IDMancha)
displayLocal.imageDisplayAddROI(ROIMancha)
}

void asignaROICentro(object self, imageDisplay displayLocal){
ROICentro = newROI()
ROICentro.ROISetPoint(mu_x,mu_y)
ROISetVolatile(ROICentro,0)

displayLocal.imageDisplayAddROI(ROICentro)
}

void creaEtiqueta(object self){

number selec = cuentaClick%2

if(selec==1){
number mu_xG = mu_x+origenROI_x
number mu_yG = mu_y+origenROI_y
ROIMancha.ROISetLabel("ID: "+IDMancha+"\n media_x: "+mu_x+" +- "+emu_x+"\n media_y:
 "+mu_y+" +- "+emu_y+"\n media_x (global): "+(mu_xG)+"\n media_y (global):
  "+mu_yG+"\n peso: "+peso+" +- "+epeso)

result("\n ------ \n")
result("\n ID: "+IDMancha)
result("\n media x: " + mu_x + " +- "+emu_x)
result("\n media y: " + mu_y + " +- " +emu_y)
result("\n media x (global): " + (mu_x+origenROI_x))
result("\n media y (global): " + (mu_y+origenROI_y))
result("\n peso: " + peso + " +- "+epeso)
result("\n sigmax: "+sigma_x+" +- "+esigma_x)
result("\n sigmay: "+sigma_y+" +- "+esigma_y)
result("\n coef. Corr.: "+rho+" +- "+erho)
result("\n ------ \n")
```

```
}
if(selec==0){
ROIMancha.ROISetLabel("ID: "+IDMancha)
}

}

}
```

D.1.3. Código DM: clase "Accion"

```
/* ************************************************************************** */
/* ******* CLASE ACCION implementa el listener para distancias y parametros **** */
/* ************************************************************************** */
class Accion : object{

object listaManchas

number escalaCalibracion, origenCalibracion
string unidadesCalibracion

void setListaManchas(object self, object lista) listaManchas = lista

void setCalibracion(object self, number escCal, number origCal, string udsCal){
escalaCalibracion = escCal
origenCalibracion = origCal
unidadesCalibracion = udsCal
}

void muestraInfo(Object self, Number e_fl, ImageDisplay disp, Number r_fl, Number r_fl2, ROI r){

//condicional para ejecutar la accion de mostrar parametros con una ROI PUNTUAL
if(r.ROIIsPoint()==1){

number x,y
number contador=0
number idMancha
r.ROIGetPoint(x,y)

//bucle para recorrer ‘‘listaManchas’’ y buscar la que corresponde al punto (x,y)
for(contador=0 ; contador<listaManchas.sizeOfList() ; contador++){

//asignamos la id de cada mancha en ‘‘listaManchas’’ a la variable idMancha
idMancha = ScriptObjectGetID( listaManchas.ObjectAt( contador ) )

//y accedemos a la ROI asociada al objeto de clase Mancha que contenga al punto
if(getScriptObjectFromID(idMancha).getROIMancha().ROIContainsPoint(x,y)==1){

getScriptObjectFromID(idMancha).setCuentaClick()
getScriptObjectFromID(idMancha).creaEtiqueta()

break
}

}

}//fin del ‘‘if’’ para ROI-punto


//condicional para ejecutar la accion de medir distancia con una ROI LINEA
```

```
if(r.ROIIsLine()==1){

number idMancha1, idMancha2
number x1, y1, x2, y2, distancia
number contador = 0, continuaContador = 0

r.ROIGetLine(x1,y1,x2,y2)

if(listaManchas.sizeOfList()>1){

//bucle para recorrer ‘‘listaManchas’’ y buscar la que corresponde al punto (x1,y1) y (x2,y2)
for(contador=0; contador<listaManchas.sizeOfList(); contador++){

idMancha1 = ScriptObjectGetID( listaManchas.ObjectAt( contador ) )

    //y accedemos a la ROI asociada al objeto de clase Mancha que contenga al punto
if(getScriptObjectFromID(idMancha1).getROIMancha().ROIContainsPoint(x1,y1)==1){
break
}

}
for(contador=0; contador<listaManchas.sizeOfList(); contador++){

idMancha2 = ScriptObjectGetID( listaManchas.ObjectAt( contador ) )

    //y accedemos a la ROI asociada al objeto de clase Mancha que contenga al punto
if(getScriptObjectFromID(idMancha2).getROIMancha().ROIContainsPoint(x2,y2)==1){
break
}

