Tesis de Licenciatura

“Extensiones a SNARK93, un sistema de programación para evaluación de algoritmos de reconstrucción de imágenes”

Alumno: Pablo Sala
Director: Dr. Hugo Scolnik

Departamento de Computación, FCEyN, UBA.

Resumen

En el área de reconstrucción de imágenes, siempre se desea comparar dos o más técnicas de reconstrucción y medir sus méritos relativos. El sistema de programación SNARK93 provee un marco de trabajo uniforme en el cual implementar algoritmos de reconstrucción, evaluar su performance y visualizar los resultados. En el presente trabajo tal sistema y sus herramientas fueron extendidos para permitir su funcionamiento con imágenes representadas, no sólo mediante pixels, sino también por medio de blobs. Se usó además SNARK93 para implementar el algoritmo ACCAV2 para solución de grandes sistemas de ecuaciones propuesto en [SCO/01] y aplicarlo al problema de reconstrucción de imágenes, comparándolo contra dos de los algoritmos usados actualmente en esta disciplina y se obtuvieron interesantes resultados.

Abstract

In the area of image reconstruction, it is often desirable to compare two or more reconstruction techniques and assess their relative merits. The programming system SNARK93 provides a uniform framework in which to implement algorithms and evaluate their performance and visualize the results. In the present work such system and its tools were extended to allow their work with images represented, not only through the use of pixels, but also by means of blobs. SNARK93 was used to implement the algorithm ACCAV2 for solution of large systems of equations presented in [SCO/01] and apply it to an image reconstruction problem, comparing it to two of the algorithms presently used in this discipline. Interesting results were obtained.

Introducción

En el presente trabajo se portó al ambiente Windows el sistema de programación SNARK93 [BRO/93], desarrollado por investigadores del área de reconstrucción de imágenes para evaluación de performance y comparación de algoritmos de reconstrucción, y se lo extendió para soportar la representación de imágenes mediante elementos de volumen esférico simétrico (blobs), en su uso para reconstrucción, visualización y análisis de algoritmos. Se uso además este sistema para comparar la performance de un nuevo algoritmo desarrollado para resolver grandes sistemas de ecuaciones, (ACCAV2 [SCO/01]), aplicado al problema de reconstrucción de imágenes, versus dos algoritmos usados actualmente en esta disciplina. Se presentan los resultados obtenidos, los cuales muestran que el nuevo algoritmo supera a los actuales en la tarea de reconstrucción.

El trabajo está organizado del siguiente modo: primero se da una introducción a las técnicas usadas para reconstrucción de imágenes a través de proyecciones. Se exponen dos algoritmos actualmente en uso para resolver este tipo de problemas, y se da la definición del algoritmo ACCAV2. Luego se presenta el sistema de programación SNARK93 y se detallan las modificaciones realizadas a este. Se muestran los resultados de las comparaciones entre ACCAV2 y los otros algoritmos. Finalmente se presentan las conclusiones obtenidas.

1. Reconstrucción de imágenes a través de proyecciones.

Los problemas de reconstrucción de imágenes difieren ampliamente según el área de aplicación en el que aparecen. Aun en un mismo campo – tal como la medicina o la industria – aparecen una diversa cantidad de problemas de reconstrucción debido a los diferentes métodos de recolección de datos. A pesar de esas diferencias, existe una naturaleza matemática común a los problemas de reconstrucción: hay una distribución desconocida (en dos o tres dimensiones) de algún parámetro físico. Este parámetro podría ser, por ejemplo, los coeficientes lineales de atenuación de rayos x en el tejido humano, o los coeficientes de atenuación del material en un reactor nuclear, o la densidad de electrones en la corona solar. Un número finito de integrales de línea de este parámetro puede estimarse a partir de mediciones físicas, y con esto se desea obtener una estimación de la distribución del parámetro original. Por ejemplo, en el caso de tomografía computada por transmisión de rayos x, la atenuación total del haz de rayos x entre la fuente y el detector es aproximadamente igual a la integral del coeficiente lineal de atenuación a lo largo de la línea entre la fuente y el detector.

Denotemos por f(x, y) la imagen que debe ser reconstruida, esto es, la función bidimensional que representa la distribución espacial del parámetro físico de interés. Aunque f es desconocida a priori, en la mayoría de los casos se sabe que su distribución esta espacialmente acotada, por lo cual f es 0 fuera de una región finita Ω del plano. Es a veces más conveniente expresar f en términos de coordenadas polares (r, ϕ) en lugar de coordenadas cartesianas (x, y) (ver Figura 1.1) En este caso escribimos:

$$f(x, y) = f(r \cos \phi, r \operatorname{sen} \phi). \qquad (1.1)$$

Una recta (rayo) en el plano es especificada por dos parámetros: su distancia (con signo) l del origen y su ángulo θ con respecto al eje y. Denotamos por p(l, θ) la función de dos variables cuyo valor en (l, θ) esta definido como la integral de línea de f a lo largo de la línea L especificada por l y θ,

$$p(l,\theta) = \int_L f(x,y) \quad dz \qquad (1.2)$$

donde los limites de integración dependen, en general, de l, θ y Ω.

El problema de reconstrucción a partir de proyecciones es encontrar f(x, y) dado p(l, θ). Como problema matemático, donde f y p son funciones (desconocida y conocida, respectivamente), encontrar f requiere resolver la ecuación integral (1.2). En realidad la solución fue publicada por Radon en 1917 como una formula de inversión expresando f en términos de p.

El problema matemático expresado arriba y resuelto por Radon, representa una abstracción idealizada del problema tal como ocurre en diversas aplicaciones. En la práctica a partir de los datos de proyecciones discretas, que son estimaciones de p para un número finito de rayos, queremos encontrar la función de la imagen bidimensional que es una estimación por reconstrucción del objeto desconocido. Además de la discretizacion y precisión finita de las mediciones, debemos notar que una variedad de problemas físicos pueden llevar a una significativa no-linealidad en la relación entre los datos de las mediciones y el objeto original. Sin embargo, por su tratabilidad matemática, el modelo lineal expresado por (1.2) es el punto de partida preferido para la derivación de algoritmos para reconstrucción de imágenes.

Fundamentalmente existen dos diferentes acercamientos al problema de reconstrucción de imágenes. En uno de ellos, el problema es formulado para funciones continuas f y p, y una formula de inversión es derivada en este modelo continuo. Dado que este enfoque envuelve el estudio matemático de transformadas y sus inversas, es generalmente llamado enfoque por *métodos de transformadas*. El acercamiento alternativo es discretizar las funciones involucradas en el problema. El objeto y las mediciones se transforman en vectores en un espacio Euclideano finito y métodos de diferentes ramas de la matemática, mayormente álgebra lineal y teoría de la optimización, son usados para estudiar el modelo y construir algoritmos para resolver el problema. Este acercamiento es llamado *expansión finita en series* o *modelo totalmente discreto*. [CEN/97]

Los métodos usados para la reconstrucción pueden ser o bien *analíticos* o bien *iterativos* en su naturaleza. Los algoritmos analíticos [LEW/83] son rápidos, pero su desventaja radica en que no son capaces de modelar adecuadamente las características físicas y estadísticas del proceso de adquisición de datos. Uno generalmente se encuentra en una situación donde los datos procedentes de las mediciones contienen ruido y solo un monto moderado de datos esta disponible. En tales situaciones es mejor usar algoritmos iterativos

[CEN/83], los cuales son más lentos pero pueden lidiar mejor con datos ruidosos y un muestreo inadecuado del espacio de mediciones.

El problema de reconstrucción de imágenes por transmisión de rayos x en el modelo discreto se formula del siguiente modo: Una grilla cartesiana de elementos de imagen cuadrados, llamados *pixels*, se introduce en la región de interés, de modo que cubra toda la imagen que debe ser reconstruida. Los pixels se numeran de algún modo preestablecido, digamos de 1 (pixel superior izquierdo) a n (pixel inferior derecho) (ver Figura 1.2).

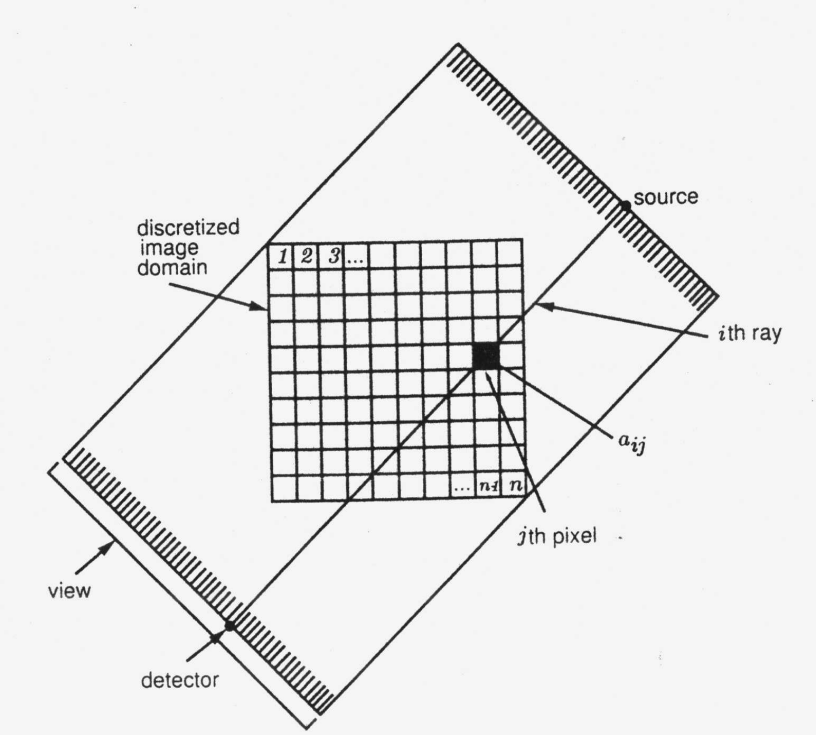

Figura 1.2. Modelo totalmente discreto para reconstrucción de imágenes.

La función de atenuación de rayos x se asume que toma un valor constante x_j en todo el pixel j-ésimo, para j = 1, 2, ..., n. Las fuentes y detectores que transmiten y reciben energía se consideran como puntos y los rayos entre estos se asumen que son líneas. También se asume que la intersección del i-ésimo rayo con el j-ésimo pixel, denotado por a_{ij} para todo i = 1, 2, ... m, j = 1, 2, ..., n, representa los pesos de la contribución del j-ésimo pixel a la atenuación total de la energía a lo largo del i-ésimo rayo.

La medición física de la atenuación total a lo largo del i-ésimo rayo, denotada por y_i, representa la integral de línea de la función desconocida de atenuación a lo largo del camino del rayo. Por lo tanto, en este modelo discreto, la integral de línea resulta ser una suma finita y todo el modelo es descrito por un sistema de ecuaciones lineales:

$$\sum_{j=1}^{n} x_j \cdot a_{ij} = y_i, \quad i = 1,2,\cdots,m \qquad (1.3)$$

En notación matricial escribimos (3) como:

$$y = A\,x, \quad (1.4)$$

donde $y = (y_i) \in \Re^m$ es el vector de mediciones, $x = (x_j) \in \Re^n$ es el vector de imagen, y la matriz $A = (a_{ij}) \in \Re^{m \times n}$ es la matriz de proyecciones.

Podemos describir el modelo anterior de un modo diferente. Sea $\{b_j(r, \phi)\}_{j=1, \ldots, n}$ un conjunto de funciones base en coordenadas polares en el plano, dado por

$$b_i(r,\phi) = \begin{cases} 1, & si \ (r,\phi) \ \ pertenece \ \ al \ \ j-ésimo \ \ pixel \\ 0, & de \ \ lo \ \ contrario \end{cases} \qquad (1.5)$$

y llamamos:

$$f^{\circ}(r,\phi) = \sum_{j=1}^{n} x_j \cdot b_i(r,\phi) \qquad (1.6)$$

a la digitalización de f(r, ϕ) con respecto a las funciones base $\{b_j\}_{j=1, \ldots, n}$, donde x_j son los coeficientes de la expansión. Sea $\{\Re_i\}_{i=1, \ldots, m}$ un conjunto de funcionales lineales y continuos los cuales asignan a una imagen f(r, ϕ) números reales $\{\Re_i f\}_{i=1, \ldots, m}$. En nuestro caso, $\Re_i f$ es la integral de línea de f(r, ϕ) a lo largo del i-ésimo rayo. Ahora bien, y_i es sólo una aproximación de $\Re_i f$ debido a las imprecisiones en las mediciones físicas. $\Re_i f$ es cercano a $\Re_i f^{\circ}$ por la continuidad de $\Re_i$ y dado que usamos f° como nuestra aproximación de f. Usando la linealidad de $\Re_i$ podemos entonces escribir:

$$y_i \approx \Re_i f \approx \Re_i f^{\circ} = \sum_{j=1}^{n} x_j \Re_i b_j(r,\phi) = \sum_{j=1}^{n} x_j a_{ij} \qquad (1.7)$$

donde $a_{ij} = \Re_i b_j(r, \phi)$, llegando así nuevamente a (1.3).

La flexibilidad del enfoque por expansión en series puede ser apreciada de esta descripción. Primero, los funcionales $\Re_i$ no tienen por que ser las integrales de línea que usamos; pueden tratarse de cualquier otros funcionales que puedan surgir en distintos problemas de reconstrucción si sus propiedades básicas hacen posible a un proceso de modelización similar. Como un ejemplo, mencionaremos solo el caso de Tomografía Computada por Emisión (ECT) donde los funcionales $\Re_i$ son las *integrales de línea atenuadas* y el enfoque de expansión en series aplicado a calcular simultáneamente los coeficientes de actividad y atenuación lleva a un sistema de ecuaciones no-lineales, en lugar del sistema lineal (4). De igual modo la familia de funciones base puede elegirse en modo diferente que en (1.5). Cuando consideramos el modelo discreto en reconstrucción de imágenes, debemos tener en cuenta algunas características del problema. El sistema (1.3) es extremadamente grande – con n (cantidad de pixels) y m (cantidad de rayos) del orden de magnitud de 10^5 cada uno – para poder producir imágenes con buena resolución. La matriz A del sistema es muy esparsa, con menos del 1% de sus entradas no nulas, debido a que solo unos pocos pixels tienen una intersección no vacía con cada rayo particular. El sistema es a veces indeterminado debido a la falta de información, pero generalmente esta muy sobredeterminado, en cuyo caso es muy probablemente inconsistente (esto es, para un vector y, no existe un x que satisface Ax = y). Mas aun, podríamos tener razón para creer que la solución algebraica exacta del sistema, aun si existiera y pudiera ser calculada exactamente, no es más deseable, en términos del problema de reconstrucción, que alguna otra "solución" definida en forma diferente. Tal creencia puede ser basada en evidencia de inexactitud en las mediciones o corrupción de los datos debido a ruido, o el hecho de que el problema original ha sido discretizado. [CEN/97]

En la descripción anterior usamos pixels como funciones base. Sin embargo las aproximaciones resultantes del uso de pixels son funciones constantes que tienen indeseables bordes artificiales perfectamente bien marcados; parece más apropiado usar funciones con una transición suave de uno a cero [GAR/01]. De hecho, estudios recientes han mostrado que existen funciones base mas apropiadas que los pixels y que tienen

estas características de transición suave [LEW/90, MAT/96a, JAC/98, MAT/96b, HER/01, GAR/01, LEW/92]. Entre ellas, y de las cuales se han escrito numerosos trabajos, están las funciones de ventana Kaiser-Bessel generalizadas. Fueron introducidas en el campo de la tomografía medica en 1990 por Robert Lewitt [LEW/90]. Él las llamó *blobs* y demostró que son superiores a los pixels tanto para reconstrucción como para visualización de las imágenes reconstruidas [MAT/96b], [LEW/92]. Los blobs son funciones con simetría esférica y una suave transición de uno a cero.

Los blobs $b_{n,\,m,\,a,\,\alpha}(x)$, donde $x \in \Re^n$, dependen de cuatro parámetros: la dimensión n de la imagen (por ej. 2 para 2D), el orden m de la "función de Bessel modificada" de la primer clase $I_m(.)$ la cual es usada para definir el blob, el radio a del blob, y el parámetro de forma α el cual determina la forma global del blob. Valores pequeños para α resultan en blobs teniendo picos angostos y largas colas. La forma general de un blob es:

$$b_{m,a,\alpha}(x) = \begin{cases} \dfrac{1}{I_m(\alpha)} \left(\sqrt{1 - \left(\|x\| / a \right)^2} \right)^m I_m\left(\alpha \sqrt{1 - \left(\|x\| / a \right)^2} \right), & si \quad \|x\| < a \\ 0, & de \quad lo \quad contrario \end{cases} \qquad (1.8)$$

Nótese que esta ecuación no depende de la dimensionalidad de la imagen dado que los valores de la función sólo dependen en la norma del argumento. Los valores de los parámetros deben ser optimizados de acuerdo a las características de los datos. Los valores dados a cada parámetro influenciaran los resultados obtenidos por el algoritmo de reconstrucción, por tanto la selección adecuada de ellos es de altísima importancia. La Figura 1.3 muestra un grafico de la función de un blob de radio 2 para distintos alfa:

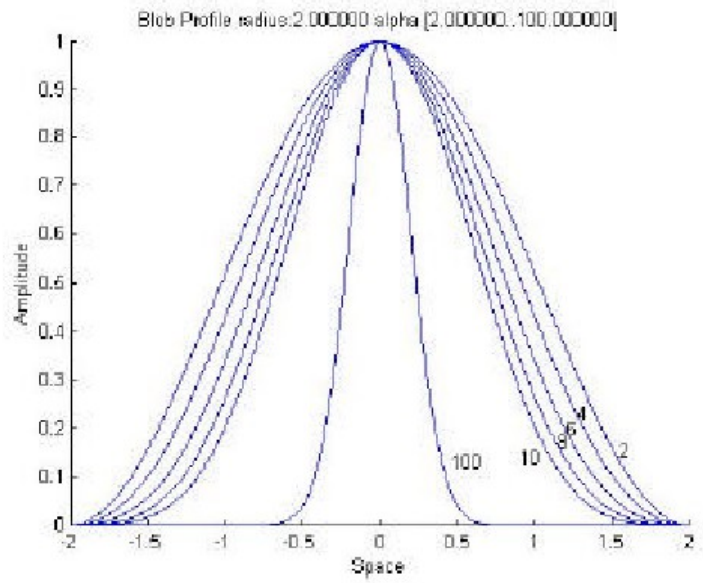

Figura 1.3. Grafico de la función de un blob de radio 2 para distintos valores de α.

Debe definirse además una *grilla*, la cual consiste en un conjunto de puntos $\{p_j\}_{j=1,\,\ldots,\,B}$ (B es el número de blobs), donde p_j es la coordenada del centro del blob j-ésimo. La elección del arreglo espacial del conjunto de puntos es importante para la calidad de la reconstrucción final. Todos los blobs se definen usando los mismos parámetros m, a y α. Resulta entonces:

$$b_j(r, \phi) = b((r \cos \phi, r \operatorname{sen} \phi) - p_j) \qquad (1.9)$$

En 2-D, la grilla *hexagonal* resulta de particular interés, ya que es la más eficiente en el sentido de que logra "cubrir" una superficie con la menor cantidad de puntos. En la Figura 1.4 puede apreciarse la distribución de los centros de blobs p_j en este tipo de grilla: $p_j \in \{d \,.\, (k + ½\,(r \bmod 2),\ \sqrt{3}.r\,/6\,) / k,\ r \in N_{\geq 0}\}$.

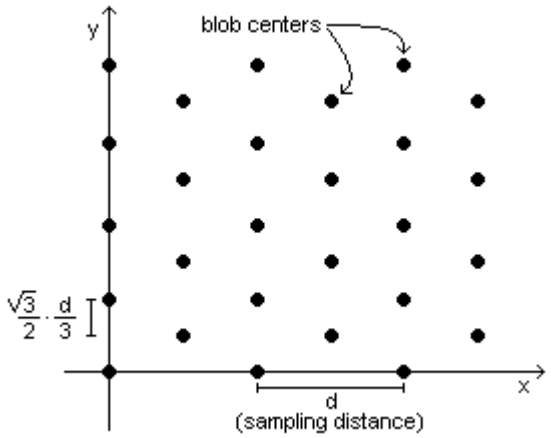

Figura 1.4. Grilla Hexagonal

2. Algoritmos para reconstrucción de imágenes.

Existen diversos algoritmos propuestos para el problema de reconstrucción de imágenes. Aquí se dará una descripción de ART y CAV, algoritmos de uso actual en esta disciplina, y de un nuevo algoritmo presentado en [SCO/01] para solución de grandes sistemas de ecuaciones, el cual en la sección 5 de este trabajo es aplicado a la reconstrucción de imágenes y comparado en su performance contra los dos algoritmos aquí descritos.

2.1 ART

El algoritmo ART (*Algebraic Reconstruction Technique*), también conocido como algoritmo de Kaczmarz, para resolver un sistema de ecuaciones $\langle a_i, x\rangle - b_i = 0$, para $i \in I$, donde $a_i \in \Re^n$ y $b_i \in \Re$ están dados, tiene la siguiente definición [CEN/97]:

Paso 0: (Inicialización) $x^0 \in \Re^n$ es arbitrario,

Paso 1: (Paso Iterativo)

$$x^{k+1} = x^k + \lambda_k \frac{b_{i(k)} - \langle a_{i(k)}, x^k \rangle}{\left\| a_{i(k)} \right\|^2} a_{i(k)} \qquad (2.1.1)$$

Parámetros de relajación: Para todo $k \geq 0$, $\varepsilon_1 \leq \lambda_k \leq 2 - \varepsilon_2$, para algún $\varepsilon_1, \varepsilon_2 > 0$.

Control: La sucesión $\{i(k)\}_{k\geq 0}$ es casi cíclica sobre I.

Podemos interpretar el algoritmo de acuerdo a (2.1.1). Dados la iteración actual x^k, y el hiperplano $H_{i(k)} = \{x \in \Re^n / \langle a_{i(k)}, x\rangle = b_{i(k)}\}$, determinado por la i(k)-ésima ecuación, el nuevo x^{k+1} cae en la línea a través de x^k la cual es perpendicular a $H_{i(k)}$. Para el caso de relajación unitaria, $\lambda_k = 1$, x^{k+1} es la proyección ortogonal

de x^k en $H_{i(k)}$ (ver Figura 2.1.1). La opción de relajación permite que la próxima iteración x^{k+1} sea un punto dentro del segmento de línea que une a x^k con su reflexión ortogonal respecto a $H_{i(k)}$.

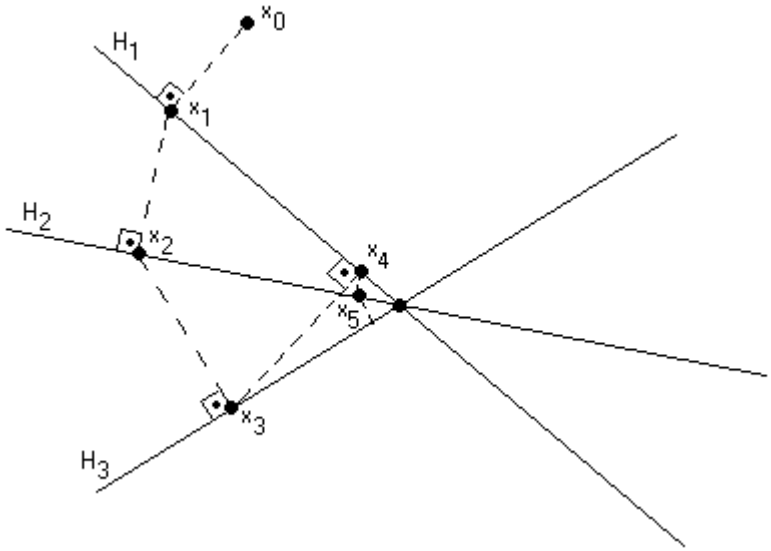

Figura 2.1.1. ART (con relajación unitaria) para resolver un sist. lineal de ecuacs., ilustrado para n=2 y m=3.