}

if(idMancha1==idMancha2 ||
 getScriptObjectFromID(idMancha2).getROIMancha().ROIContainsPoint(x2,y2)==0 ||
  getScriptObjectFromID(idMancha2).getROIMancha().ROIContainsPoint(x2,y2)==0){
r.ROISetLabel("error")
}
else{
object m1 = getScriptObjectFromID(idMancha1)
object m2 = getScriptObjectFromID(idMancha2)
distancia = ((m1.getMux()-m2.getMux())**2+(m1.getMuy()-m2.getMuy())**2)**(1/2)
r.ROISetLabel("distancia: "+distancia*escalaCalibracion+"  "+unidadesCalibracion)
result("\n ------- \n")
result("\n extremos: ("+x1+","+y1+") y ("+x2+","+y2+")")
result("\n distancia (pixeles): "+distancia+" - "+distancia*escalaCalibracion+"  "+unidadesCalibracion)
result("\n ------- \n")
}

}


else{
okDialog("Solo hay un elemento. No podemos medir distancias.")
}
}//fin del ‘‘if’’ para medida de distancia (ROI-linea)

}//fin de metodo ‘‘muestraInfo’’


Accion(object self){
result("\n evento INFORMACION lanzado "+self.ScriptObjectGetID())
}
```

```
~Accion(object self){
result("\n evento INFORMACION destruido "+self.ScriptObjectGetID())
}

}//cierre de la clase ‘‘Accion’’
```

D.2. Descripción del R-script

Remitimos a la explicación del código del apéndice C puesto que implementa las mismas funciones. Hay pequeños cambios respecto de este otro para adaptarlo al funcionamiento junto con DM. Por ejemplo, aquí los parámetros iniciales se leen de un fichero de texto, se añade alguna salida de datos por pantalla tras la ejecución del algoritmo, y se tiene en cuenta que ahora, para imágenes digitales, el origen se tiene en la esquina superior izquierda, mientras que el código presentado en C se desarrolló considerando el origen en la esquina inferior izquierda.

D.2.1. Código R

```
##########################################
# script para algoritmo Levenberg-Marquardt
# version V5 para DM
##########################################

library(minpack.lm)

###################################
#FUNCION PARA LEER DATOS DE UN TXT
###################################
datosEntrada <- function(){
  txtDatosIN <- "C:\\Users\\Usuario\\Desktop\\plugInDM\\datosImgDM_OUT.txt"
  txtEstimadoresIni <- "C:\\Users\\Usuario\\Desktop\\plugInDM\\estimImgDM_OUT.txt"

  DATOS_IN <<- read.table(txtDatosIN)
  ESTIMADORES_INICIALES <<- read.table(txtEstimadoresIni)

  #definimos el offset
  offs <<- min(DATOS_IN)

}


#############################
#FUNCION PARA CONVERTIR LA MATRIZ DE DATOS EN UNA TABLA
###################################
convierteMatrizATabla <- function(matrizImagen){
  #escribimos los datos en forma de tabla en vez de como matriz...
  tablaDatos <- data.frame(x=numeric(0),y=numeric(0),z=numeric(0))
  #...los cogemos y organizamos de la matriz ‘‘frecuencias’’...
  for(col in 1:ncol(matrizImagen)){
    for(fil in 1:nrow(matrizImagen)){
```

```
      tablaDatos <- rbind(tablaDatos,c(col-0.5,fil-0.5,matrizImagen[fil,col]))
    }
  }
  names(tablaDatos) <- c("x","y","z")
  return(tablaDatos)
}


#####################################
#FUNCION PARA RESTAR EL OFFSET Y NORMALIZAR LOS DATOS (TABLA)
#########################################
procesaTablaEntrada <- function(tabla){
  #restamos offset a los datos...
  tabla[,3] <- tabla[,3]-offs

  #Normalizamos...
  sumaDatosEntrada <<- sum(tabla[,3])
  tabla[,3] <- tabla[,3]/sumaDatosEntrada

  return(tabla)
}


###############################################
#FUNCION PARA ALGORITMO DE LEVENBERG-MARQUARDT
###############################################
algoritmoLMLibreria <- function(tablaDatos){

  x <- tablaDatos$x
  y <- tablaDatos$y
  z <- tablaDatos$z

  ################
  #generamos la formula completa
  estimParametros <- ESTIMADORES_INICIALES
  #y el número de componentes de la mixtura
  nComp <- nrow(ESTIMADORES_INICIALES)
  cotaSup <- c()
  cotaInf <- c()

  for(componente in 1:nComp){

    valorMx <- estimParametros[componente,1]
    valorMy <- estimParametros[componente,2]
    valorSx <- estimParametros[componente,3]
    valorSy <- estimParametros[componente,4]
    valorRho <- estimParametros[componente,5]
    valorPeso <- estimParametros[componente,6]
    mx <- paste("mx",componente, sep="")
    my <- paste("my",componente, sep="")
    sx <- paste("sx",componente, sep="")
    sy <- paste("sy",componente, sep="")
    rho <- paste("rho",componente, sep="")
    peso <- paste("peso",componente, sep="")

    #la funcion expresada como cadena de caracteres
    funcionNormal <- paste("(",paste(peso),"*(
     1/(2*pi*",paste(sx,"*",sy),"*sqrt(1-",paste(rho),"^2)) )*( exp(
      (-1/(2*(1-",paste(rho),"^2)))*(((x-",paste(mx),")^2/",paste(sx),"^2)
      +((y-",paste(my),")^2/",paste(sy),"^2)-(2*",paste(rho),"*(x-",paste(mx),")
      *(y-",paste(my),")/(",paste(sx,"*",sy),"))) ) ))" )
```

```
  ini <- list(mx=valorMx,my=valorMy,sx=valorSx,sy=valorSy,rho=valorRho,peso=valorPeso)
  names(ini)[1]<-paste(mx)
  names(ini)[2]<-paste(my)
  names(ini)[3]<-paste(sx)
  names(ini)[4]<-paste(sy)
  names(ini)[5]<-paste(rho)
  names(ini)[6]<-paste(peso)

  cotaSup <- c(cotaSup, valorMx+7,valorMy+7,valorSx+7,valorSy+7,0.99,5)
  cotaInf <- c(cotaInf,valorMx-7,valorMy-7,valorSx-7,valorSy-7,-0.99,0)


  if(componente==1){
    formula <- paste("z ~",paste("of"),"+",funcionNormal)

    inicio <- c(of=min(tablaDatos[,3]),ini)
    cotaSup <- c(min(tablaDatos[,3])+(max(tablaDatos[,3])/2), cotaSup)
    cotaInf <- c(min(tablaDatos[,3])-(max(tablaDatos[,3])/2), cotaInf)

  }
  if(componente>1){
    #sumamos las demás componentes al modelo completo
    formula <- paste(formula,"+",funcionNormal)
    inicio <- c(inicio,ini)
  }


}#fin del for


#...lanzamos el ajuste con la fórmula generada...
ajuste <- nlsLM(formula,
                data=tablaDatos,
                start=inicio,
                #par=list(mx=5,my=3,sx=1,sy=1,rho=0.8),
                upper=cotaSup,
                lower=cotaInf,
                control=nls.lm.control(maxiter=70,nprint=0),#ptol, ftol, factor...
                trace=T
                #jac=as.expression(calculaGradiente())
                )

#print(summary(ajuste))


tablaDatosAjustados <- data.frame(tablaDatos$x,tablaDatos$y,predict(ajuste))
estimParametros <- matrix(coef(ajuste)[-1],nrow=nComp,byrow=T)
offsAjustado <- coef(ajuste)[1]
errorOffsAjustado <- summary(ajuste)$coefficients[1,2]
errorEstimParametros <- matrix(summary(ajuste)$coefficients[-1,2],nrow=nComp,byrow=T)

cat("parámetros ajustados: \n")
cat("media x   media y    sigma x    sigma y    coefCorr    peso")
print(estimParametros)
cat("offset para la mixtura \n")
print(offsAjustado)

Sys.sleep(5)

valoresAjuste <- list(tablaDatosAjustados,estimParametros,errorEstimParametros,offsAjustado,errorOffsAjustado)
```

```
  return(valoresAjuste)

}#fin de la funcion ‘‘algoritmoLMLibreria’’