2.2 CAV

CAV (*Component Averaging*) fue presentado en 2000 por Yair Censor et al. [CEN/00] como una técnica de solución eficiente para sistemas grandes y esparsos de ecuaciones lineales. El algoritmo es el siguiente:

Inicialización: $x^0 \in \Re^n$ es arbitrario,

Paso Iterativo:

$$x^{k+1} = x^k + \lambda_k \sum_{i=1}^{m} \frac{b_i - \langle a_i, x^k \rangle}{\sum_{l=1}^{n} s_l (a_{il})^2} a_i \qquad (2.2.1),$$

donde $\{\lambda_k\}_{k\geq 0}$ son parámetros de relajación y $\{s_j\}_{j=1,\ldots,n}$ es el número de elementos no nulos $a_{i,j} \neq 0$ en la j-ésima columna de A.

En [CEN/00] se demuestra que este algoritmo con $\lambda_k = 1$ para todo $k \geq 0$ genera sucesiones $\{x_k\}$ convergentes, independientemente del punto inicial x^0 y de la consistencia o inconsistencia del sistema subyacente Ax = b. Los resultados empíricos mostrados en ese trabajo muestran que cuando se aplica CAV a problemas de reconstrucción de imágenes, éste generalmente converge más rápido que ART, y además produce mejores reconstrucciones.

2.3 ACCAV2

El algoritmo ACCAV2 (*Accelerated CAV, versión 2*) fue propuesto en 2001 por Hugo Scolnik et al. para resolver grandes sistemas de ecuaciones Ax = b, dados $A \in \Re^{m\times n}$ y $b \in \Re^m$, con m>n. ACCAV2 es la versión del algoritmo ACCAV, presentado en [SCO/01] (vèase tambièn [SCO/00, SCO/02]), para encontrar

en sistemas inconsistentes la solución $x \in \Re^n$ que minimiza $||b - Ax||$, la cual se corresponde con $A^tAx = A^tb$, bajo la hipótesis de que rango(A) = n y como consecuencia A^tA es no singular.

La definición del algoritmo es la siguiente:

Inicialización: $x^0 \in \Re^n$ es arbitrario, $Q_0 = I_n$, (I_n es la matriz identidad en $\Re^{n\times n}$)

Hacer: $b_i := b_i / || a_i ||$
$a_i := a_i / ||a_i||$

Calcular s_j para j = 1, 2, ..., n (número de elementos no nulos de cada columna de A) como se definió en CAV (2.2).

Calcular $w_i = \dfrac{1}{\sum_1^n s_l (a_{il})^2}$ para i=1, 2, ..., m

Paso Iterativo:

Para i=1, 2, ..., m calcular:
$r_i^k = b_i - \langle a_i, x^k \rangle$
$d_i^k = r_i^k \,.\, a_i$

$$\hat{d}_i^k = \begin{cases} d^k, & si \quad k = 0 \\ P_{v^\perp}(d^k), \quad con \quad v = x^k - x^{k-1}, & si \quad k > 0 \end{cases}$$

Fin Para

$$D^k = \sum_{i=1}^{m} w_i \hat{d}_i^k$$

$x^{k+1} = x^k + \lambda_k D^k$, donde:
$\lambda_k = \text{argmin}_\lambda ||x^k + \lambda D^k - x^*||^2_{AtA}$, donde x* denota una solución de $A^tAx = A^tb$.

En el mismo trabajo se demuestra que:

$$\lambda_0 = \frac{\langle Ad^0, r^0 \rangle}{\|Ad^0\|_2^2}$$

$$\lambda_k = \frac{\langle AD^k, r^k \rangle}{\|AD^k\|_2^2}, \quad k > 0$$

También se propone en el trabajo una implementación eficiente del algoritmo en la que los valores de r^{k+1}, D^{k+1}, AD^{k+1}, $||r^{k+1}||^2$ y $||AD^k||^2$, necesarios en cada paso, pueden calcularse directamente en función de sus valores en la iteración k, sin necesidad de efectuar los costosos productos matriciales.

3. SNARK93: un sistema de programación para comparación y análisis de algoritmos para reconstrucción de imágenes a través de proyecciones. [BRO/93]

3.1 Introducción:

En el área de reconstrucción de imágenes, los investigadores siempre desean comparar dos o más técnicas de reconstrucción y medir sus méritos relativos. SNARK93 provee un marco de trabajo uniforme en el cual implementar algoritmos y evaluar su performance. SNARK93 fue diseñado para tratar tanto geometrías de proyecciones paralelas como divergentes y puede crear datos de test para el uso de los algoritmos de reconstrucción. Incorpora además un cierto número de algoritmos clásicos de reconstrucción.

SNARK93 es un sistema de programación diseñado para ayudar a los investigadores interesados en el desarrollo y evaluación de algoritmos de reconstrucción. Fue desarrollado por J. A. Browne, G. T. Herman y D. Odhner, del Departamento de Radiología de la Universidad de Pennsylvania, en 1993. Es un descendiente de versiones más tempranas de SNARK, la primera de las cuales fue escrita por R. Gordon en 1970, seguida por SNARK77 ([HER/78]) y SNARK89 ([HER/89b]).

El sistema de programación SNARK93 esta implementado en FORTRAN 77 y fue diseñado para:

a) Ser capaz de tratar con varios modos de colección de datos (diferentes arreglos geométricos de fuentes y detectores de rayos-x, presencia o ausencia de baños de agua, diferentes espectros de rayos-x, etc.);
b) contener varios de los algoritmos de reconstrucción publicados;
c) ser capaz de generar phantoms descriptos matemáticamente representando realisticamente varias secciones transversales del cuerpo humano, junto con datos de proyección matemáticamente simulados, reflejando las características (incluyendo ruido) de varios posibles dispositivos de reconstrucción;
d) contener rutinas para la evaluación estadística de los algoritmos de reconstrucción;
e) ser capaz de varios modos de visualización;
f) contener subrutinas para llevar a cabo el trabajo que resulta común a la mayoría de los algoritmos de reconstrucción, como también para facilitar la incorporación de algoritmos adicionales;
g) proveer una metodología para testear la existencia de diferencias estadísticamente significativas entre algoritmos de reconstrucción [BRO/93].

SNARK93 es un conjunto de módulos FORTRAN 77, a los cuales el usuario puede agregarse nuevos módulos conteniendo los algoritmos que desea testear. Una vez compilado, el programa corre como una aplicación de cónsola y espera recibir un archivo de comandos desde stdin, en el que se le podrá describir el phantom de la imagen a reconstruir, la geometría del scanner, los algoritmos a correr con sus parámetros, y las evaluaciones a realizar en los resultados de las corridas. Como resultado de las corridas se generan archivos con las imágenes resultantes de las reconstrucciones para cada iteración de los algoritmos ejecutados; así como archivos con los resultados obtenidos en las evaluaciones. Una de las herramientas que acompaña al sistema es SnarkDisplay, desarrollada en C++ (en la versión portada a Windows), que permite visualizar tanto los phantoms como las reconstrucciones obtenidas en cada iteración. Otra herramienta del paquete es SnarkEval (C++), que permite graficar las curvas de los parámetros evaluados. Y por ultimo SuperSnark (C++), se trata de una aplicación de cónsola que lee de stdin una lista de comandos en la que debe especificarse un set de experimentos a realizar con distintos algoritmos. Con tales especificaciones, SuperSnark ejecutara SNARK93 para cada experimento, y generará un archivo de salida con conclusiones estadísticas.

Para información mas detallada sobre SNARK93, remitirse al manual ([BRO/93]) y al Apéndice de esta tesis en el que se encuentra un agregado al manual de SNARK93 con la documentación de las extensiones realizadas para soporte de blobs.

3.2 SNARK Framework:

El uso de unidades:

SNARK93 trata con cantidades medibles físicamente, tales como la distancia entre una fuente de rayos-x y un detector, o la atenuación lineal (por unidad de longitud) en un punto dentro del cuerpo humano. Los valores reales asignados a estas cantidades son dependientes en la unidad de longitud elegida. Aunque la

unidad de longitud no necesita ser especificada en SNARK93, debe mantenerse consistencia a lo largo de una corrida de SNARK93.

Asumiremos un sistema de coordenadas en dos dimensiones. El punto *(x, y)* en este sistema representa la ubicación física cuya abscisa es *x* y su ordenada *y* medidas en la unidad de longitud elegida.

Imágenes y digitalización:

Cuando hablamos de una *imagen*, asumimos que tiene dos componentes:
i) la *región de la imagen* la cual es un cuadrado cuyo centro está en el origen del sistema de coordenadas;
ii) una *función* de dos variables cuyo valor es cero fuera de la región de la imagen.

Nos referiremos al valor de la imagen en el punto *(x, y)* como la *densidad* en *(x, y)*.

Una *grilla de n-elementos* subdivide la región de la imagen en n^2 cuadrados de igual tamaño. Cada uno de estos cuadrados más pequeños lo denominaremos *pixel*.

Una *imagen digitalizada de $n \times n$* es aquella cuyo valor en el interior de un pixel de una grilla de n-elementos es uniforme.

Una *digitalización de $n \times n$* de una imagen es una imagen digitalizada de $n \times n$ tal que la integral de la imagen original sobre un pixel de una grilla de n-elementos es igual a la integral de la digitalización sobre el mismo pixel.

En SNARK93 solo se manejan imágenes digitalizadas de $n \times n$, donde n es un entero impar. El origen se ubica en el centro del pixel central.

En SNARK93 la variable NELEM es usada para denotar la cantidad de elementos en la grilla, y la variable PIXSIZ es usada para denotar la longitud del lado de un pixel. Así la región de la imagen es un cuadrado cuyas esquinas tienen coordenadas (c, c), (-c, c), (-c, -c), (c, -c), donde

$$c = \text{PIXSIZ} \times \text{NELEM} / 2$$

El usuario de SNARK93 debe especificar tanto NELEM como PIXSIZ. El alcance de la región de reconstrucción es calculado por el programa.

Rayos y sumas de rayos:

Hay dos clases de rayos en SNARK93:

i) un rayo de línea (LINE ray), el cual es una línea recta, y
ii) un rayo de banda (STRIP ray), el cual es una región del plano entre un par de líneas rectas paralelas.

Dada una imagen y un rayo, la *suma real del rayo* es la integral de la imagen a lo largo del rayo. (Puede tratarse de una integral de línea o una integral de banda.)

La *pseudo suma del rayo* es una aproximación de la suma real del rayo, la cual depende de la grilla de n-elementos utilizada para digitalizar la imagen. En el caso de un rayo LINE, la pseudo suma del rayo es la suma real del rayo de la versión digitalizada $n \times n$ de la imagen a lo largo del mismo rayo. En el caso STRIP, la pseudo suma del rayo es la integral de la imagen sobre el dominio que consiste de aquellos pixels cuyos centros caen dentro del rayo. (Equivalentemente, es la suma de los valores de los pixels cuyos centros caen dentro del rayo, multiplicados por el área del pixel.)

Con esta terminología, el *problema de reconstrucción* puede expresarse del siguiente modo. Dadas aproximaciones (basadas en mediciones físicas) de las sumas reales de rayos de una imagen, a lo largo de un cierto número de rayos, estimar los n^2 números que describen la versión digitalizada $n \times n$ de la imagen.

Geometría de una recolección de datos:

SNARK93 es capaz de manejar distintos arreglos de rayos, los cuales son típicos de lo que uno podría encontrar en la práctica.

Primeramente, el conjunto de rayos a lo largo de los cuales se recolectan datos esta dividido en un número de subconjuntos, llamados *proyecciones*, cada uno de los cuales contiene el mismo numero de rayos. Existen dos métodos básicamente distintos de recolección de datos: DIVERGENTE y PARALELO.

En geometría DIVERGENTE (ver Figura 4), una proyección consiste de un conjunto de rayos de línea (LINE) los cuales pasan a través de un punto común (la posición de la fuente). En todas las proyecciones, la fuente está a una distancia fija (RADIUS) del origen. El ángulo que forma la línea del origen a la fuente con el eje-x (llamado THETA en la Figura 3.1) se llama *ángulo de la proyección*. Los rayos en una proyección conectan la fuente a puntos (detectores) los cuales caen bien en un arco de un circulo cuyo centro está en la fuente, o bien en una línea recta. En cualquier caso, uno de los detectores (identificado como C en la Figura 4) esta en la línea que conecta la fuente con el origen, a una distancia STOD de la fuente.

Los otros detectores están espaciados simétricamente a intervalos regulares a ambos lados de C, ya sea en el arco (ARC) cuyo centro esta en la fuente o en la recta tangente (TANGENT) a ARC en C. El espaciado entre detectores (la longitud del arco ARC o de la recta tangente TANGENT entre dos detectores vecinos) puede ser especificado por el usuario.

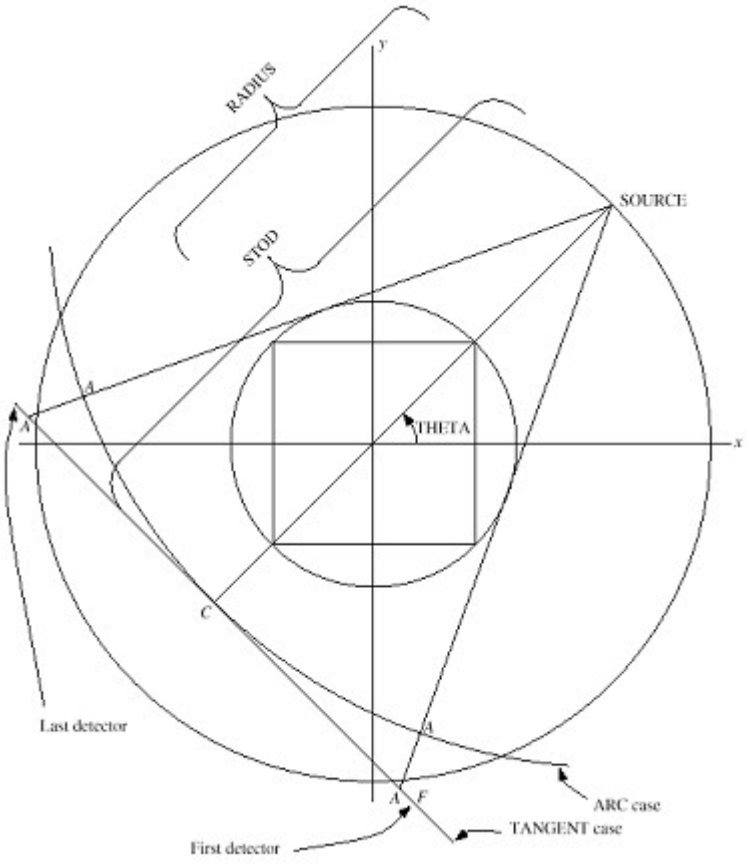

Figura 3.1. Geometría Divergente en SNARK93.

En geometría PARALELA (ver Figura 3.2), una proyección consiste en un conjunto de rayos LINE o STRIP paralelos. El ángulo que estos rayos hacen con el eje-x (denotado por THETA en la Figura 5), es llamado el ángulo de proyección. En el caso LINE uno de los rayos pasa a través del origen, en el caso STRIP el origen esta en el centro de uno de los rayos. Los otros rayos están espaciados simétricamente a intervalos regulares a ambos lados de este rayo. En el caso STRIP los rayos están dispuestos en forma contigua.

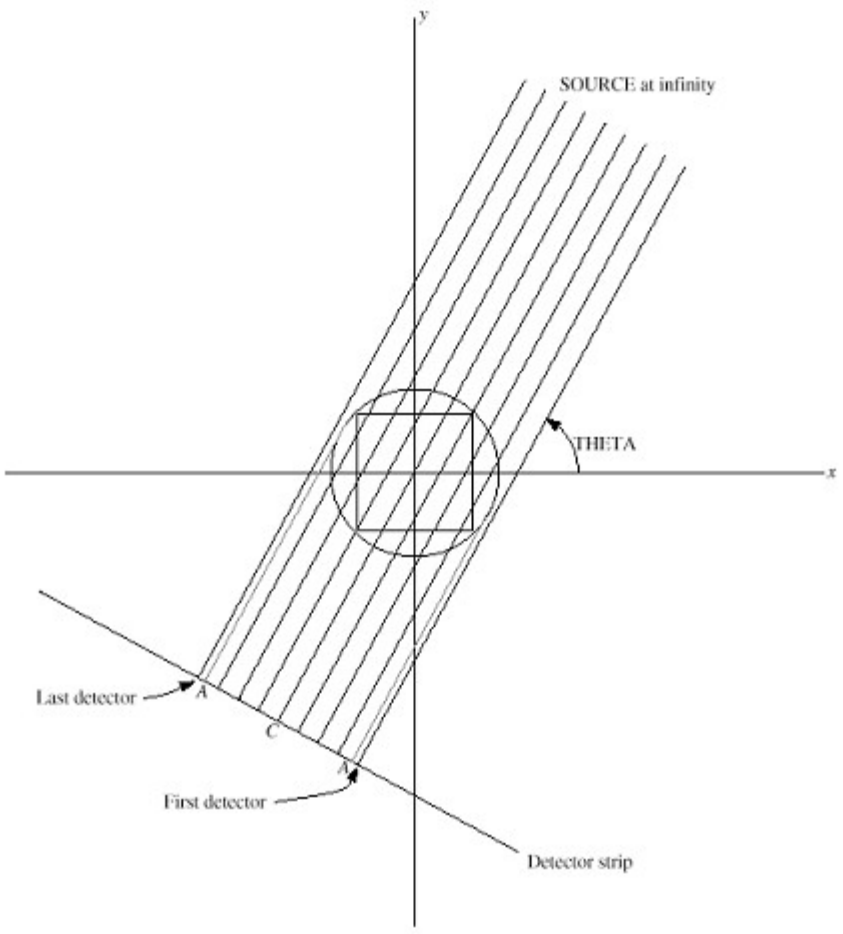

Figura 3.2. Geometría Paralela en SNARK93.

Corridas de SNARK:

Una corrida de SNARK93 consiste de una combinación de las siguientes tres fases:

i) Fase de Generación de Datos:

Durante esta fase se genera la imagen a reconstruir (*test phantom*) y los datos de proyecciones para el test phantom. Los datos de proyecciones consisten en las sumas reales de rayos del test phantom, posiblemente contaminadas por los tipos de ruido que generalmente uno puede encontrarse en un dispositivo de rayos-x usado para la colección de datos para reconstrucción.

La definición del phantom se especifica en el archivo de input por medio de comandos específicos. (Ver 5.4 en [BRO/93]) El phantom esta basado en una región de imagen la cual es un cuadrado con centro en el origen dividida en pixels de igual tamaño. Esta región esta definida por el número de pixels por lado (NELEM) y la longitud del lado de cada pixel (PIXSIZ). El origen se asume que está en el centro del pixel central.

El phantom se forma por la sobre imposición de una cantidad de objetos elementales, ubicados en posiciones elegidas, con orientaciones y tamaños deseados y valores de densidad (absorción) a elección. Los objetos elementales que pueden usarse: rectángulo, elipse, triangulo, sección circular y segmento circular. La densidad de un objeto elemental puede ser negativa. La densidad (valor de la imagen) en un punto está definida como la suma de las densidades asociadas con todos los objetos elementales que caen en ese punto. Para obtener una medida de la densidad promedio en un pixel, el usuario especifica la variable NAVE1, la cual produce el efecto de que la densidad en el interior de un pixel se determina promediando el valor de la densidad en NAVE1 × NAVE1 puntos uniformemente espaciados en el interior del pixel. En caso de radiación poli-energética, el usuario especifica la absorción en cada nivel de energía para

cada objeto elemental y una densidad promedio se obtiene como la suma pesada sobre el espectro. Así, la densidad asignada a un pixel puede ser expresada como la triple suma:

$$\frac{1}{(NAVE1)^2}\sum_{k=1}^{(NAVE1)^2}\sum_{j=1}^{NOBJ}\delta_{k,j}\sum_{i=1}^{NERGY}d_{j,i}p_i$$

con:

$$\sum_{i=1}^{NERGY}p_i = 1$$

donde NERGY es el numero de niveles de energía usados, NOBJ es la cantidad de objetos elementales en el phantom, p_i es la porción del espectro de energía en el nivel i, $d_{j,i}$ es la densidad del j-ésimo objeto elemental en el nivel de energía i, $\delta_{k,j}$ =1 si el k-ésimo de los NAVE1 x NAVE1 puntos del pixel esta en el j-ésimo objeto elemental, y $\delta_{k,j}$ =0 sino.

ii) Fase de Inicialización y Reconstrucción:

Esta fase puede subdividirse en dos subfases: la subfase de inicialización y la subfase de reconstrucción. La última no puede realizarse si no se realiza la primera en la misma corrida.

ii.i) Subfase de Inicialización:

Durante esta fase se especifican la naturaleza de la grilla de pixels o blobs para la región de reconstrucción así como la geometría para la recolección de datos.

ii.ii) Subfase de Reconstrucción:

Durante esta subfase los algoritmos de reconstrucción especificados en el archivo de input son ejecutados y las reconstrucciones y resultados son guardados en un archivo de salida.

iii) Fase de Análisis:

Durante esta fase se generan resultados del phantom y las reconstrucciones, sobre la base de la información generada en la subfase de Reconstrucción. También pueden realizarse análisis estadísticos de los resultados, así como comparaciones entre las reconstrucciones y el phantom.

Implementación de Algoritmos definidos por el Usuario:

SNARK93 provee al usuario la opción de definir sus propios algoritmos de reconstrucción y tests de terminación. El usuario es provisto con dos funciones (ALG1 y ALG2) en las que puede implementar el paso iterativo de su(s) algoritmo(s). Tales funciones son ejecutadas por SNARK93 una vez por cada iteración, y en su llamado se pasan los parámetros correspondientes al número de iteración actual y al array conteniendo la imagen reconstruida por el paso iterativo anterior. Asimismo, son provistas dos funciones (TRM1, TRM2) en las que el usuario puede especificar las condiciones de terminación de su(s) algoritmo(s). Para uso de estos algoritmos están disponibles funciones para obtener los coeficientes de las funciones base (pixels o blobs, según corresponda) correspondientes a cada rayo (i.e.: filas de la matriz del sistema), así como una función para obtener el valor de la suma real de cada rayo.

Algoritmos de Reconstrucción incluidos en SNARK93:

SNARK93 incluye los siguientes algoritmos de reconstrucción: Backprojection (ver [GOR/74]), Convolution (ver [GOR/74]), Rho Filtered Layergram (ver [SMI/73]), Fourier (ver [GOR/74]), Convolution para proyecciones de haz divergente (ver [HER/76b]), Additive Algebraic Reconstruction Techniques (ver [HER/76a]), Multiplicative Algebraic Reconstruction Techniques (ver [HER/76a]), Quadratic Optimization Techniques (ver [HER/76c]), Simultaneous Iterative Reconstruction Technique (ver [GIL/72]), Reconstrucción por Expansión en Series de Perry, Altschuler y Altschuler (ver [PER/75]), Método de Expansión en Series de Marr, Smith, Peters y Bates (ver [MAR/74]), Maximum *a posteriori* Probability

Algorithm based in Expectation Maximization (ver [SHE/82]), Linogram Method (ver [EDH/87].) Para una descripción detallada de estos algoritmos ver [BRO/93].

3.3 Valores obtenidos para Análisis:

En la fase de análisis, el comando EVALUATE de SNARK93 produce la evaluación de ciertas mediciones cuantitativas de las diferencias entre el phantom y las reconstrucciones. Tales comparaciones pueden realizarse sobre toda la gama de densidades de la imagen, o bien sobre una ventana de densidad definida por el usuario. Puede compararse tanto la totalidad de pixels de la imagen, como también solo regiones elegidas.

Se describen a continuación las mediciones provistas por este comando. Sea $\rho^q_{i,j}$ la densidad asignada al pixel en la fila i-ésima y columna j-ésima de la reconstrucción luego de q iteraciones si $q \geq 1$, y la densidad en el pixel correspondiente en el phantom si $q = 0$. Sea S el conjunto de pares (i, j) tal que el pixel en la i-ésima fila y j-ésima columna esta en la región de interés, esto es, esta dentro de la región seleccionada de la imagen y su densidad pertenece a la ventana de densidad definida.

Área:

$$\alpha = \sum_{(i,j)\in S} 1$$

Densidad Media:

$$\bar{p}^q = \frac{1}{\alpha} \sum_{(i,j)\in S} p^q_{i,j}$$

Varianza:

$$v^q = \frac{1}{\alpha} \sum_{(i,j)\in S} \left(p^q_{i,j} - \bar{p}^q \right)^2$$

Desviación Estándar:

$$\sigma^q = \sqrt{v^q}$$

Distancia:

$$\delta^q = \begin{cases} \dfrac{1}{\sigma^0} \sqrt{\dfrac{1}{\alpha} \sum_{(i,j)\in S} \left(p^q_{i,j} - p^0_{i,j} \right)^2}, & si \quad \sigma^0 > ZERO \\ \sqrt{\sum_{(i,j)\in S} \left(p^q_{i,j} - p^0_{i,j} \right)^2}, & si \quad \sigma^0 \leq ZERO \end{cases}$$

(donde ZERO es un número de punto flotante muy pequeño, pero que puede ser distinguido del número cero por la computadora)

Error Relativo:

$$m = \sum_{(i,j)\in S} \left| p^0_{i,j} \right|$$

$$R^q = \begin{cases} \left(\sum_{(i,j)\in S} \left| p^q_{i,j} - p^0_{i,j} \right| \right) \Big/ m, & si \quad m > ZERO \\ \sum_{(i,j)\in S} \left| p^q_{i,j} - p^0_{i,j} \right|, & si \quad m \le ZERO \end{cases}$$

Residuo:

Sea $R_{p,r}$ la estimación de la atenuación real del r-ésimo rayo de la p-ésima proyección (es decir, $R_{p,r}$ es el valor de la caída de energía del rayo que se produce entre la fuente y el detector luego de que este cruza el objeto bajo análisis). Sean $R^0_{p,r}$ y $R^q_{p,r}$ las pseudo atenuaciones del r-ésimo rayo y p-ésima proyección del phantom y de la imagen reconstruida en la iteración q, respectivamente. En notación matricial, si estamos resolviendo el sistema A . x = b, donde A es la matriz de proyecciones, x es la imagen reconstruida y b los valores de atenuación real de cada rayo; y llamamos i = (p, r), resultan $R_{p,r} = b_i$ y $R^q_{p,r} = A_i$. x, para x la imagen reconstruida luego de q iteraciones.

El valor del residuo entonces resulta:

$$r^q = \sqrt{\sum_{p=1}^{PRJNUM} \sum_{r=FUSRAY}^{LUSRAY} \left(R^q_{p,r} - R_{p,r} \right)^2}$$

Resolución Puntual:

Para cada imagen digitalizada *p* de $n \times n$, usamos *p(k)* para denotar a la imagen digitalizada de $\lfloor n/2^k \rfloor \times \lfloor n/2^k \rfloor$, obtenida a partir de *p* luego de promediar el valor de $2^k \times 2^k$ pixels de *p* en un pixel de *p(k)*. Si 2^k no divide a n, las columnas de más a la derecha y las filas de abajo de *p* son descartadas en el cálculo de *p(k)*. Para dos imágenes digitalizadas *p* y *p'* definimos:

$$E(p, p') = \max |p_{i,j} - p'_{i,j}|, \quad 1 \le i,j \le n$$

donde $p_{i,j}$ y $p'_{i,j}$ denotan los valores asignados al pixel en la i-ésima fila y j-ésima columna de *p* y *p'*, respectivamente. Usando estos conceptos definimos, para cada entero k, tal que $1 \le 2^k \le NELEM$,

$$E(2^k)^q = E(p^q(k), p^0(k)).$$

3.4.1 Valores de Mérito:

En el área de imágenes medicas, una enorme variedad de algoritmos han sido propuestos para reconstruir el interior del cuerpo humano. Por lo tanto es deseable evaluar la eficacia relativa de dos o más métodos de reconstrucción para una tarea específica, en una manera que sea estadísticamente consistente.. Tal evaluación, por tanto, debe ser hecha usando un conjunto muestral lo suficientemente grande como para proveernos con un resultado estadísticamente significativo.

Realizar esta evaluación en phantoms matemáticos requiere un medio de correr los algoritmos a comparar sobre datos de proyecciones obtenidos de un número grande de phantoms generados al azar. Luego, varias mediciones numéricas de comparación entre las imágenes reconstruidas y los phantoms originales pueden ser usadas para llegar a una conclusión que tenga algún valor estadístico. Una manera en que esto puede lograrse en el sistema de programación SNARK93, es por medio de un programa *driver* que alimente a

SNARK93 de los comandos de entrada necesarios para generar tantos phantoms como sea necesario (además de los datos de las proyecciones), ejecutar los algoritmos deseados sobre estos datos y evaluar las imágenes así reconstruidas. Tal programa driver es provisto en la herramienta SuperSNARK. El método usado en la evaluación comparativa de los algoritmos consiste de los siguientes procedimientos ([FUR/93]):

a) generación de muestras al azar, de un conjunto de phantoms descrito estadísticamente y sus datos de proyecciones;

b) reconstrucción a partir de los datos de proyecciones para cada algoritmo a comparar;

c) asignación de un Valor de Mérito a cada imagen reconstruida. El valor de mérito debe ser una medida de la calidad de la imagen para resolver la tarea especificada;

d) calculo del nivel de significancia estadística, basada en el valor de mérito de las reconstrucciones, bajo el cual puede rechazarse la hipótesis de que los métodos son igualmente útiles para realizar la tarea.

Existen tres Valores de Mérito implementados en SNARK, además de la posibilidad de que el usuario implemente el suyo propio. Los Valores de Mérito implementados son:

Exactitud Estructural ([HER/91]):

Consideremos un phantom que contiene un total de N estructuras. Sea A^q_k el valor promedio de los pixels cuyos centros están en la estructura k en la imagen resultante de la reconstrucción luego de q iteraciones. Y sea A^0_k el valor promedio de los pixels correspondiendo a esa estructura en el phantom.

Definimos el valor de *exactitud estructural* de una reconstrucción en la iteración q como:

$$-\frac{1}{N}\sum_{k=1}^{N}\left|A_k^q - A_k^0\right|$$

Exactitud Puntual ([HER/91]):

El valor de *exactitud puntual* esta definido como el negativo de la raíz de la distancia cuadrática media normalizada, δ^q (definido en 3.3), entre una reconstrucción y el phantom.

Proporción de Aciertos ([HER/91]):

Este valor de mérito es solo aplicado en aquellos phantoms que contienen pares simétricos de estructuras (simétricos respecto al eje-y y con densidades distintas en cada una.) Para cada par ocurre un *acierto* si el índice de anormalidad en la reconstrucción es más alto para la estructura del par para la cual el índice de anormalidad es más alto en el phantom. La *proporción de aciertos* de una reconstrucción es el número de aciertos dividido el número total de pares.

3.4.2 Calculo de Nivel de Significancia Estadística

El paso final en el proceso de evaluación es el cálculo de significancia estadística. Los Valores de Mérito definidos anteriormente indican cuan buena es una reconstrucción para una cierta tarea. Un algoritmo con un mayor valor de mérito es considerado mas apropiado para la tarea dada que uno con un valor de mérito menor. ([HER/89a]) Si concluimos, basados en un valor de mérito, que un método es más apropiado que otro para una cierta tarea, aún necesitamos determinar si nuestra conclusión es estadísticamente significativa. Para determinar esto, asignamos un nivel de significancia estadística al rechazo de la hipótesis-nula de que dos métodos de reconstrucción son igualmente buenos (desde el punto de vista del valor de mérito usado), en favor de la hipótesis que el método con mayor valor de mérito es mejor, del siguiente modo:

Sean φ^q_m y ϕ^r_m los valores de mérito de las reconstrucciones del m-ésimo (de un total de M) conjunto de datos en las iteraciones q y r, reconstruidos mediante los dos métodos, respectivamente. Entonces, de acuerdo a la hipótesis-nula, $\varphi^q_m - \phi^r_m$ son una muestra de una variable aleatoria con media cero. Por lo tanto, para M suficientemente grande ($M \geq 30$), resulta que:

$$\sum_{m=1}^{M}\left(\varphi_m^q - \phi_m^r\right)$$

es una muestra aleatoria normal con media cero y varianza ([HER/91])

$$\sum_{m=1}^{M}\left(\varphi_m^q - \phi_m^r\right)^2$$

Podemos entonces usar la distribución normal para calcular la significancia estadística ([MOU/89]).

3.5 Visualización

Dos herramientas se incluyen para la visualización de las salidas de SNARK93. Un programa llamado SnarkEval es provisto con SNARK93, para graficar en pantalla las curvas de los valores obtenidos para análisis mediante el comando EVALUATE (ver 3.3). (Las Figuras 5.1 a 5.24 fueron generadas con esta herramienta.)

También se provee un programa llamado SnarkDisplay, el cual permite mostrar imágenes por pantalla representando los datos de proyecciones así como los phantoms e imágenes reconstruidas por SNARK93 en cada iteración. (Ver Figuras 5.25 a 5.48 para ejemplos de gráficos producidos con SnarkDisplay.) Esta herramienta permite seleccionar un factor de escala para cambiar el tamaño de la imagen en pantalla. Asimismo es posible seleccionar una ventana de densidad de modo de solo visualizar las secciones de la imagen pertenecientes a tal ventana.

4. Extensiones realizadas a SNARK93:

En el presente trabajo se realizó la portación de SNARK93 y sus herramientas al ambiente Windows, ya que su versión original fue desarrollada para Sun, y las herramientas de visualización para el ambiente Suntools. También se realizó la extensión de este sistema para permitir la reconstrucción, evaluación y visualización de imágenes considerando su representación mediante blobs. Se implementaron los algoritmos CAV, ART y ACCAV2 y se los agregó a SNARK93.

4.1 Reconstrucción de imágenes usando blobs

Para permitir realizar reconstrucciones de imágenes usando blobs para su representación se agregó el comando BLOBS a SNARK93. Por medio de este comando pueden especificarse los parámetros a y α de los blobs, así como la distancia de muestreo de la grilla hexagonal. En [MAT/96b], [GAR/01] se trata el tema de la buena elección de parámetros para los blobs según el tipo de problema a resolver. Como se sugiere en [MAT/96b], hemos usado m=2 de modo que los blobs resulten dos veces diferenciables.

Asimismo debió extenderse el formato del archivo RECFIL que almacena las distintas imágenes reconstruidas en cada iteración, de modo que pudiera contener tanto reconstrucciones de imágenes representadas con pixels como aquellas resultantes del uso de blobs.

También debió modificarse el manejo interno del array RECON conteniendo la reconstrucción de la imagen para que en el caso de BLOBS se lo interpretase adecuadamente.

El corazón de los cambios a SNARK93 para soporte de blobs en la fase de reconstrucción está en la función BWRAY, la cual es la versión para blobs de la función WRAY. El propósito de WRAY es obtener, dadas una proyección y un rayo de esa proyección, la lista de los índices de los pixels intersecados por ese rayo, y las integrales de línea de la intersección entre el rayo y cada uno de esos pixels. Matemáticamente hablando, WRAY retorna la fila de la matriz de proyecciones A correspondiente al rayo.

BWRAY, análogamente, retorna los índices de los blobs intersecados por cada rayo así como las integrales de línea de la función de los blobs sobre el rayo. Para esto se obtiene primero la ecuación del rayo y se calculan, para cada fila de la grilla, los blobs intersecados por el rayo en esa fila. Tales blobs son aquellos cuyos centros distan de la intersección del rayo con la fila en no más de a / sen(θ), donde a es el radio del blob y θ es el ángulo que el rayo forma con el eje-x. Si la distancia entre el centro de un blob de una determinada fila y la intersección del rayo con esa fila es mayor que ese valor, entonces la distancia del centro del blob al rayo es mayor de a, y entonces la intersección entre el blob y el rayo es vacía. (Ver Figura 4.1)

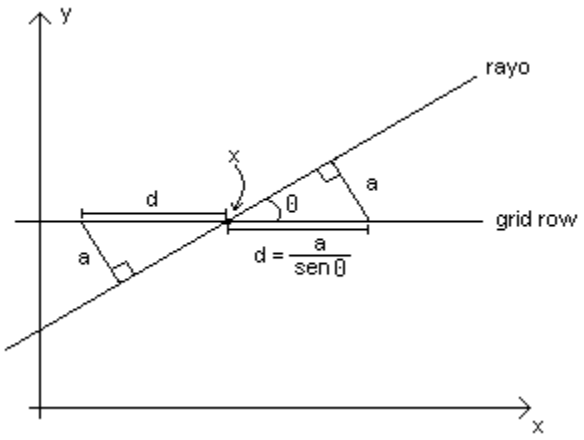

Figura 4.1

Por cada blob intersecado por el rayo se calcula la integral de la función del blob sobre el rayo. Tal integral depende únicamente de la distancia del rayo al centro del blob. La distancia d del rayo, que pasa por el punto (rayX, rayY) y forma un ángulo θ con al eje x, al centro del blob (blobX, blobY) se calcula del siguiente modo: (Ver Figura 4.2)

$$d = \| (pX, pY) \|,$$

donde:

$$(pX, pY) = (blobX, blobY) - (x_0, y_0),$$

$$(x_0, y_0) = \alpha \, . \, (\cos \theta, \operatorname{sen} \theta) + (rayX, rayY),$$

$$\alpha = \langle (blobX, blobY) - (rayX, rayY), (\cos \theta, \operatorname{sen} \theta) \rangle$$

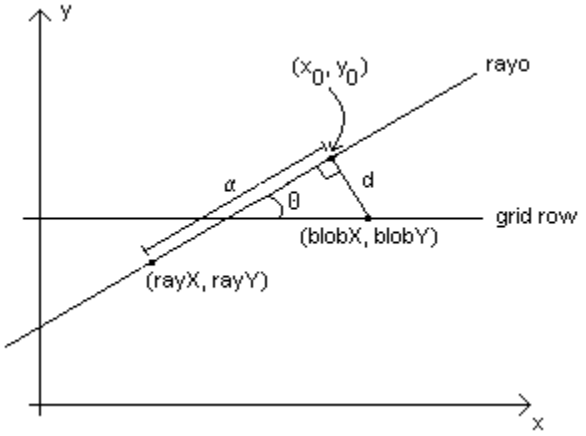

Figura 4.2

La integral de la función del blob sobre la línea del rayo, en función de la distancia s del rayo al centro del blob, tiene la siguiente forma (usando m=2) [LEW/90]:

$$p(s) = a / I_2(\alpha) \times (2\pi/\alpha)^{1/2} \times (1 - (s/a)^2)^{5/4} \times I_{5/2}(\alpha\,(1 - (s/a)^2)^{1/2}), \qquad (4.1)$$

La definición de $I_2(x)$ es la siguiente [WAT/44]:

$$I_2(x) = \left(\frac{x}{2}\right)^2 \cdot \sum_{k=0}^{\infty} \frac{(x/2)^{2k}}{k!\Gamma(2+k+1)} \qquad (4.2)$$

El valor de esta función se aproxima sumando unos pocos términos de la serie.

$I_{5/2}(x)$ esta definido por [WAT/44]:

$$I_{5/2}(x) = (2/(\pi x))^{1/2}\ \{(3/x^2 + 1)\ \text{senh}\ x - 3/x\ \cosh x\} \qquad (4.3)$$

Los valores de p(s) se precalculan para $s \in \{x / x = r.a / 1000,\ 0 \le r < 1000\}$ y se almacenan en una tabla. Los valores de p(x) para los x que no tienen entradas en la tabla se hallan por interpolación.

Existen dos arrays prealocados en los que, dado un rayo, la función WRAY almacena los índices de los blobs interceptados y los valores de los coeficientes correspondientes a cada uno de esos blobs (integrales de la función del blob sobre la línea del rayo). Para prealocar tales arrays es necesario conocer una cota máxima razonable de la cantidad de blobs interceptados por un rayo. Una cota burda es el total de blobs, pero este número es en general bastante grande y supera grandemente la cantidad máxima de blobs que un rayo puede interceptar, con lo cual habría mucho desperdicio de memoria si se usara este valor como tamaño de los arrays. La cota máxima exacta puede calcularse como:

$$\max_{0 \le \theta \le \frac{\pi}{2}} bI(\theta)$$

donde la función $bI(\theta)$ retorna la cantidad de *blobs interceptados* por un rayo que pasa por el punto central de la grilla de blobs y forma un ángulo θ con el eje x, y está definida por:

$$bI(\theta) = fI(\theta) \cdot bA(\theta)$$

La función $fI(\theta)$ es la cantidad de *filas* (de la grilla) *interceptadas* por tal rayo, y la función $bA(\theta)$ da la cantidad de *blobs alcanzados* (intersecados) por el rayo en cada fila.
$fI(\theta)$ se define del siguiente modo:

$$fI(\theta) = min(rC, \text{tg}\,\theta \cdot rC)$$

donde rC es la *cantidad de filas* de la grilla. Es fácil ver que $fI(\theta)$ puede escribirse como:

$$fI(\theta) = \begin{cases} rC, & si \quad \frac{\pi}{4} \le \theta \le \frac{\pi}{2} \\ \text{tg}\,\theta \cdot rC, & si \quad 0 \le \theta \le \frac{\pi}{4} \end{cases}$$

bA(θ) esta definida así:

$$bA(\theta) = min\left(\frac{2 \cdot \frac{a}{\operatorname{sen}\theta}}{sD}, \frac{N}{sD} \right)$$

donde: a es el radio de los blobs, N es el ancho (alto) de la región cubierta por la grilla, y sD es la *sampling distance* (ie: distancia entre centros de blobs contiguos en una fila de la grilla.)
La lógica detrás de esta definición está en que N/sD es la cantidad total de blobs en una fila, mientras que 2 . a / (sen θ . sD) es la cantidad de blobs alcanzados por el rayo en una fila, asumiendo que la fila contiene infinitos blobs cuyos centros están separados por sD.

bA(θ) puede entonces escribirse como:

$$bA(\theta) = \frac{1}{sD} \cdot min\left(\frac{2 \cdot a}{\operatorname{sen}\theta}, N \right)$$

Ahora bien:

$$\frac{2 \cdot a}{\operatorname{sen}\theta} \leq N \Leftrightarrow \frac{2 \cdot a}{N} \leq \operatorname{sen}\theta \xleftarrow[\substack{\text{sen es estrictamente} \\ \text{creciente en } \left[0, \frac{\pi}{2}\right]}]{} \operatorname{arcsen}\left(\frac{2 \cdot a}{N} \right) \leq \theta$$

Teniendo en cuenta lo anterior podemos entonces rescribir bI(θ) como:

$$bI(\theta) = \begin{cases} \dfrac{rC \cdot 2 \cdot a}{sD \cdot \operatorname{sen}\theta}, & si \quad \dfrac{\pi}{4} \leq \theta \leq \dfrac{\pi}{2} \qquad (*) \\ \dfrac{\operatorname{tg}\theta \cdot rC \cdot 2 \cdot a}{sD \cdot \operatorname{sen}\theta}, & si \quad \operatorname{arcsen}\left(\dfrac{2 \cdot a}{N} \right) \leq \theta < \dfrac{\pi}{4} \qquad (**) \\ \dfrac{\operatorname{tg}\theta \cdot rC \cdot N}{sD}, & si \quad 0 \leq \theta < \operatorname{arcsen}\left(\dfrac{2 \cdot a}{N} \right) \qquad (***) \end{cases}$$

(*) es una función estrictamente decreciente en [π/4, π/2], luego alcanza su máximo valor en θ = π/4 y tal máximo entonces resulta:

$$\frac{rC \cdot 2 \cdot a}{sD \cdot \operatorname{sen}\frac{\pi}{4}} = \frac{rC \cdot 2 \cdot a}{sD \cdot \frac{1}{\sqrt{2}}} = \frac{rC \cdot 2 \cdot \sqrt{2} \cdot a}{sD} \qquad (\bullet)$$

(**) es una función estrictamente creciente en [arcsen(2.a/N), π/4], luego alcanza su máximo valor en θ = π/4 y tal máximo entonces resulta:

$$\frac{rC \cdot 2 \cdot a}{sD \cdot \cos\frac{\pi}{4}} = \frac{rC \cdot 2 \cdot a}{sD \cdot \frac{1}{\sqrt{2}}} = \frac{rC \cdot 2 \cdot \sqrt{2} \cdot a}{sD} \qquad (\bullet\bullet)$$

(***) es estrictamente creciente en [0, arcsen(2.a/N)], luego alcanza su máximo valor en $\theta = \text{arcsen}(2.a/N) \Leftrightarrow \text{sen}\,\theta = 2.a/N \Rightarrow \cos\theta = \sqrt{(1 - 4.a^2/N^2)} \Rightarrow \text{tg}\,\theta = 2.a/(N.\sqrt{(1 - 4.a^2/N^2)})$, luego su máximo resulta:

$$\frac{\frac{2\cdot a}{N}}{\sqrt{1-\frac{4\cdot a^2}{N^2}}}\cdot\frac{rC\cdot N}{sD}=\frac{2\cdot a\cdot rC\cdot N}{\sqrt{N^2-4\cdot a^2}\cdot sD}\qquad(\bullet\bullet\bullet)$$

Finalmente el máximo de $bI(\theta)$ para θ en $[0, \pi/2]$ es el máximo de (•), (••) y (•••), es decir:

$$\max_{0\le\theta\le\frac{\pi}{2}} bI(\theta) = max\left(\frac{rC\cdot 2\cdot\sqrt{2}\cdot a}{sD},\frac{2\cdot a\cdot rC\cdot N}{\sqrt{N^2-4\cdot a^2}\cdot sD}\right)=\frac{2\cdot a\cdot rC}{sD}\cdot max\left(\sqrt{2},\frac{N}{\sqrt{N^2-4\cdot a^2}}\right)$$

4.2 Evaluación de performance usando blobs

Para poder realizar comparaciones entre una reconstrucción realizada usando blobs y el correspondiente phantom, dado que este ultimo está especificado en pixels, es necesaria una traducción de, o bien el phantom a blobs, o bien de la reconstrucción a pixels. Lo primero requiere aproximar el phantom por una combinación lineal de blobs y para esto debería resolverse (en el sentido de mínimos cuadrados) un sistema de ecuaciones para poder así obtener los coeficientes de los blobs. La segunda opción ("pixelizacion") es muchísima menos problemática y es la que es usada por los investigadores en la disciplina de reconstrucción de imágenes. Para poder realizar esta conversión, lo único que debe hacerse es, para cada pixel, sumar las contribuciones de los blobs en el punto central del pixel, multiplicados estos por el correspondiente coeficiente. (Se calcula la contribución de cada blob en el punto central del pixel en lugar de la integral de la función del blob sobre la región del pixel, pues aquella es una buena aproximación de esta y resulta muy rápida de calcular.) Es decir, sea p_{ij} la densidad que tendrá el pixel de la fila i-ésima, columna j-ésima luego de la conversión, sea B la cantidad de blobs, sea c_k el coeficiente correspondiente al k-ésimo blob, sea b(r) la función de los blobs en un punto a distancia r del centro de este, sea x_k el punto de coordenadas del centro del blob k-ésimo, sea r_{ij} el punto central del pixel de la fila i-ésima, columna j-ésima, entonces resulta:

$$p_{ij}=\sum_{k=1}^{B} c_k\cdot b\left(\left\|x_k-r_{i,j}\right\|\right)$$

En SNARK93 la función blob2pixel realiza esta tarea, y la función EVALUATE fue modificada para que en el caso de uso de blobs llame a blob2pixel antes de realizar los cálculos de los valores para análisis descritos más arriba sobre la reconstrucción.

4.3 Visualización de reconstrucciones usando blobs

Para poder presentar en forma visual una reconstrucción obtenida usando blobs debe seguirse un procedimiento similar al de la pixelizacion descripta mas arriba con fines de comparación. La única diferencia con tal proceso radica en el hecho que para la visualización puede elegirse un factor de escala que indicará cuantos pixels de la pantalla gráfica se usarán para representar un pixel de la imagen. Así en el proceso de pixelizacion se tiene en cuenta tal factor de escala y se calcula la suma de las contribuciones de cada blob para cada uno de los pixels de la pantalla gráfica (que solo coincidirán con los del phantom si el factor de escala es 1).