##############################################
#FUNCION PARA PROCESAR LA TABLA DE DATOS DE SALIDA
##############################################
procesaTablaSalida <- function(tablaDatosAjuste){
  #‘‘desnormalizamos’’ los valores ajustados y sumamos el offset que
  # habíamos restado en ‘‘procesaTablaEntrada’’
  tablaDatosAjuste[,3] <- tablaDatosAjuste[,3]*sumaDatosEntrada
  tablaDatosAjuste[,3] <- round(tablaDatosAjuste[,3]+offs)

  matrizAjuste <- matrix(data=tablaDatosAjuste[,3],nrow=nrow(DATOS_IN),ncol=ncol(DATOS_IN),byrow = FALSE)
  return(matrizAjuste)
}


##############################################
#funcion para escribir datos
##############################
datosSalida <- function(estimAjustados,datosAjustados,errorEstimAjustados){
  txtDatosOUT <- "C:\\Users\\Usuario\\Desktop\\plugInDM\\datosImgDM_IN.txt"
  txtEstimadoresOUT <- "C:\\Users\\Usuario\\Desktop\\plugInDM\\estimImgDM_IN.txt"
  txtErrorEstimadoresOUT <- "C:\\Users\\Usuario\\Desktop\\plugInDM\\errorEstimImgDM_IN.txt"

  write.table(estimAjustados,txtEstimadoresOUT, row.names=F, col.names=F)
  write.table(datosAjustados,txtDatosOUT, row.names=F, col.names=F)
  write.table(errorEstimAjustados,txtErrorEstimadoresOUT, row.names=F, col.names=F)


  #esribimos datos y estimadores en una columna, habiendolos tomado de la matriz por filas
  txtDatosOUTColumna <- "C:\\Users\\Usuario\\Desktop\\plugInDM\\ColumnaDatosImgDM_IN.txt"
  txtEstimadoresOUTColumna <- "C:\\Users\\Usuario\\Desktop\\plugInDM\\ColumnaEstimImgDM_IN.txt"
  txtErrorEstimadoresOUTColumna <- "C:\\Users\\Usuario\\Desktop\\plugInDM\\ColumnaErrorEstimImgDM_IN.txt"
  vectorDatos <- numeric(0)
  vectorEstim <- numeric(0)
  vectorErrorEstim <- numeric(0)

  for(fil in 1:nrow(datosAjustados)){
    for(col in 1:ncol(datosAjustados)){
      vectorDatos <- c(vectorDatos,datosAjustados[fil,col])
    }
  }
  write.table(as.matrix(vectorDatos),txtDatosOUTColumna, row.names=F, col.names=F)

  for(fil in 1:nrow(estimAjustados)){
    for(col in 1:ncol(estimAjustados)){
      vectorEstim <- c(vectorEstim,estimAjustados[fil,col])
      vectorErrorEstim <- c(vectorErrorEstim,errorEstimAjustados[fil,col])
    }
  }
  write.table(as.matrix(vectorEstim),txtEstimadoresOUTColumna, row.names=F, col.names=F)
  write.table(as.matrix(vectorErrorEstim),txtErrorEstimadoresOUTColumna, row.names=F, col.names=F)
}



#####################################################################
```

```
####  MAIN  ##################################################
#########################################################################
#RECIBIMOS DATOS Y ESTIMADORES
datosEntrada()
#PONEMOS LA MATRIZ DE ENTRADA COMO TABLA
tablaDatosIn <- convierteMatrizATabla(DATOS_IN)
#PROCESAMOS LA TABLA
tablaDatosInProc <- procesaTablaEntrada(tablaDatosIn)

#EJECUTAMOS EL ALGORITMO
t <- proc.time()
nuevosDatos <- algoritmoLMLibreria(tablaDatosInProc)
proc.time()-t

tablaDatosAjustados <- as.data.frame(nuevosDatos[1])
estimadoresAjustados <- as.data.frame(nuevosDatos[2])
errorEstimadoresAjustados <- as.data.frame(nuevosDatos[3])
offsetAjustado <- as.numeric(nuevosDatos[4])
errorOffsetAjustado <- as.numeric(nuevosDatos[5])


#PROCESAMOS LA TABLA DE DATOS AJUSTADOS Y LA ESCRIBIMOS COMO MATRIZ
DATOS_OUT <- procesaTablaSalida(tablaDatosAjustados)


#ESCRIBIMOS LOS DATOS AJUSTADOS...
datosSalida(estimadoresAjustados,DATOS_OUT,errorEstimadoresAjustados)
```

Apéndice E

Otras R-funciones y R-scripts

E.1. Prueba de normalidad

E.1.1. Descripción

Script usado en la sección 2.1 para realizar el test de Shapiro-Wilk a distintas filas y columnas (en concreto, aquí se presenta para los picos visibles en la figura 2.1) de la matriz de datos.

E.1.2. Código en R

```
#cargamos la matriz en la variable m
m<-read.table("C:\\Users\\Usuario\\Desktop\\TFM\\MiTrabajoV1\\ImagenReal\\datosImgDM_OUT.txt")
#le restamos el mínimo de m a todos los elementos
m<-m-min(m)#(max(m)-min(m))/min(m)
#y la ponemos como variable tipo matriz (por defecto es data.frame)
m<-as.matrix(m)

#tomamos una fila de m
mFila <- m[7,]
#y la ploteamos
x <- seq(1:length(mFila))
#ploteamos la gráfica del perfil de superficie
plot(x,mFila, xlab="pixeles x", ylab="brillo")

#cogemos la zona que comprende cada pico
mFilaPico1 <- mFila[1:16]
mFilaPico2 <- mFila[17:length(mFila)]
#y sumamos sus elementos (total de eventos detectados)
sumaFilaPico1 <- sum(mFilaPico1)
sumaFilaPico2 <- sum(mFilaPico2)
#calculamos un vector con elementos proporcionales, pero de suma aproximadamente 50
#puesto que el test de S-W se aplica a muestras pequeñas (incluso menores)
mFila50Pico1 <- round((50/sumaFilaPico1)*mFilaPico1)#sum(mFila50Pico1)
mFila50Pico2 <- round((50/sumaFilaPico2)*mFilaPico2)#sum(mFila50Pico2)

#ahora generaremos un vector (para cada pico) que tenga cada valor que puede tomar
#la variable aleatoria (pixeles x) tantas veces como indique la frecuencia asociada a dicho valor
```

```
#Este vector contendrá lo que serían los resultados de cada observación.
valoresPico1 <- integer(0)
for(i in 1:length(mFila50Pico1)){
  n <- mFila50Pico1[i]
  valoresPico1 <- c(valoresPico1, rep(i,times=n))
}#length(valoresPico1)
valoresPico2 <- integer(0)
for(i in 1:length(mFila50Pico2)){
  n <- mFila50Pico2[i]
  valoresPico2 <- c(valoresPico2, rep(i,times=n))
}#length(valoresPico2)

#aplicamos el test de S-W a estos vectores
#...en nuestro caso, al primer pico...
shapiro.test(valoresPico1)
#...y al segundo
shapiro.test(valoresPico2)
```

E.2. Cálculo del gradiente

E.2.1. Descripción

Función que calcula el gradiente de una normal bivariante multiplicada por un factor ("peso"). Función usada en algunas pruebas.

E.2.2. Código en R

```
####################################3
#FUNCION PARA CALCULAR EL GRADIENTE
####################################3
calculaGradiente <- function(){
  #fórmula de UNA normal bivariante
  exp1 <- expression( peso*(( 1/(2*pi*sx*sy*sqrt(1-rho^2)) )*( exp( (-1/(2*(1-rho^2)))*(((x-
  mx)^2/sx^2)+((y-my)^2/sy^2)-(2*rho*(x-mx)*(y-my)/(sx*sy))) ) )) )

  #calculamos el gradiente de una normal (una de las componentes)....
  #...el gradiente de la mixtura será un vector tal que (gradiente_comp1,...,gradiente_compG)
  parcialmx <- D(exp1, "mx")
  parcialmy <- D(exp1, "my")
  parcialsx <- D(exp1, "sx")
  parcialsy <- D(exp1, "sy")
  parcialrho <- D(exp1, "rho")
  parcialpeso <- D(exp1, "peso")

  gradiente <- c(parcialmx,parcialmy,parcialsx,parcialsy,parcialrho,parcialpeso)

  return(gradiente)
}
```

E.3. Implementación del algoritmo de Levenberg-Marquardt

E.3.1. Descripción

Implementación del algoritmo realizada y usada en algunas pruebas. Recibe una tabla de datos con tres columnas (coordenadas x, coordenadas y y brillos) y devuelve los parámetros ajustados. Aunque da buenos resultados, obteníamos tiempos de ejecución muy altos y se optó por usar la librería *lm.minpack* en la que hay funciones que lo desarrollan. Al final, quedó como ejercicio académico, aun así se presenta porque funciona y permitió conocer tanto R como el propio algoritmo más a fondo.

E.3.2. Código en R