5. Resultados Experimentales

Se utilizó SNARK93 para comparar la performance del nuevo algoritmo ACCAV2 [SCO/01] con CAV [CEN/00] y con el algoritmo *row-action* ART (ver por ejemplo: [CEN/97], algoritmo 10.4.1.) en un problema de reconstrucción de imágenes por medio de proyecciones. Para el caso de ART se define una *iteración* como una pasada completa por todas las ecuaciones del sistema. Todos los experimentos se iniciaron con $x^0 = 0$. En la implementación de CAV, los valores de $\sum_{l=1}^{n} s_l \left(a_{il}\right)^2$, i=1, ..., n, se calcularon en la primera iteración y se almacenaron para las iteraciones siguientes. ACCAV2 se implementó en la forma eficiente propuesta por H. Scolnik et al.

Se examinaron ocho casos de test, (siguiendo el patrón propuesto en [CEN/00]), usando el "Herman Head Phantom" ([HER/80], sección 4.3). Los ocho casos usaron blobs y pixels, así como resoluciones de imagen distintas y distintas cantidades de proyecciones y rayos por proyección, lo cual resulta en distintas cantidades de variables y ecuaciones. (El número de ecuaciones es el producto entre el número de proyecciones y la cantidad de rayos por proyección, mientras que la cantidad de variables es el número de pixels o blobs – según corresponda – de la imagen.) Ver Tabla 5.1.

Caso	Representación de la Imagen	Ecuaciones	Variables	Tamaño Imagen	Proyecciones	Rayos por proyección
1.1	**Pixels**	13.137	13.225	115×115	151	87
1.2		26.425	13.225	115×115	151	175
1.3		126.655	119.025	345×345	365	347
1.4		232.275	119.025	345×345	475	489
2.1	**Blobs**	13.137	11.600	115×115	151	87
2.2		26.425	11.600	115×115	151	175
2.3		126.655	103.454	345×345	365	347
2.4		232.275	103.454	345×345	475	489

Tabla 5.1. Casos de test.

ART y CAV se corrieron con dos parámetros de relajación diferentes (los cuales permanecieron fijos para todas las iteraciones): ART – 0.1, 1.0; CAV – 1.0, 2.0.

Para los casos de uso de blobs, se utilizaron blobs con parámetros a=2, α=10.4 ("*standard blob"* [MAT/96b]) y la grilla hexagonal con *sampling distance* = 2.

Para las comparaciones entre algoritmos se usaron las medidas de *distancia*, *error relativo* y *desviación estándar* definidas en 3.3. Las curvas de convergencia de estas medidas para los ocho casos de test se muestran en las figuras 5.1 a 5.24.

Debe notarse aquí que si bien los tres algoritmos van reduciendo la norma del residuo ($\|b - Ax\|$) en cada iteración, esto no necesariamente significa que las sucesivas imágenes obtenidas se vayan acercando indefinidamente al phantom original. Lo que sucede en la práctica es que hasta cierto valor de iteración la *distancia* entre la imagen reconstruida y el phantom original va disminuyendo, pero a partir de un cierto punto, este valor empieza a aumentar. La razón de este fenómeno está en el hecho de que el problema original fue discretizado, y además el sistema de ecuaciones que está siendo resuelto es muy probablemente inconsistente, debido a la imprecisión y ruido en las mediciones y el error en la representación de los datos, por lo cual se está convergiendo a una "solución" de un problema que no necesariamente es el problema original, sino más bien similar.

Comparaciones de Métricas

Al observar las curvas de distancia y error relativo (Figuras 5.1 a 5.24) se puede apreciar que generalmente ACCAV2 converge más rápido que CAV y ART, sin embargo las reconstrucciones logradas por ACCAV2 empiezan a deteriorarse después de cierto número de iteraciones, mientras que CAV *aparentemente* continúa mejorando indefinidamente a medida que el número de iteraciones crece. Debe enfatizarse el hecho que ACCAV2 logra en muy pocas (alrededor de 13) iteraciones, antes de empezar a

deteriorarse, muy buenos resultados, mientras que a CAV llegar a tales resultados le lleva mucho más trabajo (alrededor de 55).

Si bien al observar los gráficos de las curvas da la impresión que CAV siempre va en mejora como se afirma en [CEN/00], esto **no es verdad**, sino que la imagen reconstruida por CAV también a partir de un cierto número de iteración empieza a deteriorarse. Lo que sucede es que al ser CAV mas lento en converger que ACCAV2, el deterioro también se presenta en un número de iteración avanzado. Se ejecutaron 300 iteraciones de ambos métodos para los casos de test 1.1, 1.2, 2.1 y 2.2 y se observó que el deterioro en CAV comenzaba alrededor de la iteración 60 (Ver Figuras 5.49 a 5.56). Es interesante observar que si bien la menor distancia / error lograda por CAV fue menor que la lograda por ACCAV2, la diferencia entre tales valores mínimos es despreciable (Ver Tabla 5.6), por lo que puede afirmarse que ACCAV2 logra un resultado tan bueno como el de CAV pero en un número mucho menor de iteraciones.

Una ventaja muy importante de ACCAV2 sobre CAV es el hecho de que ACCAV2 no requiere del uso de parámetros de relajación y por lo tanto siempre está asegurada una rápida convergencia del algoritmo. Este no es el caso de CAV, ya que para distintos problemas, se pueden necesitar valores de λ_k diferentes para obtener buenos resultados y esto implica la eventual ejecución de CAV con distintos valores de tal parámetro para hallar el mejor resultado.

Comparaciones visuales:

En las figuras 5.25 a 5.48 correspondientes a los 8 casos de test se muestran los phantoms y las imágenes reconstruidas por ART, CAV y ACCAV2, luego de 10, 20, 30, 40 y 50 iteraciones. Para los casos de CAV y ART se muestra sólo la imagen correspondiente al valor del parámetro de relajación que dio el mejor resultado.

En general puede apreciarse que ACCAV2 produce imágenes más nítidas que CAV en un número pequeño de iteraciones y que las imágenes logradas por ACCAV2 y CAV son superiores a las logradas por ART.

El deterioro de ACCAV2 que se manifiesta en las curvas de las medidas de distancia y error relativo luego de un cierto número de iteraciones resulta ser lo suficientemente pequeño, como para no apreciarse en las visualizaciones de las imágenes reconstruidas. Incluso en la mayoría de los casos las imágenes logradas en la iteración 50 de ACCAV2 resultan *visualmente* más cercanas al original que las logradas en la iteración 10, cuando la imagen es *matemáticamente* más cercana al phantom. Para un ejemplo de esto ver las figuras 5.30, iter. 50; 5.36, iter. 40; 5.39, iter. 40; 5.42, iter. 50; 5.45, iter. 50.

En los casos 1.1 y 2.1, los defectos artificiales (*artifacts*) circulares producidos por ART y CAV, son muchos mas pronunciados que los producidos por ACCAV2, para todas las iteraciones. En el caso 1.3, los *artifacts* en forma de X que aparecen en las reconstrucciones de CAV, casi no aparecen en las de ACCAV2.

Como era de esperarse, los tres algoritmos producen mejores imágenes cuando se usa un número mayor de ecuaciones para un mismo phantom. Es también observable el hecho que las reconstrucciones usando blobs resultaron mejores que las obtenidas mediante el uso de pixels. Para mencionar un ejemplo, baste notar que en los casos de menor resolución y ecuaciones (1.1 y 2.1), los *artifacts* circulares en las reconstrucciones obtenidas mediante el uso de blobs son menos pronunciados que los logrados usando pixels.

Tiempos de ejecución:

Los tiempos expuestos en la tabla 5.2 corresponden a la ejecución de los algoritmos, mediante SNARK93, en una PC laptop Compaq Evo N160 con un Pentium III de 1 Ghz y 256 Mb de RAM.

Algoritmo	**Tiempo (segs.) para 50 iteraciones**			
	Pixels		**Blobs**	
	Caso 1.1	**Caso 1.2**	**Caso 2.1**	**Caso 2.2**
ART	7.531	14.828	113.148	226.348
CAV	10.980	21.582	125.340	250.621
ACCAV2	26.148	51.359	260.352	520.090

Tabla 5.2. Tiempos de ejecución.

La relación de velocidad, (dónde por velocidad se entiende iteraciones por segundo), entre ACCAV2 y los otros dos algoritmos puede ser apreciada en la tabla 5.3 y fue obtenida a partir de los datos de la tabla 5.2.

	Pixels		**Blobs**	
	Caso 1.1	**Caso 1.2**	**Caso 2.1**	**Caso 2.2**
$\frac{t_{ACCAV2}}{t_{CAV}}$	2.381	2.379	2.077	2.075
$\frac{t_{ACCAV2}}{t_{ART}}$	3.472	3.463	2.300	2.297

Tabla 5.3. Relaciones de velocidad (iteraciones por segundo) entre los otros métodos y ACCAV2.

La cantidad de iteraciones necesarias para cada algoritmo para obtener la reconstrucción más cercana al phantom en los casos de test 1.1, 1.2, 2.1 y 2.2 se muestran en la tabla 5.4. Estos datos fueron obtenidos luego de correr 300 iteraciones de cada algoritmo. En las Figuras 5.49 a 5.56 se muestran las curvas correspondientes a estas corridas. En la tabla se muestra el número de iteración en el que ACCAV2 logró la reconstrucción a menor distancia del phantom (y tal valor de distancia entre paréntesis), así como el número de iteración en el que los otros métodos lograron una reconstrucción a ese mismo valor (o lo más cercano) de distancia del phantom. También se muestra el número de iteración y el menor valor de distancia logrado por los otros métodos si este fue menor que el obtenido por ACCAV2.

Algoritmo	**Cantidad de Iteraciones para lograr la menor distancia de ACCAV2.**			
	Pixels		**Blobs**	
	Caso 1.1	**Caso 1.2**	**Caso 2.1**	**Caso 2.2**
ART	47 (0.2876)	6 (0.0998), 8 (0.0966)	16 (0.2941), 20 (0.2926)	8 (0.2828), 12 (0.2791)
CAV	55 (0.2859), 121(0.2829)	56 (0.0997), 78 (0.0953)	53 (0.2945), 67 (0.2926)	53 (0.2840), 84 (0.2794)
ACCAV2	14 (0.2860)	13 (0.0998)	9 (0.2946)	9 (0.2840)

Tabla 5.4. Cantidad de iteraciones necesarias para lograr la reconstrucción más cercana al phantom.

Al mirar la tabla 5.3 se observa que para el caso de uso de pixels, ACCAV2 es 2.38 veces más lento que CAV y 3.4675 veces más lento que ART, mientras que usando blobs, ACCAV2 resulta 2.076 veces más lento que CAV y 2.2985 veces más lento que ART. Sin embargo la relación que interesa es aquella entre el tiempo que lleva a los algoritmos lograr la reconstrucción más cercana al phantom original. Tal relación tiene en cuenta la cantidad de iteraciones que lleva a cada algoritmo lograr ese resultado. Una estimación de tales valores puede obtenerse como:

$$\frac{t_{CAV} \cdot iter_{CAV}}{t_{ACCAV2} \cdot iter_{ACCAV2}} \text{ y } \frac{t_{ART} \cdot iter_{ART}}{t_{ACCAV2} \cdot iter_{ACCAV2}}$$

donde $iter_{ALG}$ es el número de iteración en el que el algoritmo *ALG* logra la reconstrucción más cercana al phantom. En la tabla 5.5 se muestran tales valores, calculados a partir de los datos de las tablas 5.3 y 5.4. Se observa entonces que ART resulta ser el algoritmo más veloz y CAV es el más lento de los tres métodos.

	Pixels		Blobs	
	Caso 1.1	**Caso 1.2**	**Caso 2.1**	**Caso 2.2**
$\frac{t_{CAV} \cdot iter_{CAV}}{t_{ACCAV2} \cdot iter_{ACCAV2}}$	1.649	1.810	2.835	2.838
$\frac{t_{ART} \cdot iter_{ART}}{t_{ACCAV2} \cdot iter_{ACCAV2}}$	0.966	0.133	0.772	0.386

Tabla 5.5. Relación de velocidad (de convergencia) entre ACCAV2 y los otros métodos.

% de diferencia entre la menor distancia lograda por ACCAV2 y la lograda por:	**Pixels**		**Blobs**	
	Caso 1.1	**Caso 1.2**	**Caso 2.1**	**Caso 2.2**
CAV	1.095 %	4.721 %	0.683 %	1.646 %
ART	N/A	3.312 %	0.683 %	1.755 %

Tabla 5.6. Diferencia porcentual entre el menor valor de distancia logrado por ACCAV2 y el obtenido mediante los otros métodos

Finalmente en la tabla 5.7 se expone las relaciones de tiempo entre el uso de pixels y blobs. Las reconstrucciones usando blobs emplearon para cada iteración entre 10 y 15 veces el tiempo de una iteración para el mismo algoritmo y caso cuando se usaron pixels.

Algoritmo	$\frac{t_{blobs}}{t_{pixels}}$	
	Caso x.1	**Caso x.2**
ART	15.024	15.264
CAV	11.415	11.612
ACCAV2	9.956	10.126

Tabla 5.7. Relaciones de tiempo entre reconstrucciones usando blobs y usando pixels.

El hecho que ART sea más veloz que ACCAV2 no debe ser una razón para descartar a este último, pues como se observó más arriba, la calidad de la reconstrucción lograda por este es superior a la de aquel. De igual modo, las reconstrucciones usando blobs al llevar más de 10 veces el tiempo que aquellas realizadas usando pixels no significa que estas deban ser preferidas por sobre aquellas.

Resultados estadísticamente significativos bajo condiciones reales:

Se utilizó SuperSnark para realizar corridas de sets de 50 experimentos de reconstrucción usando ACCAV2 y CAV sobre phantoms contaminados con ruido, reproduciendo así condiciones reales en el problema de reconstrucción de imágenes. Se realizaron dos sets de experimentos, uno usando pixels y otro usando blobs.

Los experimentos fueron realizados usando los phantoms: MINUS01.ATL, ..., MINUS.12.ATL, PLUS000.ATL, ..., PLUS013.ATL provistos en el ejemplo B.9 de SNARK93. La geometría del scanner usada fue divergente con 300 proyecciones de 151 rayos cada una. Se contaminaron las mediciones mediante ruido cuántico multiplicativo. Las comparaciones se realizaron entre las reconstrucciones obtenidas en la iteración 5 del algoritmo ACCAV2 y la obtenida en la iteración 14 de CAV 2 para el caso de blobs y en las iteraciones 16 y 7 para el caso de pixels, y dentro de la ventana de densidad [1.65, 2.05]. Se eligieron esos

números de iteración para realizar las comparaciones, pues para los phantoms usados en los experimentos, en general los respectivos algoritmos obtienen las reconstrucciones más cercanas al phantom original en esos valores de iteración. En las tablas 5.8 y 5.9 se muestran, para cada algoritmo, los promedios de los valores de mérito de los 50 experimentos en las iteraciones especificadas, así como el nivel de significancia estadística correspondiente.

Tipo de Valor de Mérito	Media del VM (CAV, iter. 16)	Media del VM (ACCAV2, iter. 7)	Nivel de Significancia
Exactitud Estructural	-0.24742	-0.09387	Menor a 0.000001
Exactitud Puntual	-0.72139	-0.55560	Menor a 0.000001
Proporción de Aciertos	0.80335	0.78748	0.00986

Figura 5.8. Valores de Significancia Estadística para los experimentos usando **pixels**.

Tipo de Valor de Mérito	Media del VM (CAV, iter. 14)	Media del VM (ACCAV2, iter. 5)	Nivel de Significancia
Exactitud Estructural	-0.62400	-0.58586	Menor a 0.000001
Exactitud Puntual	-10.05684	-10.05684	0.5
Proporción de Aciertos	0.54487	0.55659	0.17176

Figura 5.9. Valores de Significancia Estadística para los experimentos usando **blobs**.

En base a estos resultados puede afirmarse que hay evidencias suficientes para concluir que, desde el punto de vista de la exactitud estructural, el algoritmo ACCAV2 logra en 5 (7) iteraciones resultados superiores a los obtenidos por CAV 2 en 14 (16) iteraciones en el caso de uso de pixels (blobs). Además para el caso de exactitud puntual, cuando se usan pixels, puede concluirse que ACCAV2 supera en los resultados que este obtiene a aquellos logrados por CAV 2 en los números de iteración indicados. No hay además evidencias suficientes, desde el punto de vista de la proporción de aciertos para el caso de blobs y pixels, para afirmar que un algoritmo es mejor que el otro. Por otro lado, ambos algoritmos producen resultados similares desde el punto de vista de la exactitud puntual en el caso de uso de blobs, para los números de iteración indicados.

6. Conclusiones

De los resultados antes enunciados puede afirmarse que ACCAV2 es un algoritmo que supera a CAV en velocidad de convergencia y si bien es más lento en converger que ART, supera a este algoritmo en la calidad de las imágenes producidas. Por lo tanto ACCAV2 es un potencial candidato a emplearse en futuras implementaciones de soluciones al problema de reconstrucción de imágenes.

Siendo que ACCAV2 es en general más rápido para converger que CAV, aunque este último logra mejores aproximaciones a la imagen original en un número bastante mayor de iteraciones, se propone como posible trabajo de investigación la posibilidad de realizar reconstrucciones mediante ejecuciones híbridas de algoritmos, es decir por ejemplo, empezar ejecutando ACCAV2 unas pocas iteraciones, aprovechando su rapidísima convergencia, y con la imagen así obtenida continuar la reconstrucción mediante el uso de CAV y ver si de ese modo se logran, en pocas iteraciones, mejores reconstrucciones que las obtenidas mediante la ejecución de sólo ACCAV2 y más rápidas que las logradas ejecutando sólo CAV.

Agradecimientos

Deseo agradecer a Gabor Herman por su disposición a responder mis dudas y por sus invaluables sugerencias. Me siento además muy agradecido por la disposición que siempre ha tenido mi director Hugo Scolnik para atenderme personalmente así como responder mis e-mails de consulta, pese a lo muy ocupado

que siempre esta. No puedo dejar de mencionar a Nelly Echebest y Maria Teresa Guardarucci quienes también me fueron de ayuda con sus valiosos comentarios en nuestros intercambios de e-mails.

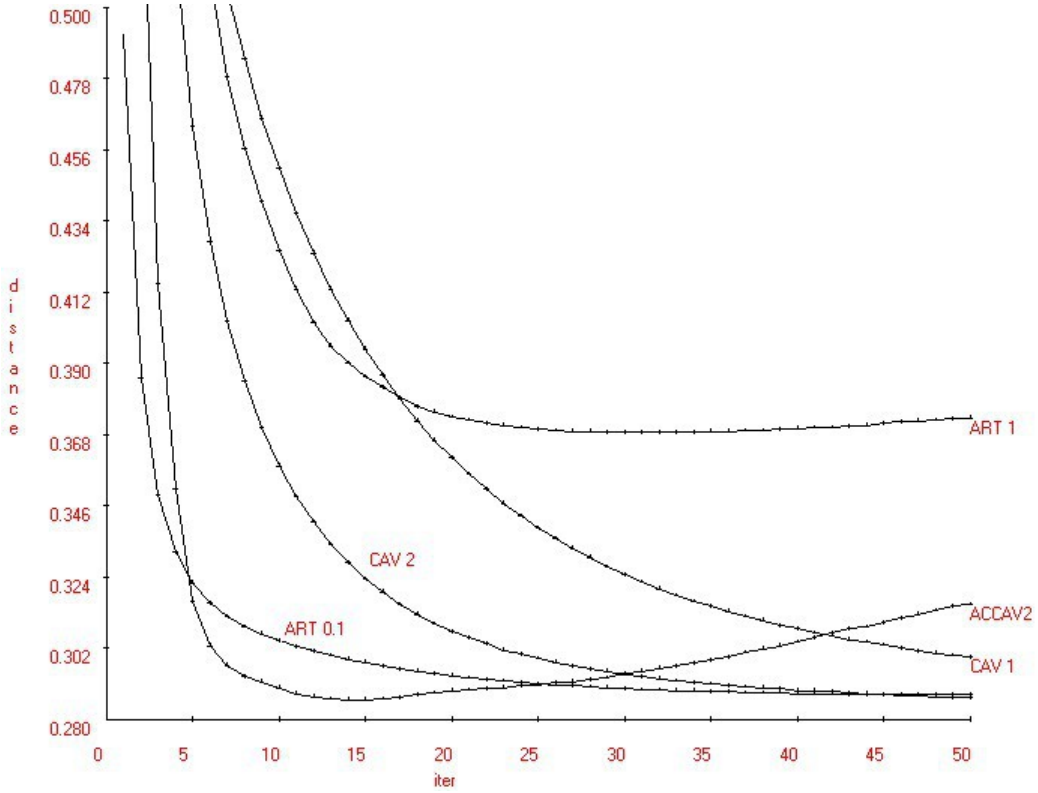

Figura 5.1. Distancia para caso 1.1 (115 × 115 pixels, 13.137 ecuaciones, 13.225 variables.)

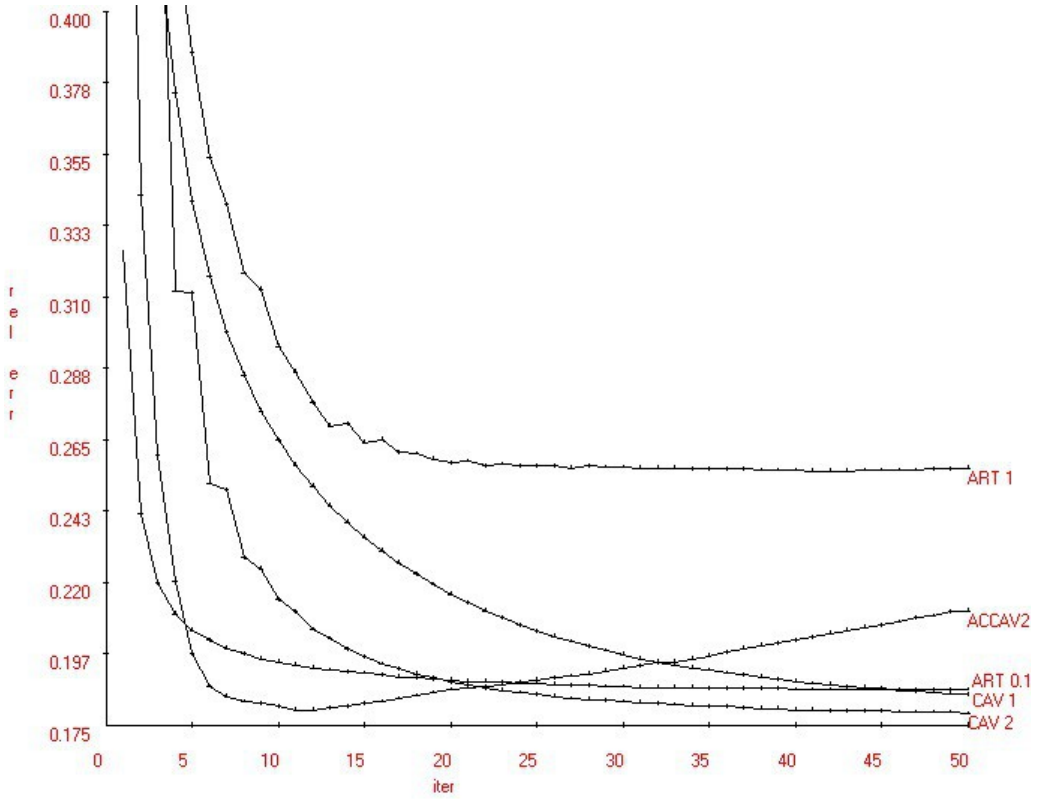

Figura 5.2. Error Relativo para caso 1.1 (115 × 115 pixels, 13.137 ecuaciones, 13.225 variables.)