```
algoritmoLM_V2 <- function(tablaDatos){

  estimParametros <- ESTIMADORES_INICIALES


  #y el número de componentes de la mixtura
  nComp <- nrow(ESTIMADORES_INICIALES)

  gradiente <- c()
  lambda <- 10

  #definimos parametros para medir bondad del ajuste
  SCE <- 1
  SCT <- sum((tablaDatos[,3]-mean(tablaDatos[,3]))^2)
  RCuad <- 0

  #definimos un iterador para enumerar las mismas
  iterador <- 1

  ####iniciamos algoritmo
  while(SCE > 0.001){

  ##############################################################################
  #PASO 1: GENERAMOS MODELO DE MIXTURAS Y GRADIENTE#########################
    #con ‘‘componente’’ recorremos las normales que componen la mixtura,
    #y generamos la expresión para el modelo de mixtura de normales
    for(componente in 1:nComp){

      mx <- estimParametros[componente,1]
      my <- estimParametros[componente,2]
      sx <- estimParametros[componente,3]
      sy <- estimParametros[componente,4]
      rho <- estimParametros[componente,5]
      peso <- estimParametros[componente,6]

      #la funcion expresada como cadena de caracteres
      funcionNormal <- paste("(",paste(peso),"*( 1/(2*pi*",paste(sx*sy),"*sqrt(1-",paste(rho),"^2)) )*(
       exp((-1/(2*(1-",paste(rho),"^2)))*(((x-",paste(mx),")^2/",paste(sx),"^2)+((y-",paste(my),")^2/",paste(sy),"^2)
```

```
 -(2*",paste(rho),"*(x-",paste(mx),")*(y-",paste(my),")/(",paste(sx*sy),"))) ) ))" )

#gradiente expresado explícitamente
gradiente[(6*(componente-1))+1] <- paste( "-(",paste(peso)," * ((1/(2 * pi * ",paste(sx*sy),"*
 sqrt(1 - ",paste(rho^2),"))) * (exp((-1/(2 * (1 - ",paste(rho^2),"))) * (((x -
  ",paste(mx),")^2/",paste(sx^2),") + ((y - ",paste(my),")^2/",paste(sy^2),") - (2 *
   ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),")))) * ((-1/(2 *
    (1 - ",paste(rho^2),"))) * (2 * (x - ",paste(mx),")/",paste(sx^2)," - 2 * ",paste(rho)," * (y
     - ",paste(my),")/(",paste(sx * sy),")))))) ")

gradiente[(6*(componente-1))+2] <- paste( "-(",paste(peso)," * ((1/(2 * pi * ",paste(sx * sy)," *
 sqrt(1 - ",paste(rho^2),"))) * (exp((-1/(2 * (1 - ",paste(rho^2),"))) * (((x -
  ",paste(mx),")^2/",paste(sx^2),") + ((y - ",paste(my),")^2/",paste(sy^2),") - (2 *
   ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),")))) * ((-1/(2 *
    (1 - ",paste(rho^2),"))) * (2 * (y - ",paste(my),")/",paste(sy^2)," - 2 * ",paste(rho)," * (x
     - ",paste(mx),")/(",paste(sx *sy),")))))) ")

gradiente[(6*(componente-1))+3] <- paste( "-(",paste(peso)," * ((1/(2 * pi * ",paste(sx * sy)," *
 sqrt(1 - ",paste(rho^2),"))) * (exp((-1/(2 * (1 - ",paste(rho^2),"))) * (((x -
  ",paste(mx),")^2/",paste(sx^2),") + ((y - ",paste(my),")^2/",paste(sy^2),") - (2 *
   ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),")))) * ((-1/(2 *
    (1 - ",paste(rho^2),"))) * ((x - ",paste(mx),")^2 * (2 * ",paste(sx),")/(",paste(sx^2),")^2 -
     2 * ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),") * ",paste(sy),"/(",paste(sx *
      sy),")^2))) + 2 * pi * ",paste(sy)," * sqrt(1 - ",paste(rho^2),")/(2 * pi * ",paste(sx *
       sy)," * sqrt(1 - ",paste(rho^2),"))^2 * (exp((-1/(2 * (1 - ",paste(rho^2),"))) * (((x -
        ",paste(mx),")^2/",paste(sx^2),") + ((y - ",paste(my),")^2/",paste(sy^2),") - (2 *
         ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),")))))))"  )