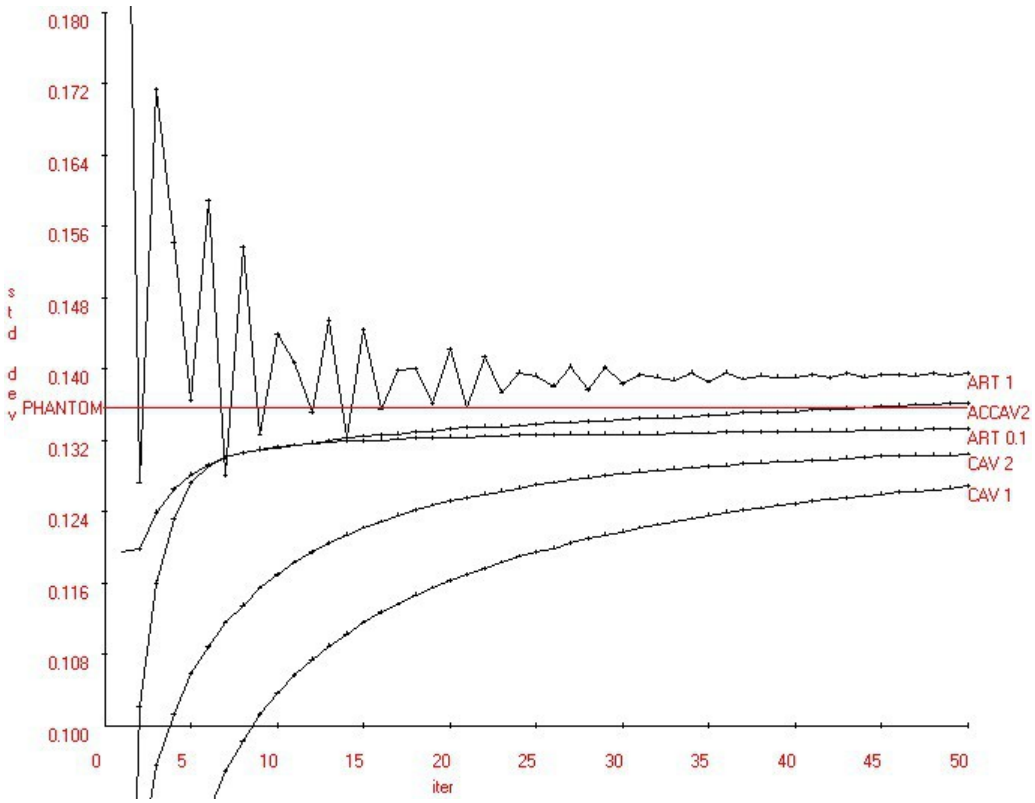

Figura 5.3. Desviación Estándar para caso 1.1 (115 × 115 pixels, 13.137 ecuaciones, 13.225 variables.)

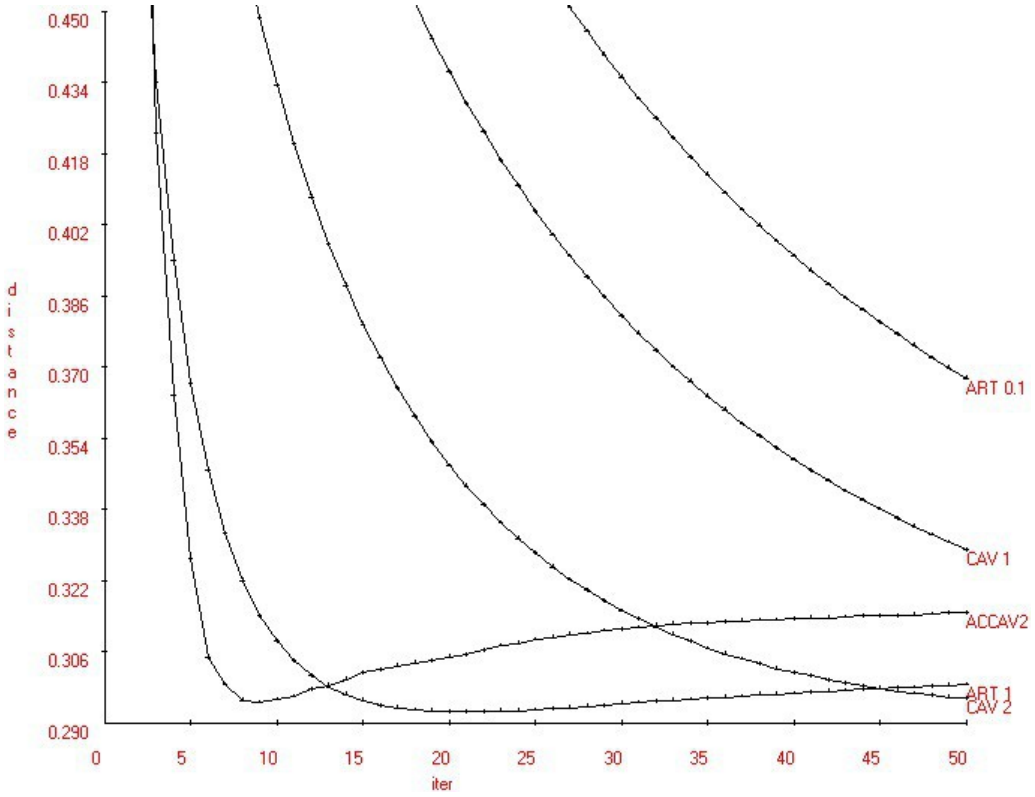

Figura 5.4. Distancia para caso 2.1 (115 × 115 pixels, 13.137 ecuaciones, 11.600 variables.)

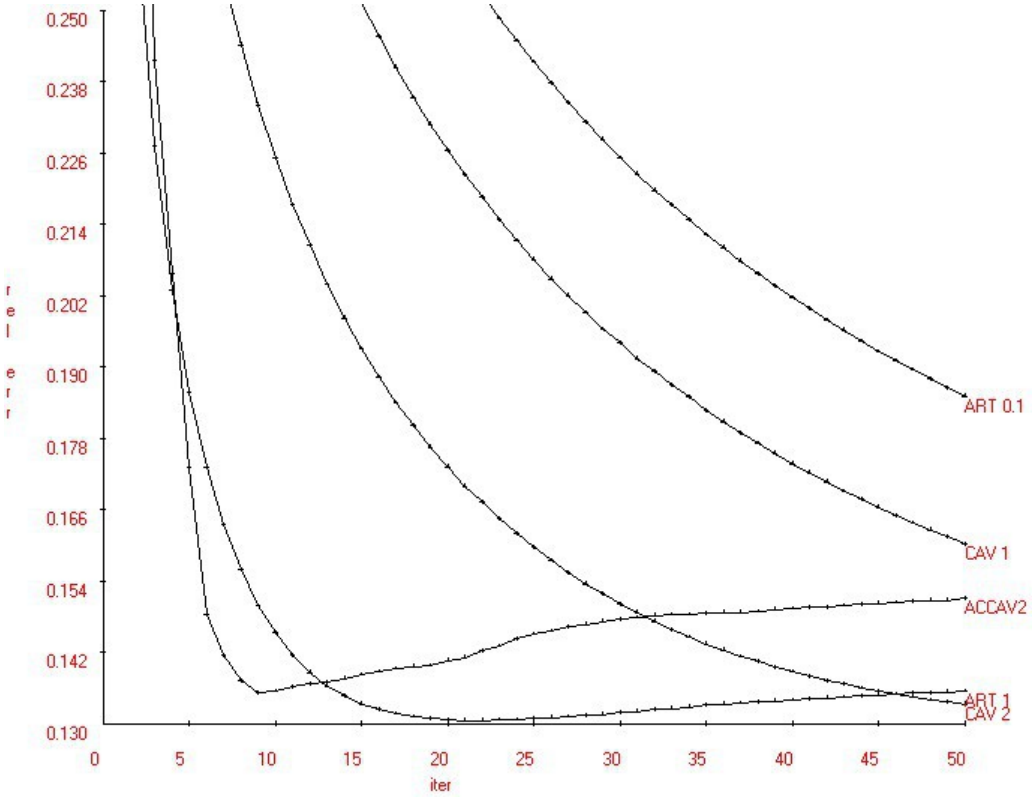

Figura 5.5. Error Relativo para caso 2.1 (115 × 115 pixels, 13.137 ecuaciones, 11.600 variables.)

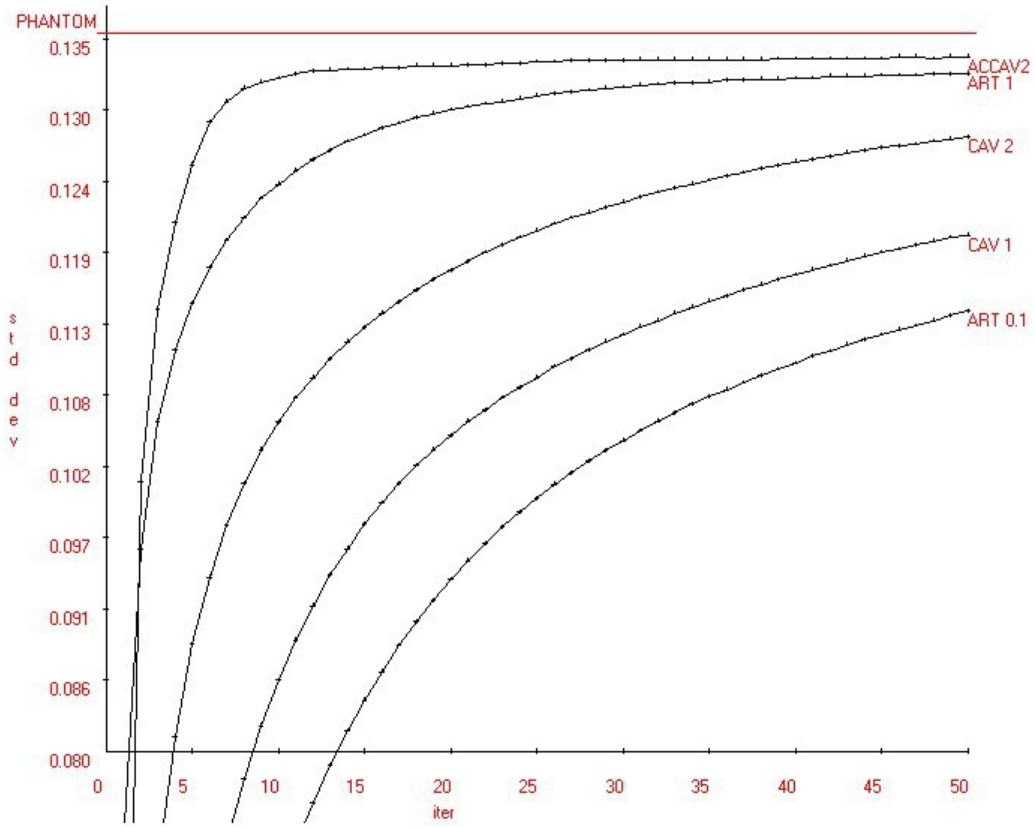

Figura 5.6. Desviación Estándar para caso 2.1 (115 × 115 pixels, 13.137 ecuaciones, 11.600 variables.)

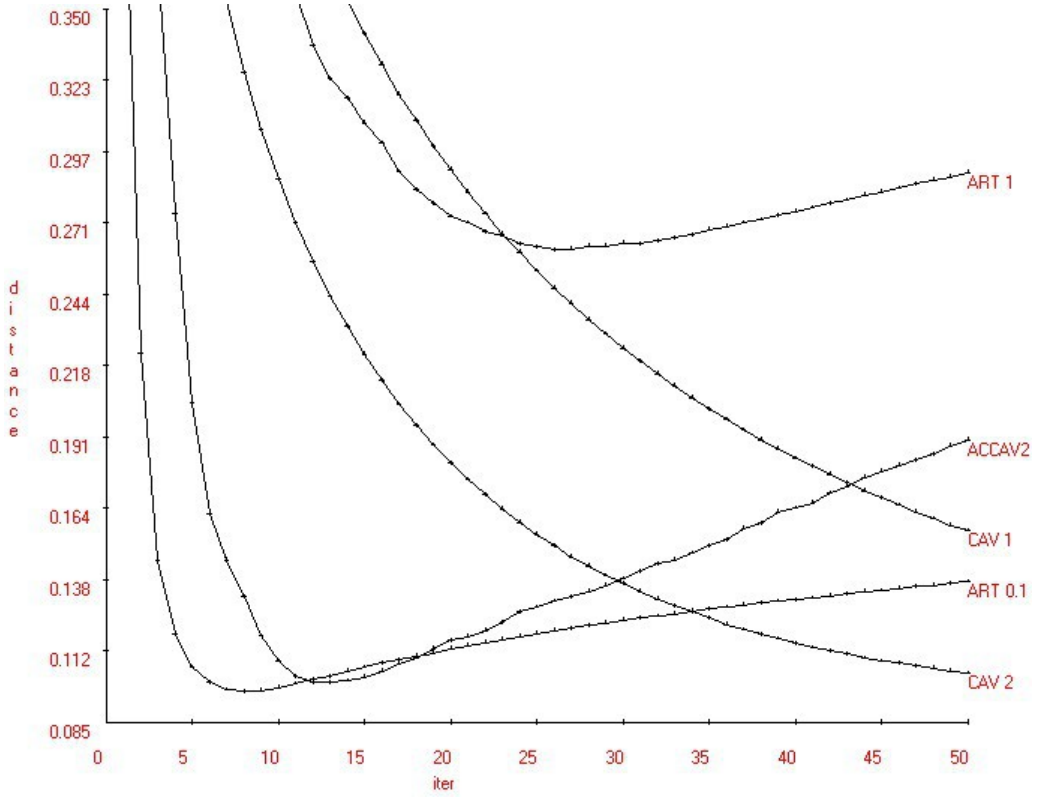

Figura 5.7. Distancia para caso 1.2 (115 × 115 pixels, 26.425 ecuaciones, 13.225 variables.)

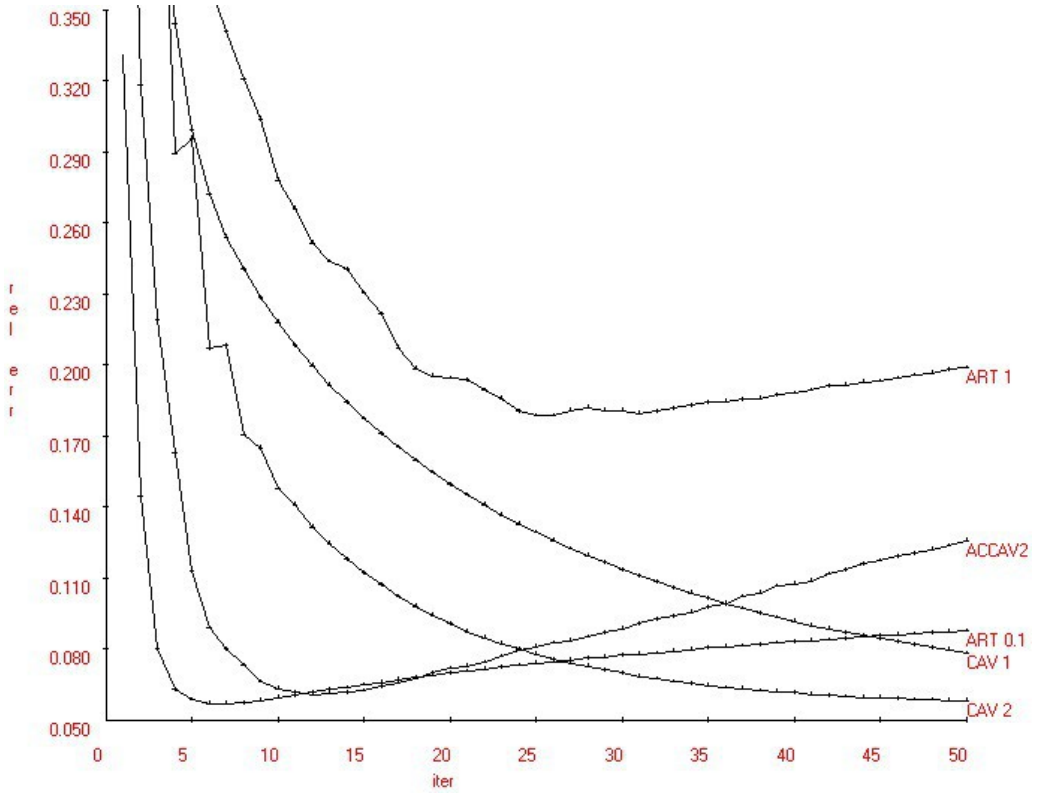

Figura 5.8. Error Relativo para caso 1.2 (115 × 115 pixels, 26.425 ecuaciones, 13.225 variables.)

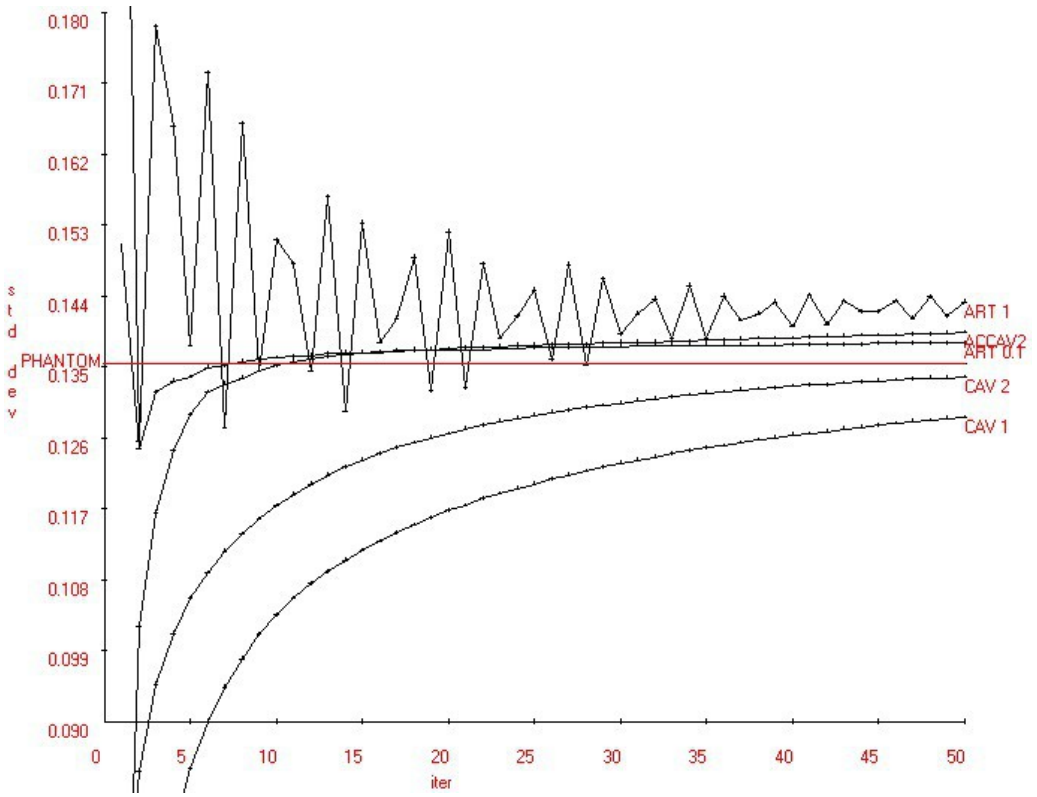

Figura 5.9. Desviación Estándar para caso 1.2 (115 × 115 pixels, 26.425 ecuaciones, 13.225 variables.)

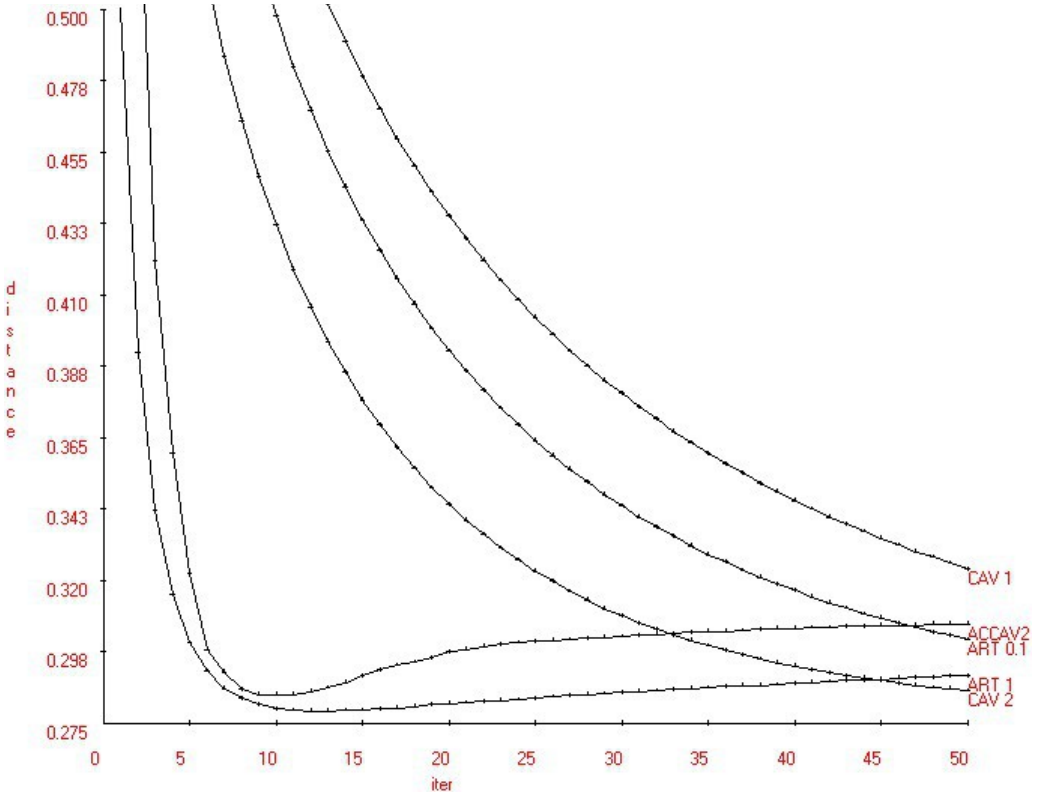

Figura 5.10. Distancia para caso 2.2 (115 × 115 pixels, 26.425 ecuaciones, 11.600 variables.)

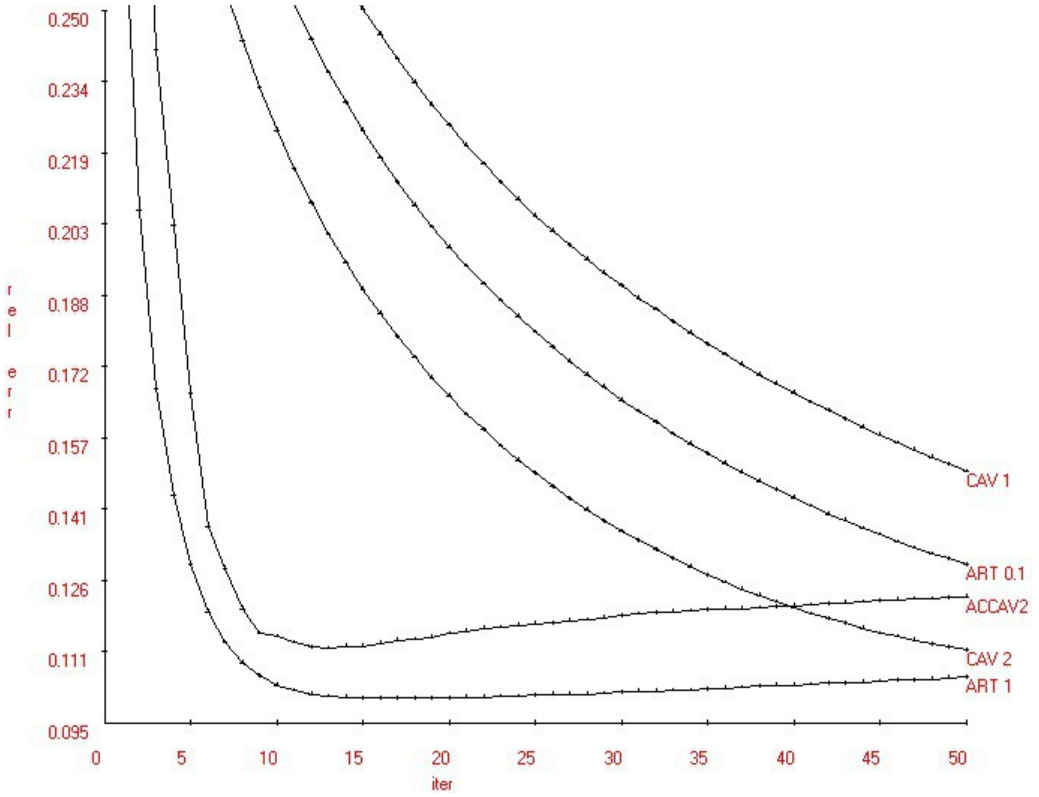

Figura 5.11. Error Relativo para caso 2.2 (115 × 115 pixels, 26.425 ecuaciones, 11.600 variables.)

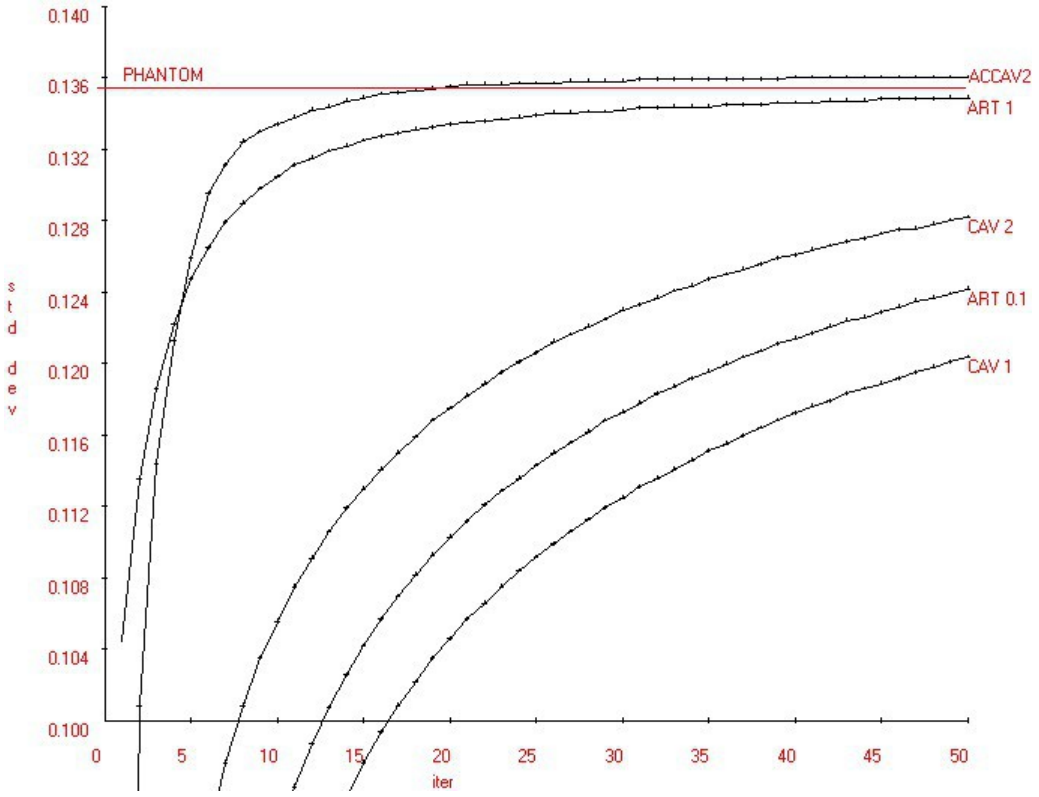

Figura 5.12. Desviación Estándar para caso 2.2 (115 × 115 pixels, 26.425 ecuaciones, 11.600 variables.)

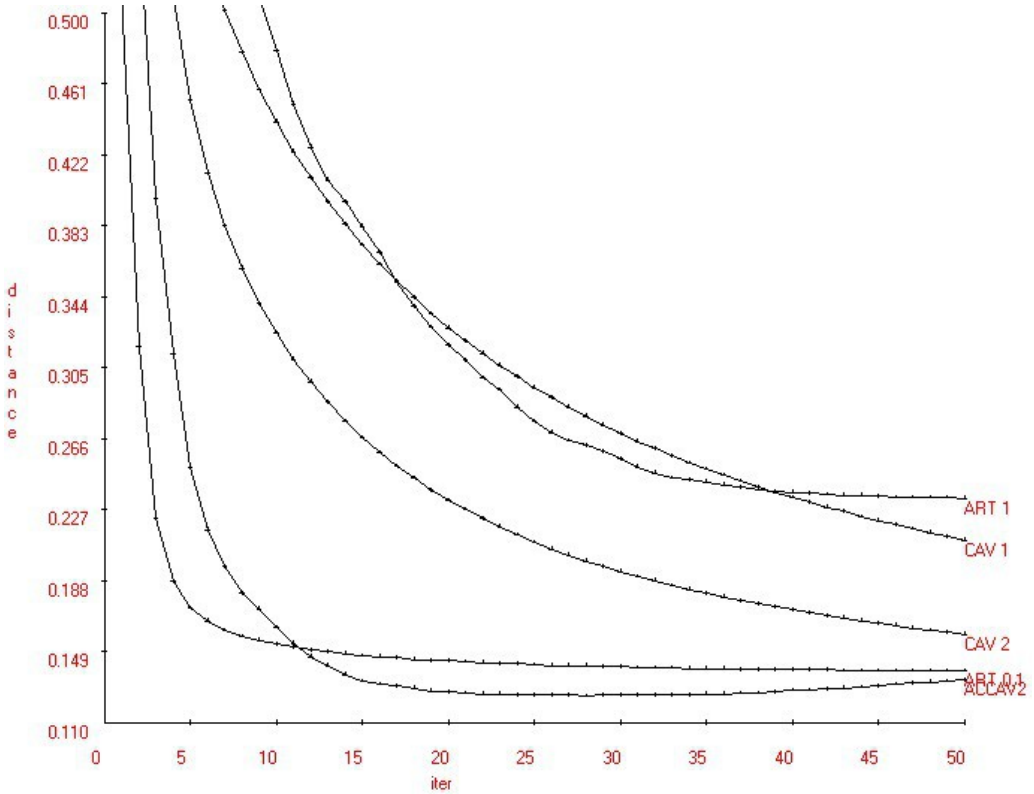

Figura 5.13. Distancia para caso 1.3 (345 × 345 pixels, 126.655 ecuaciones, 119.025 variables.)

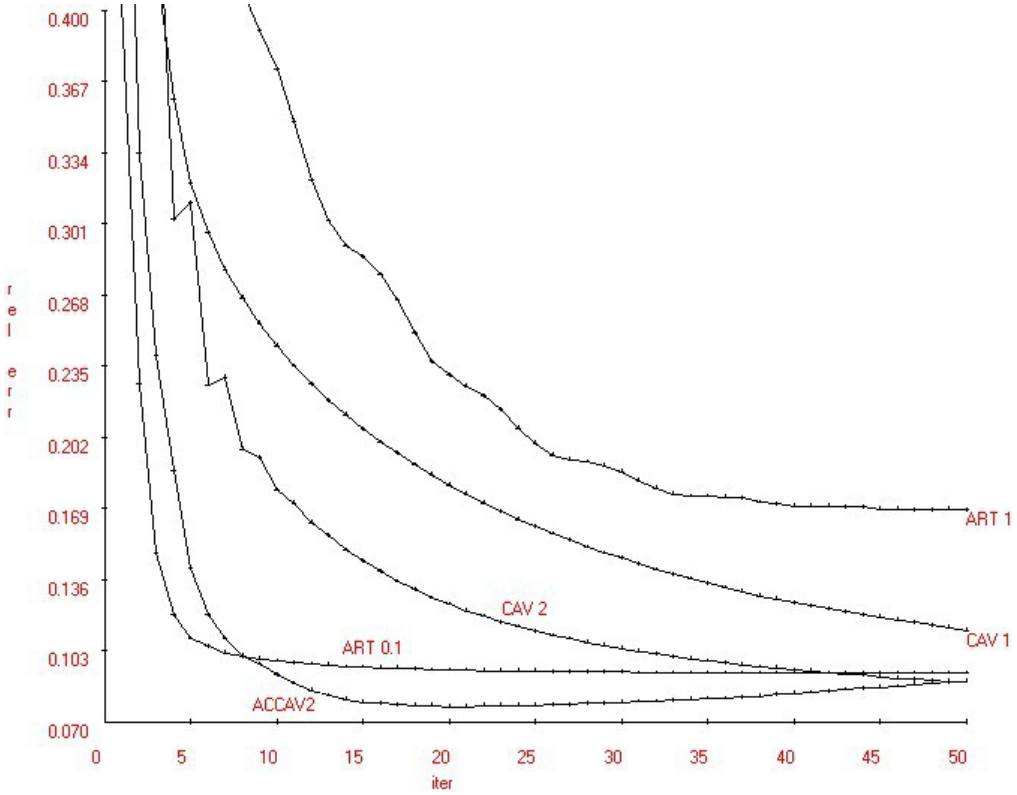

Figura 5.14. Error Relativo para caso 1.3 (345 × 345 pixels, 126.655 ecuaciones, 119.025 variables.)

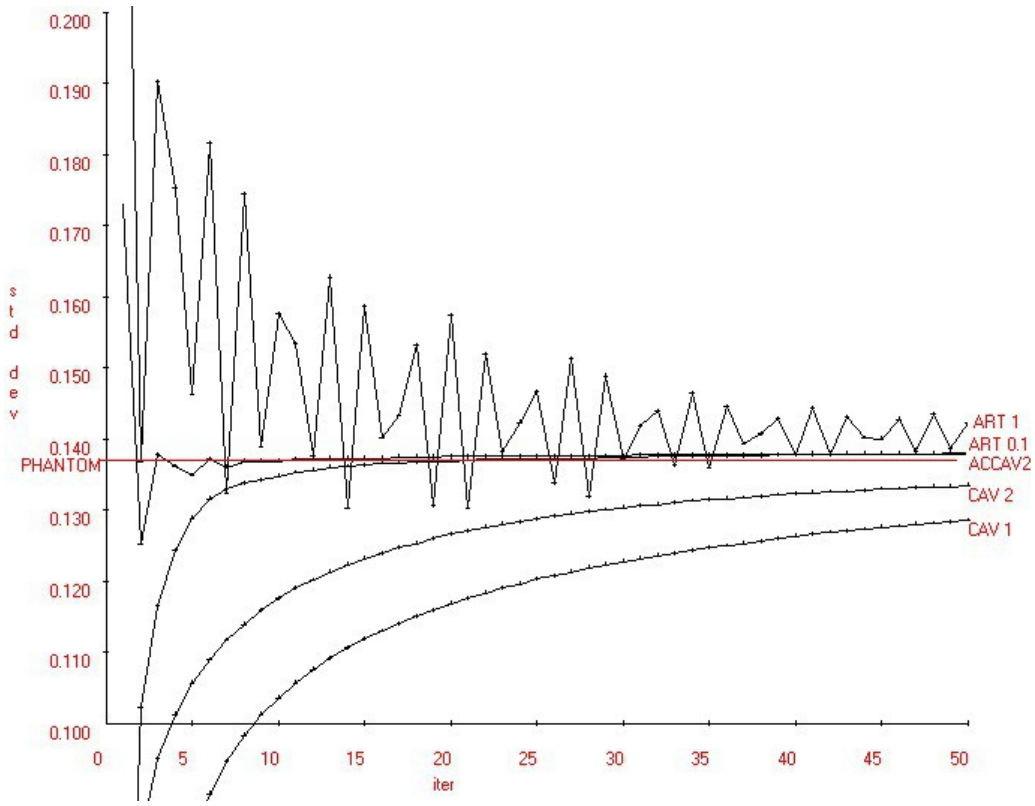

Figura 5.15. Desviación Estándar para caso 1.3 (345 × 345 pixels, 126.655 ecuaciones, 119.025 variables.)

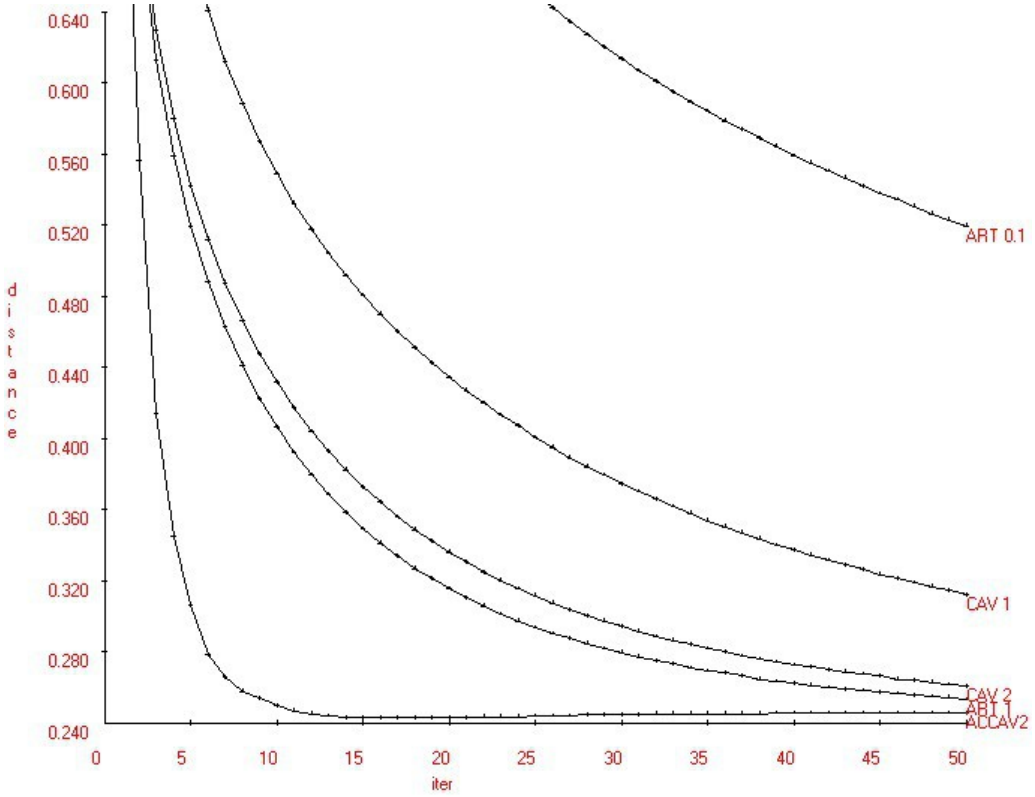

Figura 5.16. Distancia para caso 2.3 (345×345 pixels, 126.655 ecuaciones, 103.454 variables.)

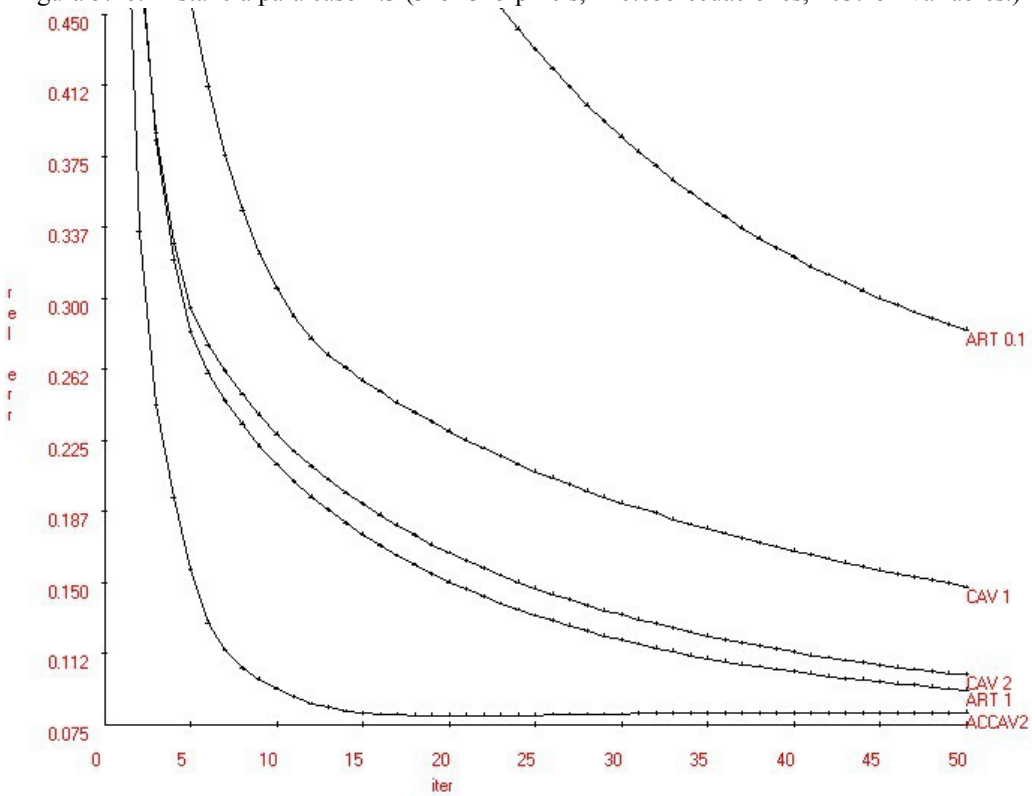

Figura 5.17. Error Relativo para caso 2.3 (345×345 pixels, 126.655 ecuaciones, 103.454 variables.)

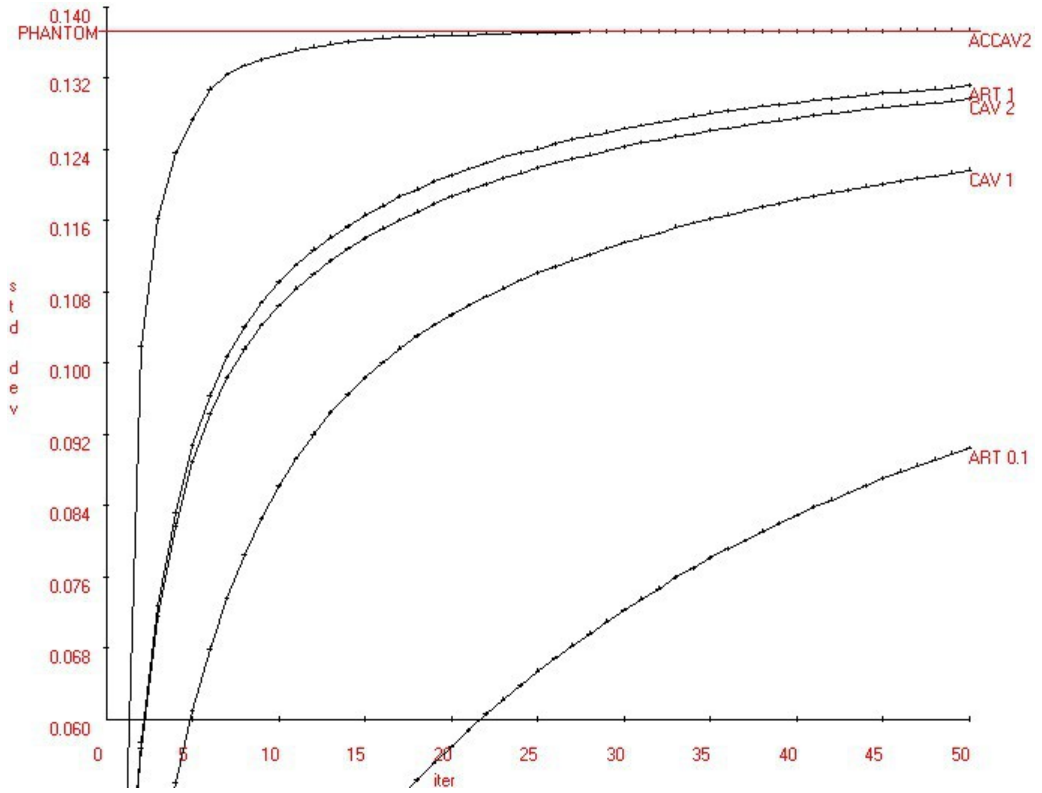

Figura 5.18. Desviación Estándar para caso 2.3 (345×345 pixels, 126.655 ecuaciones, 103.454 variables.)

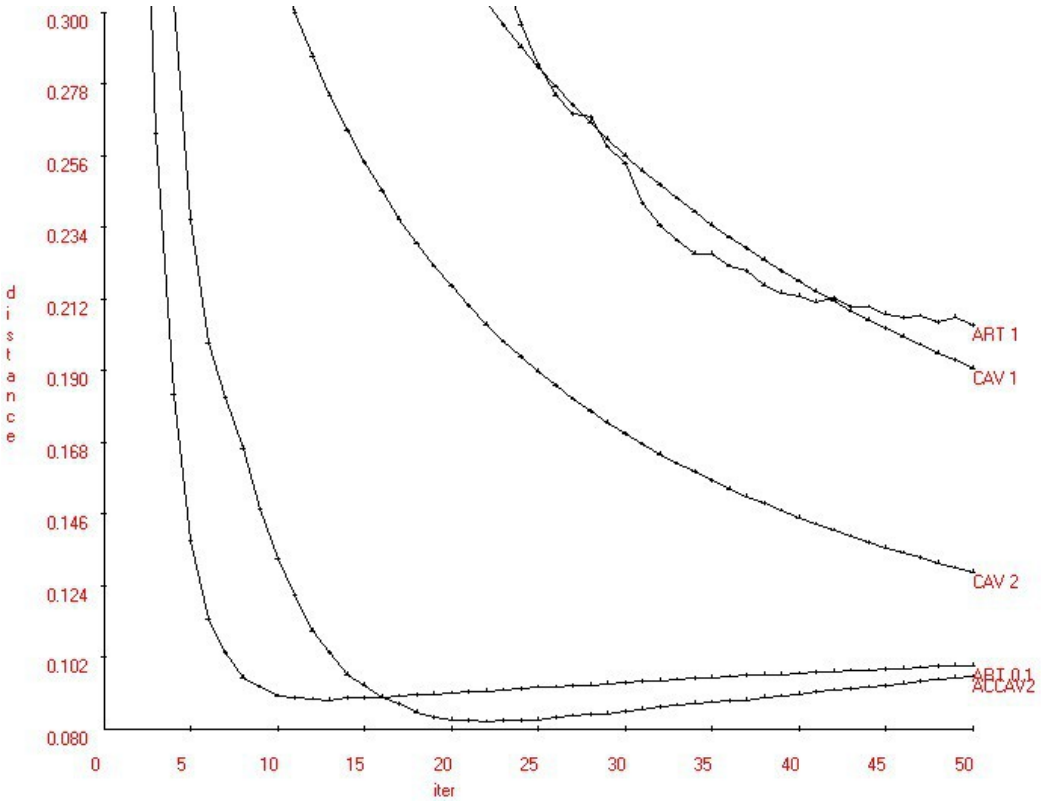

Figura 5.19. Distancia para caso 1.4 (345×345 pixels, 232.275 ecuaciones, 119.025 variables.)

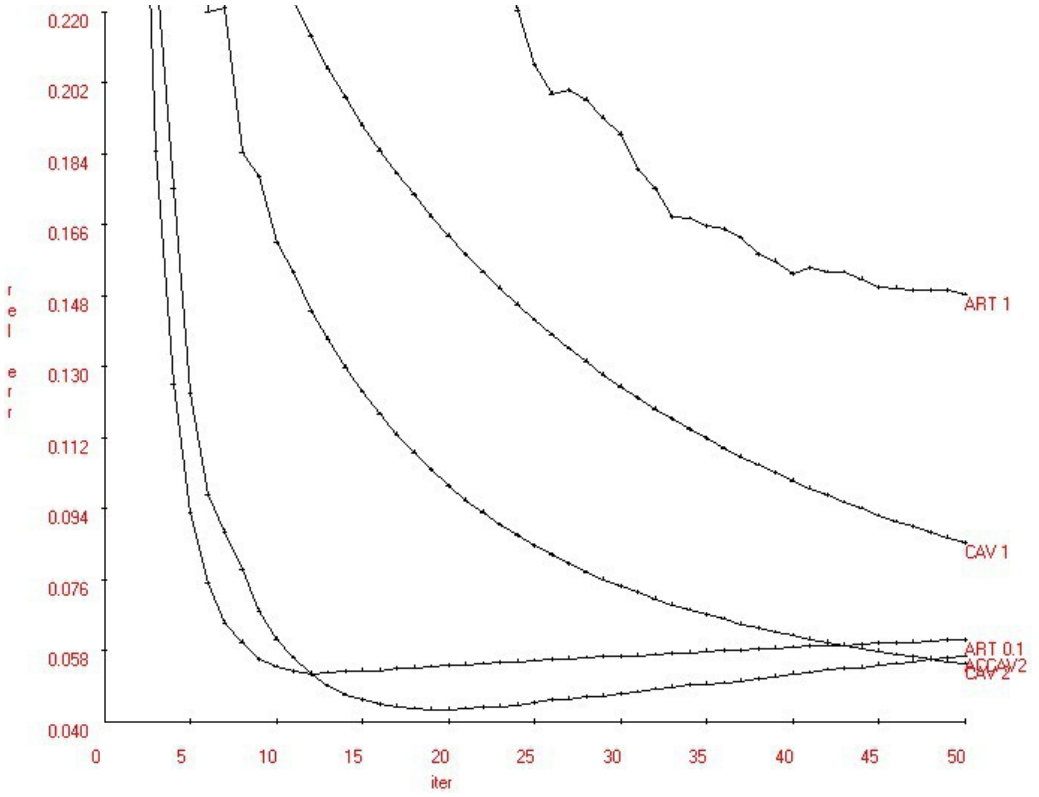

Figura 5.20. Error Relativo para caso 1.4 (345×345 pixels, 232.275 ecuaciones, 119.025 variables.)