gradiente[(6*(componente-1))+4] <- paste( "-(",paste(peso)," * ((1/(2 * pi * ",paste(sx * sy)," *
 sqrt(1 - ",paste(rho^2),"))) * (exp((-1/(2 * (1 - ",paste(rho^2),"))) * (((x -
  ",paste(mx),")^2/",paste(sx^2),") + ((y - ",paste(my),")^2/",paste(sy^2),") - (2 *
   ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),")))) * ((-1/(2 *
    (1 - ",paste(rho^2),"))) * ((y - ",paste(my),")^2 * (2 * ",paste(sy),")/(",paste(sy^2),")^2 -
     2 * ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),") * ",paste(sx),"/(",paste(sx *
      sy),")^2))) + 2 * pi * ",paste(sx)," * sqrt(1 - ",paste(rho^2),")/(2 * pi * ",paste(sx *
       sy)," * sqrt(1 - ",paste(rho^2),"))^2 * (exp((-1/(2 * (1 - ",paste(rho^2),"))) * (((x -
        ",paste(mx),")^2/",paste(sx^2),") + ((y - ",paste(my),")^2/",paste(sy^2),") - (2 *
         ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),")))))))"  )

gradiente[(6*(componente-1))+5] <- paste( "(",paste(peso)," * (2 * pi * ",paste(sx * sy)," * (0.5
 * (2 * ",paste(rho)," * (1 - ",paste(rho^2),")^-0.5))/(2 * pi * ",paste(sx * sy)," * sqrt(1 -
  ",paste(rho^2),"))^2 * (exp((-1/(2 * (1 - ",paste(rho^2),"))) * (((x -
   ",paste(mx),")^2/",paste(sx^2),") + ((y - ",paste(my),")^2/",paste(sy^2),") - (2 *
    ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),"))))) - (1/(2 *
     pi * ",paste(sx * sy)," * sqrt(1 - ",paste(rho^2),"))) * (exp((-1/(2 * (1 -
      ",paste(rho^2),"))) * (((x - ",paste(mx),")^2/",paste(sx^2),") + ((y -
       ",paste(my),")^2/",paste(sy^2),") - (2 * ",paste(rho)," * (x - ",paste(mx),") * (y -
        ",paste(my),")/(",paste(sx * sy),")))) * ((-1/(2 * (1 - ",paste(rho^2),"))) * (2 * (x -
         ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),")) + 2 * (2 * ",paste(rho),")/(2
          * (1 - ",paste(rho^2),"))^2 * (((x - ",paste(mx),")^2/",paste(sx^2),") + ((y -
           ",paste(my),")^2/",paste(sy^2),") - (2 * ",paste(rho)," * (x - ",paste(mx),") * (y -
            ",paste(my),")/(",paste(sx * sy),")))))))"  )

gradiente[(6*(componente-1))+6] <- paste( "((1/(2 * pi * ",paste(sx * sy)," * sqrt(1 -
 ",paste(rho^2),"))) * (exp((-1/(2 * (1 - ",paste(rho^2),"))) * (((x -
  ",paste(mx),")^2/",paste(sx^2),") + ((y - ",paste(my),")^2/",paste(sy^2),") - (2 *
   ",paste(rho)," * (x - ",paste(mx),") * (y - ",paste(my),")/(",paste(sx * sy),")))))"  )


if(componente==1){
  #asignamos la primera normal a la expresión para el modelo completo
  funcionMixtura <- paste(funcionNormal)
```

```
    }
    if(componente>1){
      #sumamos las demás componentes al modelo completo
      funcionMixtura <- paste(funcionMixtura,"+",funcionNormal)

    }


  }#fin del for
  #)


##################################################
#PASO 2: CALCULAMOS ERRORES PARA EL AJUSTE (usado como condicion de finalización)
  errorAjuste <- apply(tablaDatos,1,function(fila){
                           x <- fila[1]
                           y <- fila[2]
                           z <- fila[3]

                           return(z-eval(parse(text=funcionMixtura)))})
  SCE <- sum(errorAjuste^2)
  RCuad <- 1-(SCE/SCT)


##################################################
#PASO 3: CALCULAMOS MATRIZ J
  l <- as.list(gradiente)

  J <- apply(tablaDatos,1,function(fila){
                 x <- fila[1]
                 y <- fila[2]


                 g <- lapply(l,function(elem){
                                #print(elem)
                                return(eval(parse(text=elem)))
                                })
                 g <- unlist(g)
                 return(g)
               })
  J <- as.matrix(J)
  J <- t(J)


##################################################
#PASO 4: recalculamos parámetros para la siguiente iteracion


  A <- (t(J)%*%J)+( (lambda/(2^iterador))*diag(length(gradiente)) )#lambda*diag(t(J)%*%J)#
  b <- t(J)%*%errorAjuste
  delta <- solve(A,b)

  for(componente in 1:nComp){
    for(m in 1:ncol(estimParametros)){
      estimParametros[componente,m] <- estimParametros[componente,m] +
       delta[(componente*ncol(estimParametros))-6+m]
    }
  }
```

```
    cat("iteracion",iterador,"  SCE: ",SCE,"  R^2: ", RCuad)

    iterador <- iterador+1


  }#fin del while

 #imprimimos en pantalla y devolvemos resultados
  print(estimParametros)
  return(estimParametros)
}#fin de la funcion ‘‘algoritmoLM_V2’’
```

Referencias

• Bibliográficas

Experimentales

[1] E. Snoeck - A. Lubk - C. Magén, *Structural characterization of ferroic and multiferroic nanostructures by advanced tem techniques*, Ch. 10 *Nanoscale Ferroelectrics and Multiferroics: Key Processing and Characterization issues, and Nanoscale Effects*, Ed. by M. Algueró, J. Marty Gregg and L. Mitoseriu, Ed.: John Wiley & Sons (1st Edition), 2015.

[2] Williams, D. B. - Carter, C. B., *Transmission Electron Microscopy. A Textbook for Materials Science*, Ed.: Springer (2nd ed.), 2009.

[3] B. Bhushan, *Handbook of Nanotechnology*, Ed.: Springer (Berlin, Heidelberg), 2010.

[4] P. W. Hawkes - J. C. H. Spence, *Science of Microscopy*, Ed.: Springer (New York, NY) 2007.

Matemáticas

[5] Héctor Javier Moyano Niño (Dir. Henry Lamos Díaz), *Mezclas finitas de distribuciones normales: una alternativa para clasificar* (TFG), Universidad Industrial de Santander, 2007.

[6] J. A. Cristóbal Cristóbal, *Lecciones de inferencia estadística*, Ed.: Prensas Universitarias de Zaragoza, 2003.

[7] Pierre Gravel - Gilles Beaudoin - Jacques A. De Guise, *A method for modeling noise in medical images*, IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 23, NO. 10, OCTOBER 2004.

[8] D.M. Titterington - A.F.M. Smith - U.E. Makov, *Statistical Analysis of Finite Mixture Distributions*, Ed.: John Wiley & Sons (New York, London, Sydney), 1985.

Software R

[9] Christian Ritz - Jens Carl Streibig, *Nonlinear regression with R*, Ed.: Springer, 2008.

• Digitales

Experimentales

[10] http://www.uma.es/sme/nueva/Tecnicas.php

Matemáticas

[11] https://es.wikipedia.org/wiki/Distribuci %C3 %B3n_normal_multivariante

[12] https://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm

[13] http://media4.obspm.fr/public/VAU/instrumentacion/observar/analizar/ruido-poisson/APPRENDRE.html

Software Digital Micrograph

[14] *Digital Micrograph: basic and advanced scripting*, Berndhard Shaffer - Bernd Kraus.

[15] *How to script... Digital Micrograph Scripting Handbook*, Berndhard Shaffer

[16] http://portal.tugraz.at/portal/page/portal/felmi/DM-Script

[17] http://temdm.com/

[18] http://digitalmicrograph-scripting.tavernmaker.de/HowToScript_index.htm

[19] http://www.dmscripting.com/

[20] http://matwww.technion.ac.il/Mika/manuals/DigitalMicrograph%20User%20Guide.pdf

[21] http://www.gatan.com/resources/scripts

Software R

[22] https://cran.r-project.org/

[23] https://cran.r-project.org/web/packages/minpack.lm/index.html

[24] http://adv-r.had.co.nz/

[25] https://rmazing.wordpress.com/

[26] http://www.rdocumentation.org/

Posts

[27] http://stackoverflow.com/questions/29820176/how-to-transfer-to-connect-data-between-digital-micrograph-and-r

[28] http://stackoverflow.com/questions/32014758/to-create-an-object-of-some-class-in-a-listener

[29] http://stackoverflow.com/questions/32026093/to-create-an-object-of-some-class-in-a-listener-part-ii