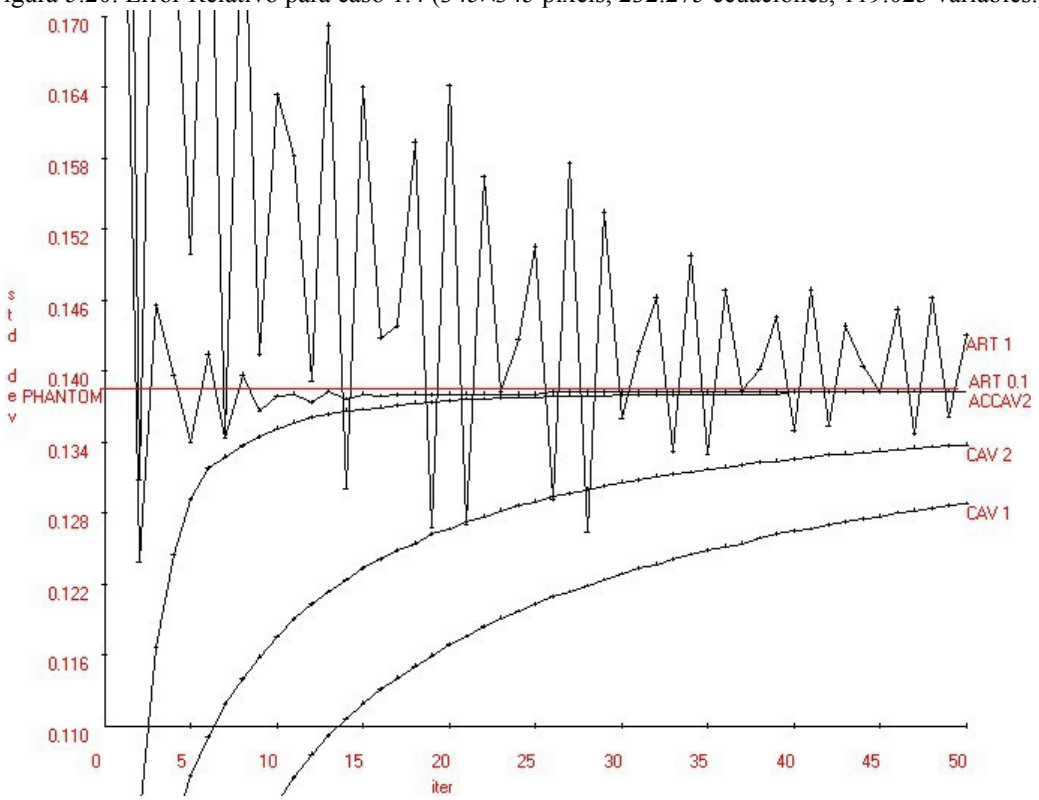

Figura 5.21. Desviación Estándar para caso 1.4 (345×345 pixels, 232.275 ecuaciones, 119.025 variables.)

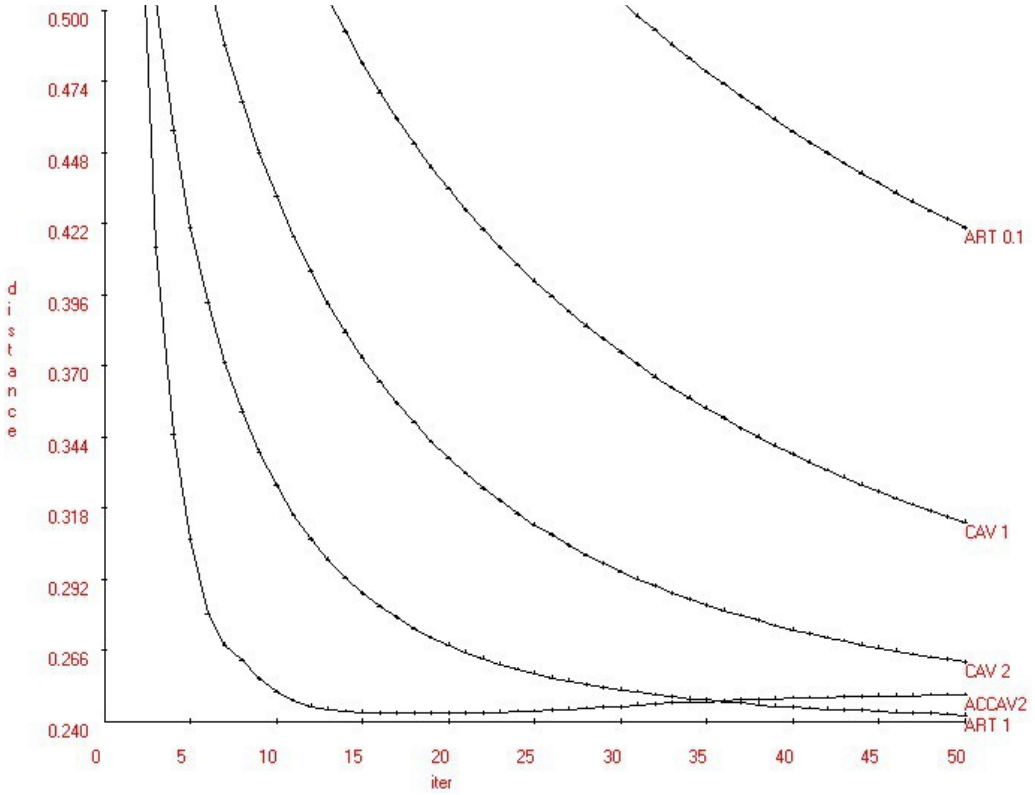

Figura 5.22. Distancia para caso 2.4 (345×345 pixels, 232.275 ecuaciones, 103.454 variables.)

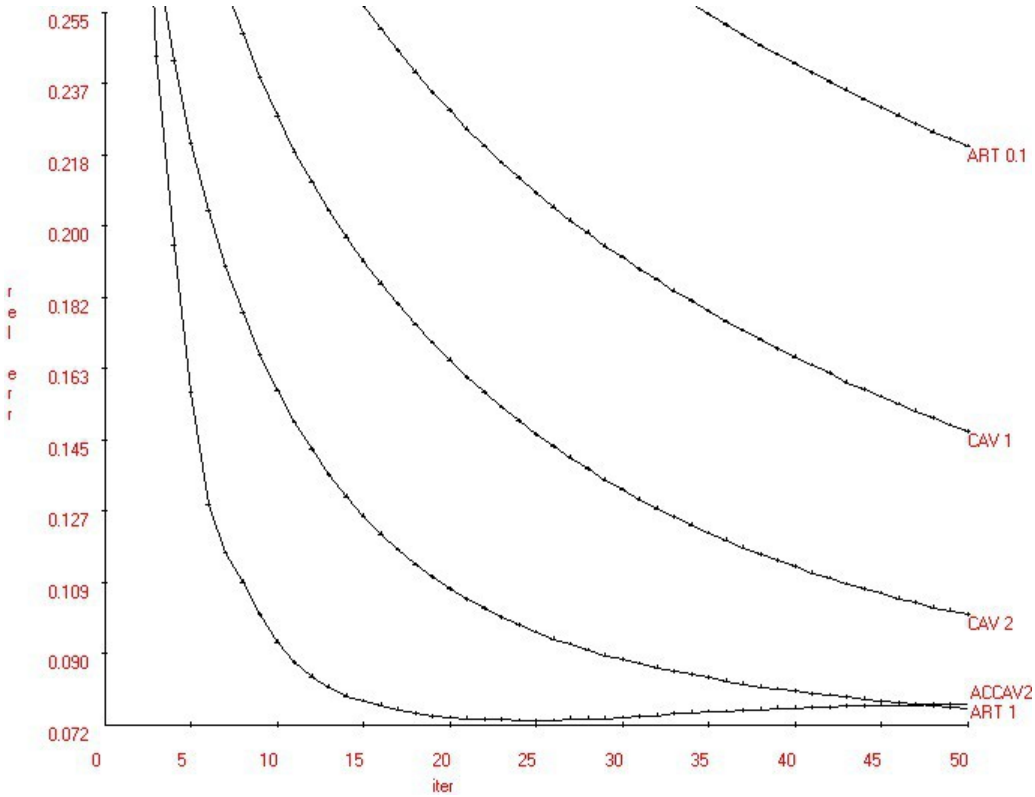

Figura 5.23. Error Relativo para caso 2.4 (345×345 pixels, 232.275 ecuaciones, 103.454 variables.)

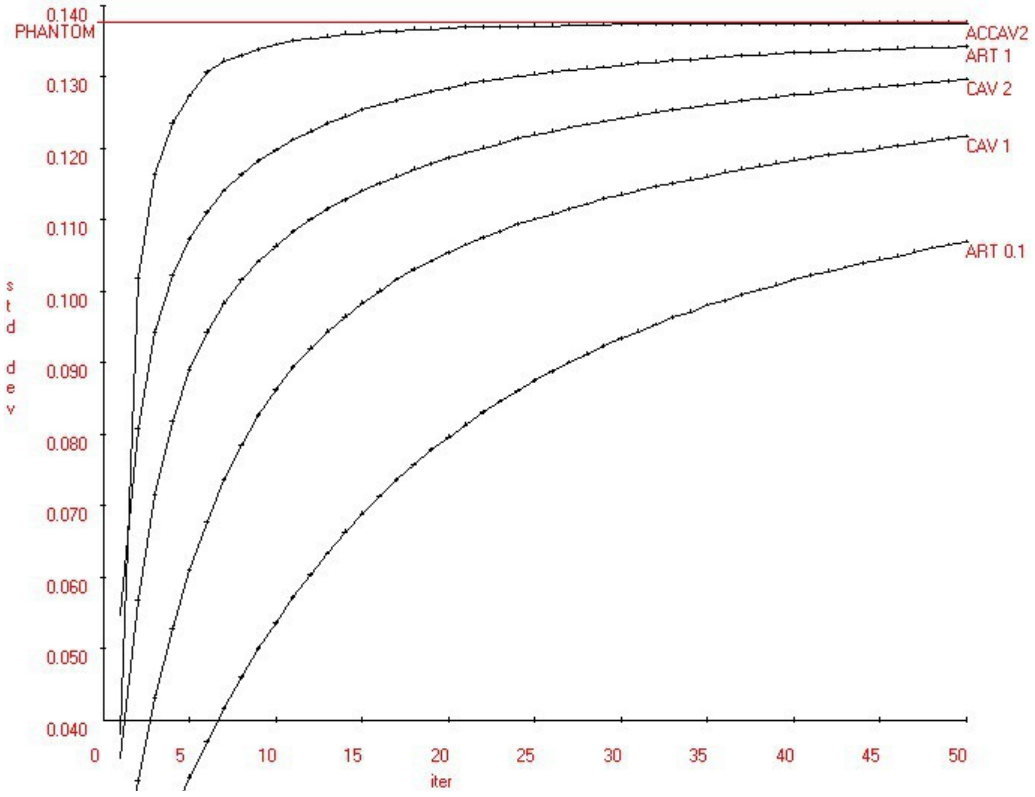

Figura 5.24. Desviación Estándar para caso 2.4 (345×345 pixels, 232.275 ecuaciones, 103.454 variables.)

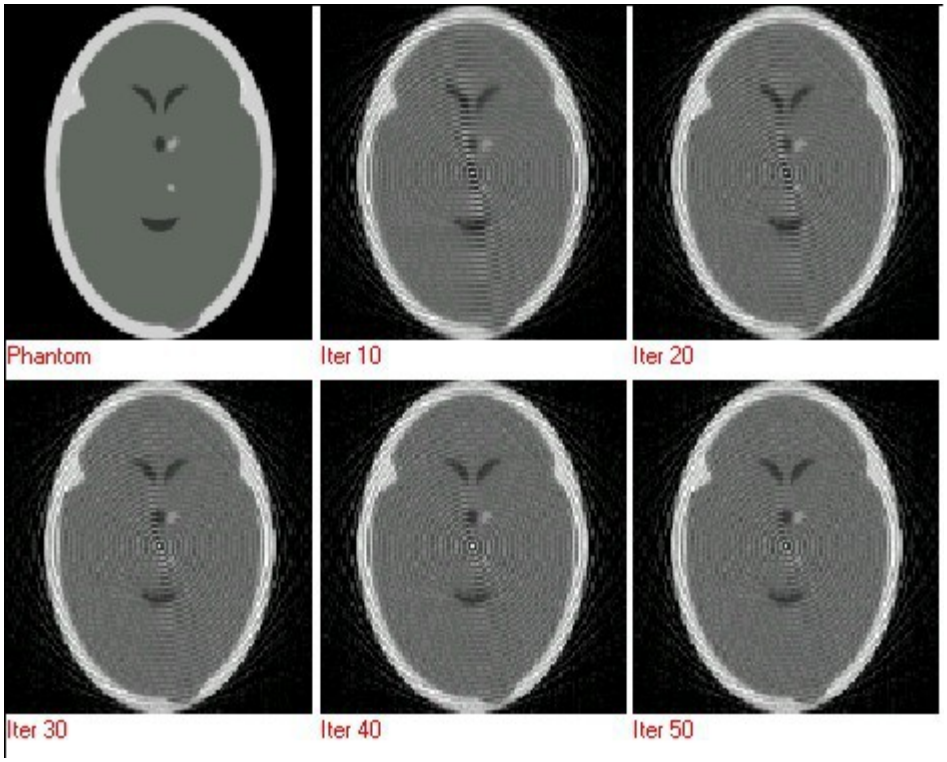

Figura 5.25. Imágenes producidas por ART 0.1 en caso 1.1 (115 × 115 pixels, 13.137 ecuacs., 13.225 vars.)

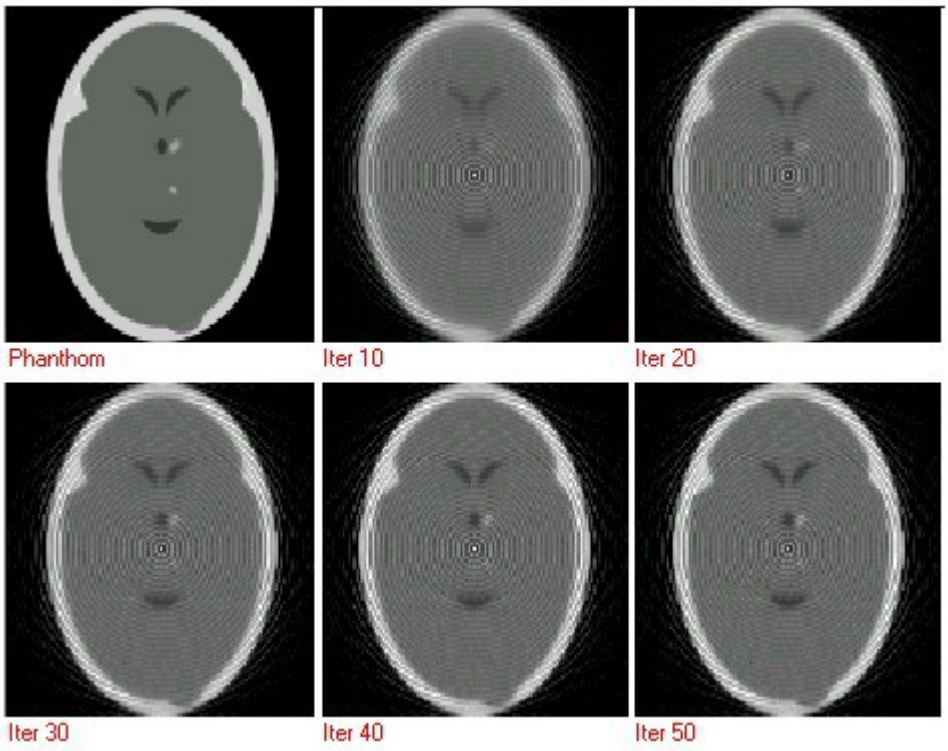

Figura 5.26. Imágenes producidas por CAV 2 en caso 1.1 (115 × 115 pixels, 13.137 ecuacs., 13.225 vars.)

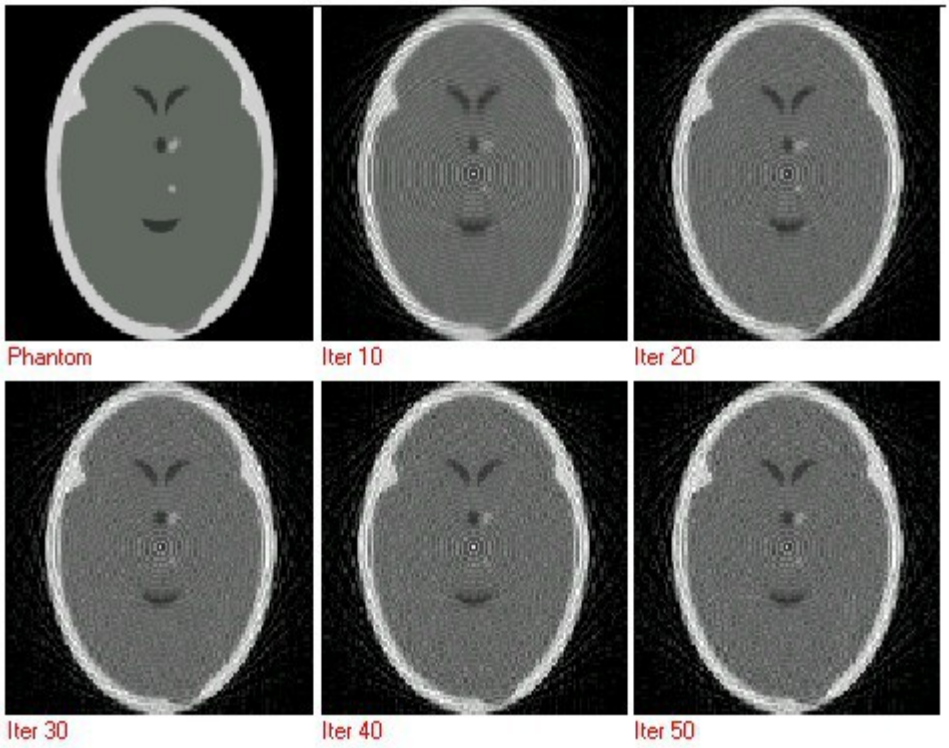

Figura 5.27. Imágenes producidas por ACCAV2 en caso 1.1 (115 × 115 pixels, 13.137 ecuacs., 13.225 vars.)

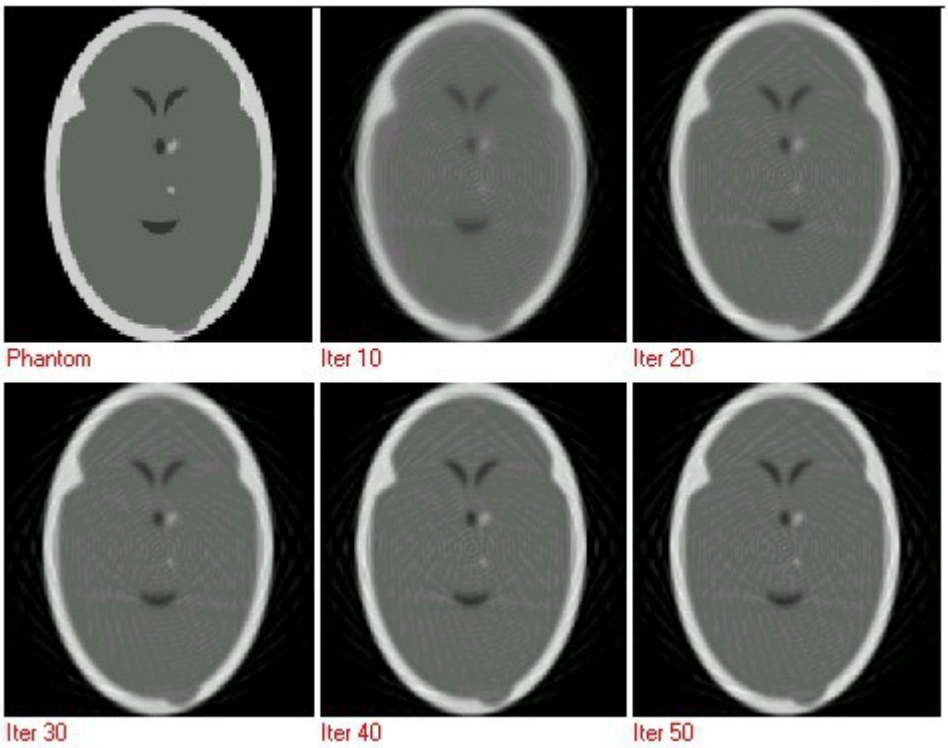

Figura 5.28. Imágenes producidas por ART 1 en caso 2.1 (115 × 115 pixels, 13.137 ecuacs., 11.600 vars.)

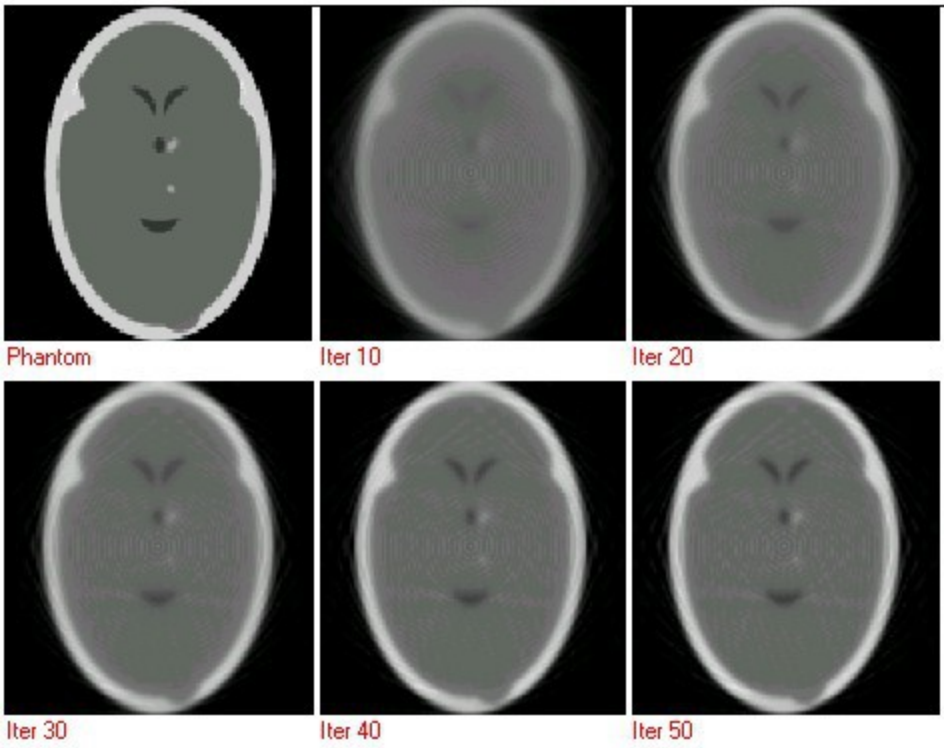

Figura 5.29. Imágenes producidas por CAV 2 en caso 2.1 (115 × 115 pixels, 13.137 ecuacs., 11.600 vars.)

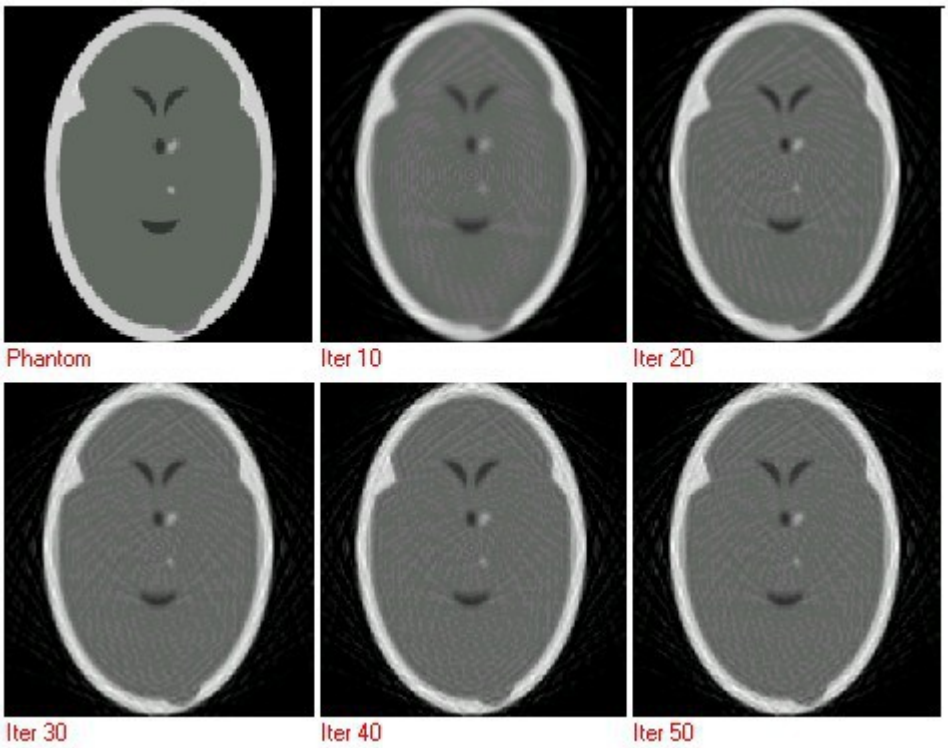

Figura 5.30. Imágenes producidas por ACCAV2 en caso 2.1 (115 × 115 pixels, 13.137 ecuacs., 11.600 vars.)

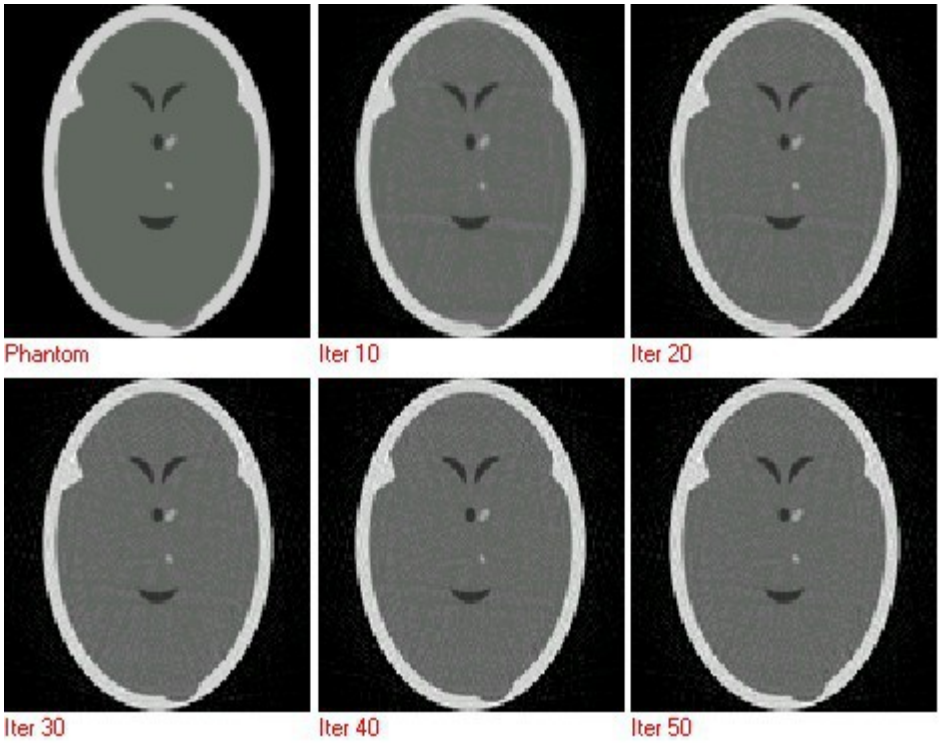

Figura 5.31. Imágenes producidas por ART 0.1 en caso 1.2 (115 × 115 pixels, 26.425 ecuacs., 13.225 vars.)

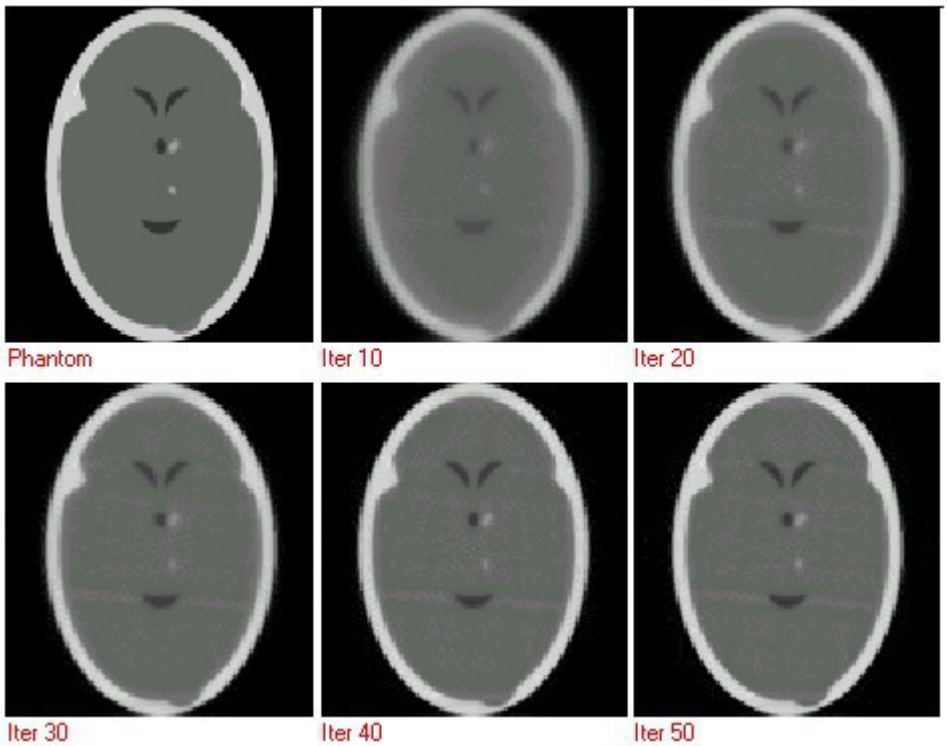

Figura 5.32. Imágenes producidas por CAV 2 en caso 1.2 (115 × 115 pixels, 26.425 ecuacs., 13.225 vars.)

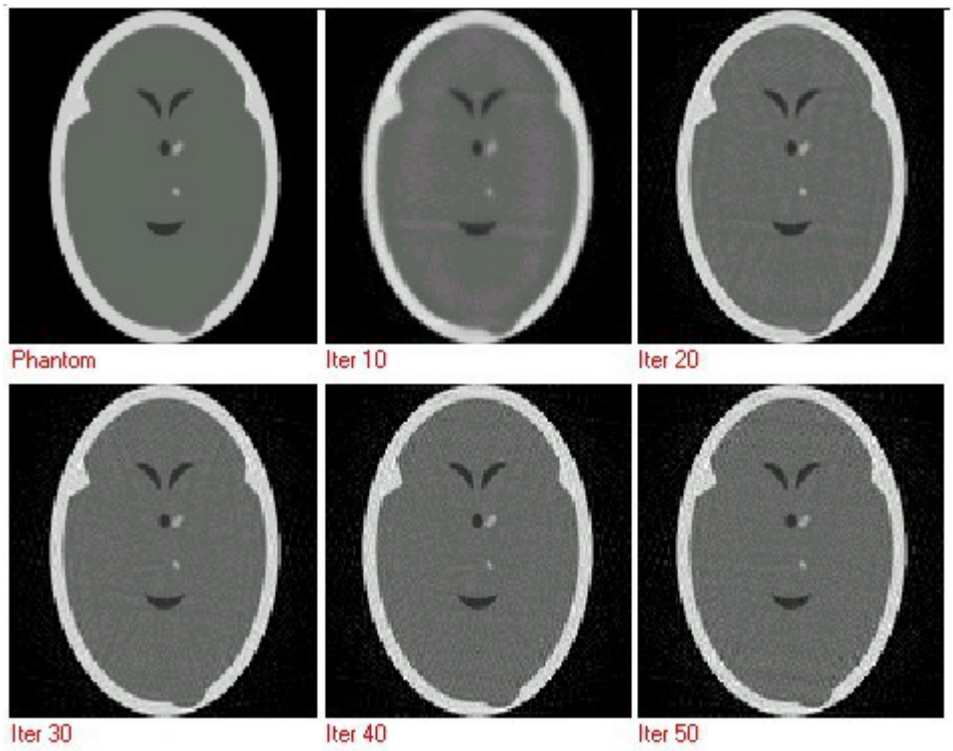

Figura 5.33. Imágenes producidas por ACCAV2 en caso 1.2 (115 × 115 pixels, 26.425 ecuacs., 13.225 vars.)

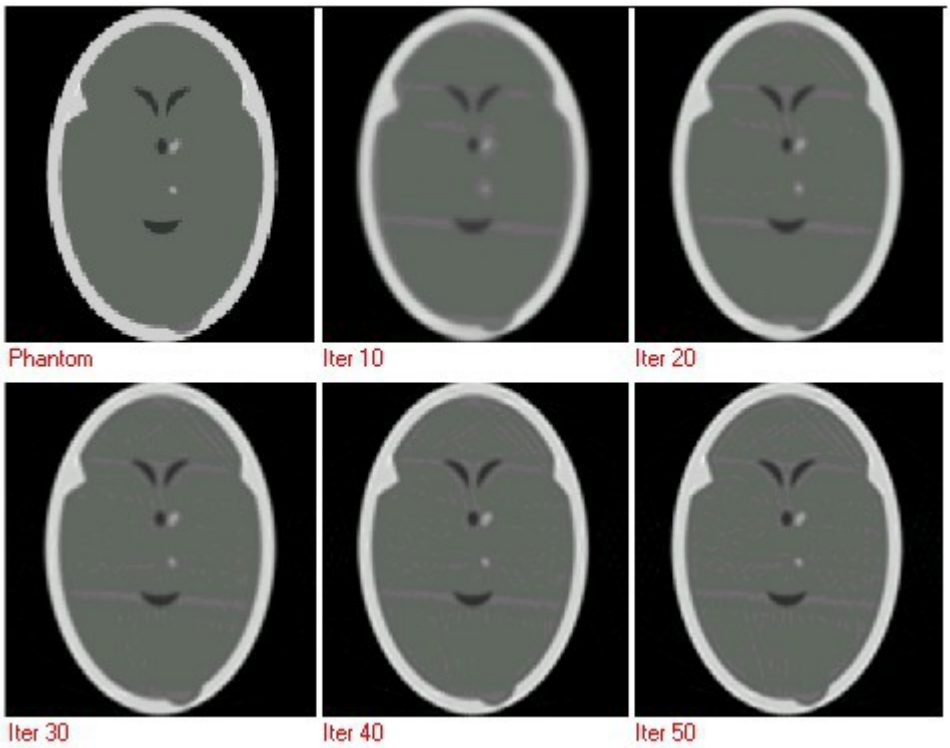

Figura 5.34. Imágenes producidas por ART 1 en caso 2.2 (115 × 115 pixels, 26.425 ecuacs., 11.600 vars.)

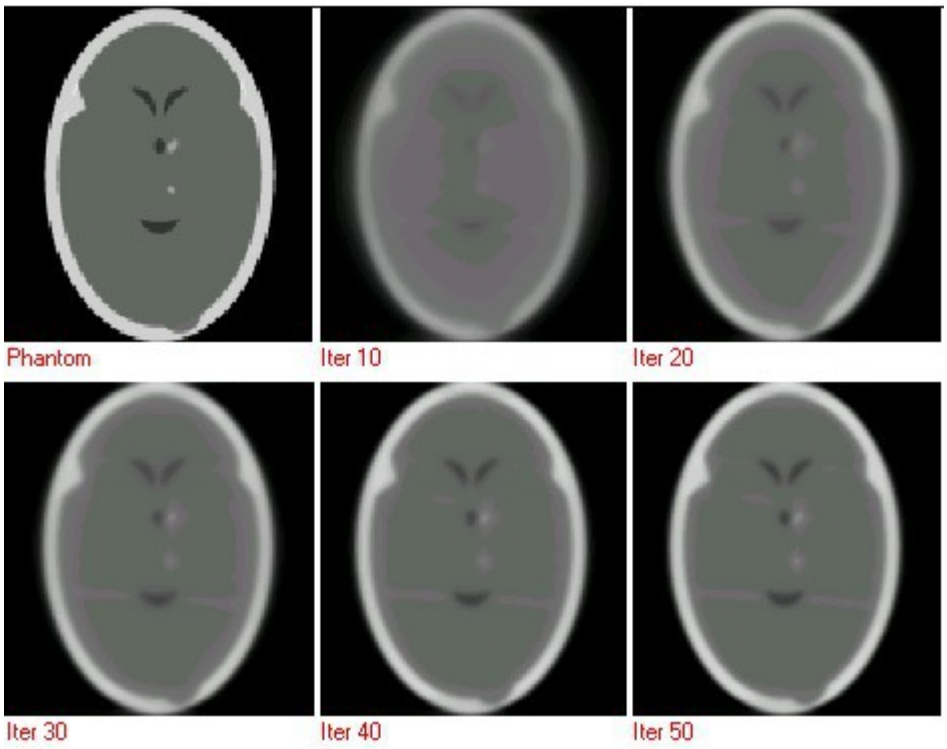

Figura 5.35. Imágenes producidas por CAV 2 en caso 2.2 (115 × 115 pixels, 26.425 ecuacs., 11.600 vars.)

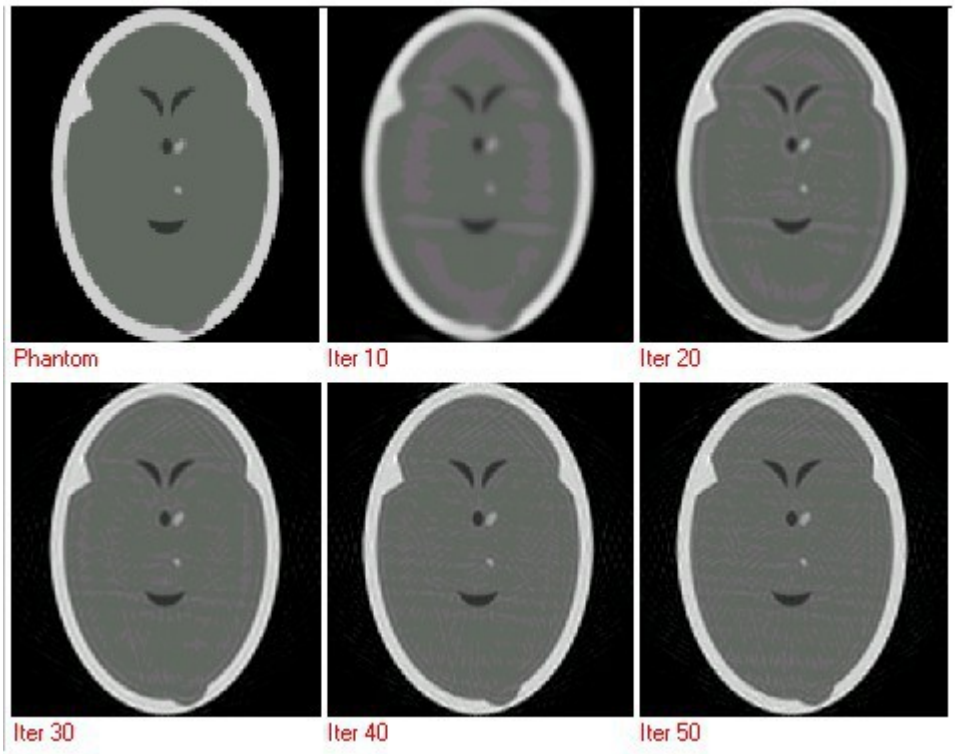

Figura 5.36. Imágenes producidas por ACCAV2 en caso 2.2 (115 × 115 pixels, 26.425 ecuacs., 11.600 vars.)

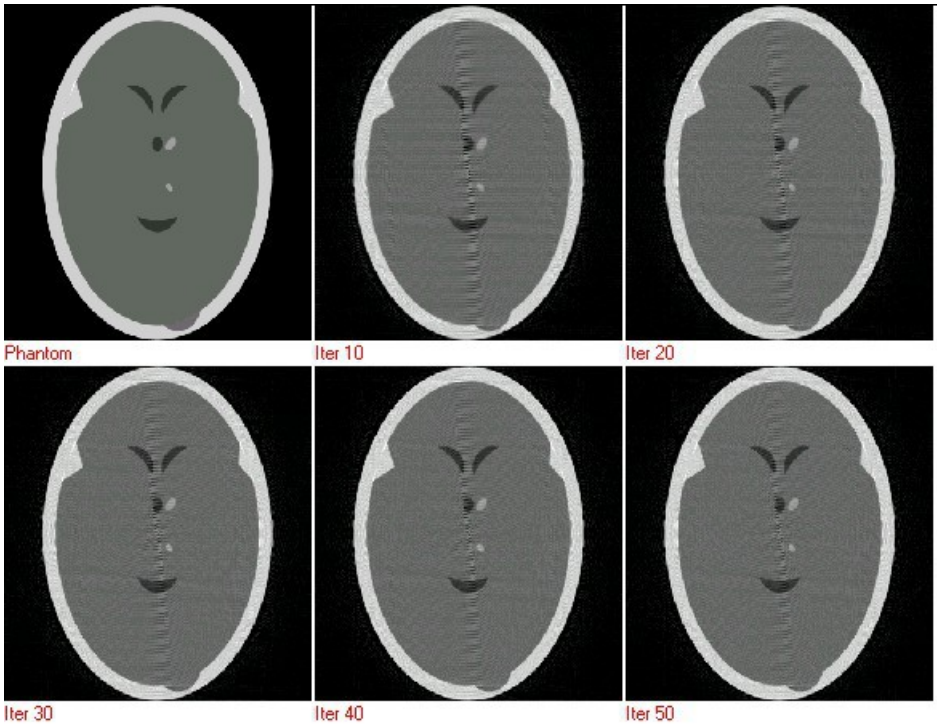

Figura 5.37. Imágenes producidas por ART 0.1 en caso 1.3 (345 × 345 pixels, 126.655 ecuacs., 119.025 vars.)

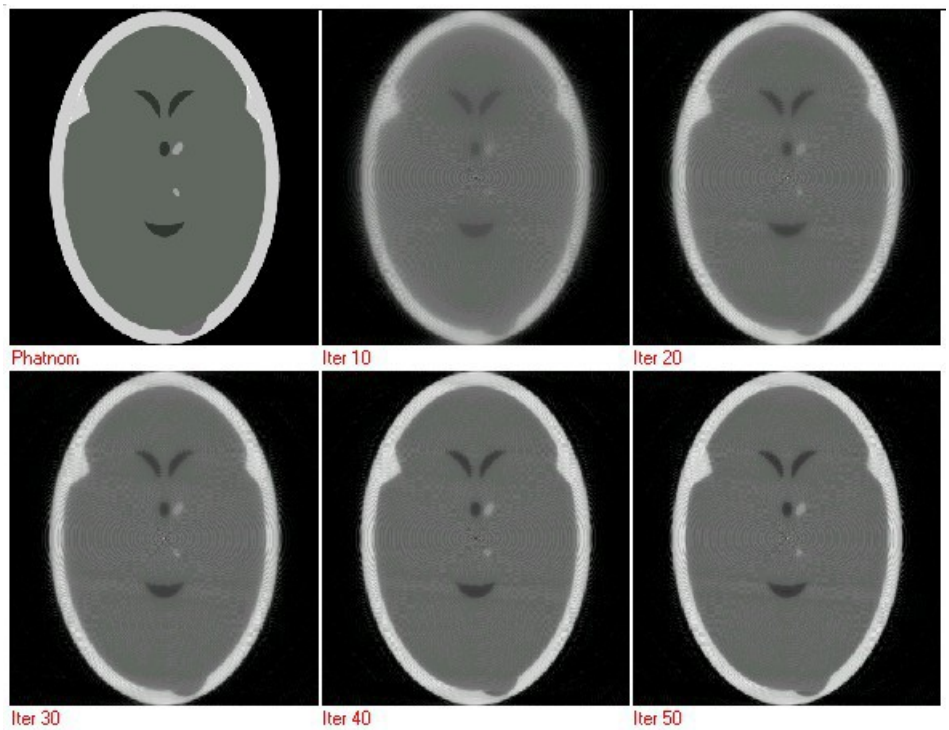

Figura 5.38. Imágenes producidas por CAV 2 en caso 1.3 (345 × 345 pixels, 126.655 ecuacs., 119.025 vars.)

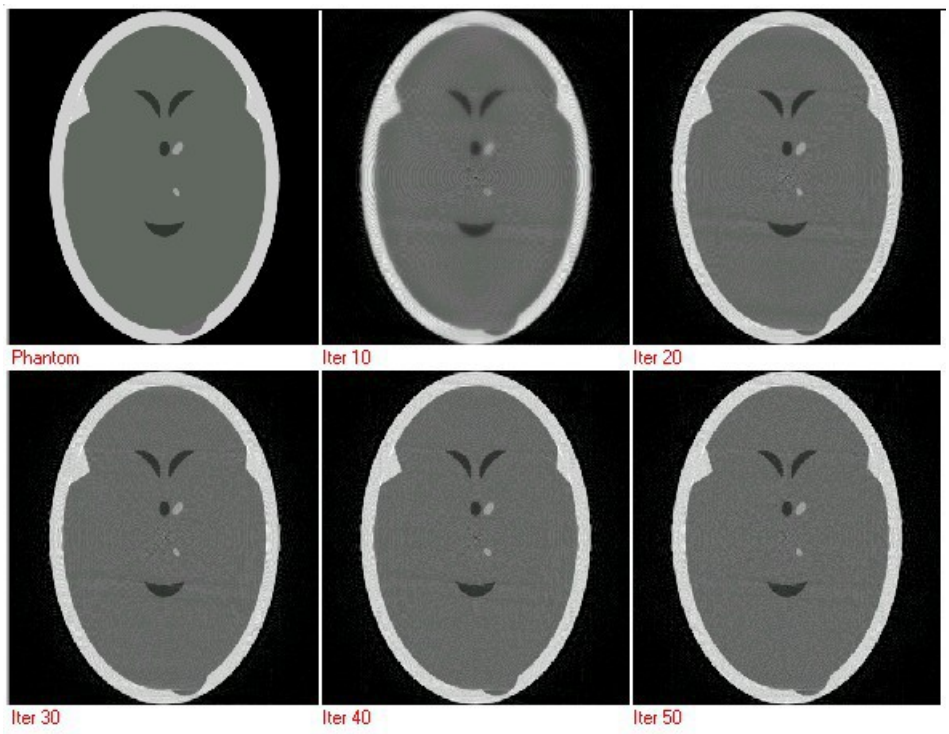

Figura 5.39. Imágenes producidas por ACCAV2 en caso 1.3 (345×345 pixels, 126.655 ecuacs., 119.025 vars.)

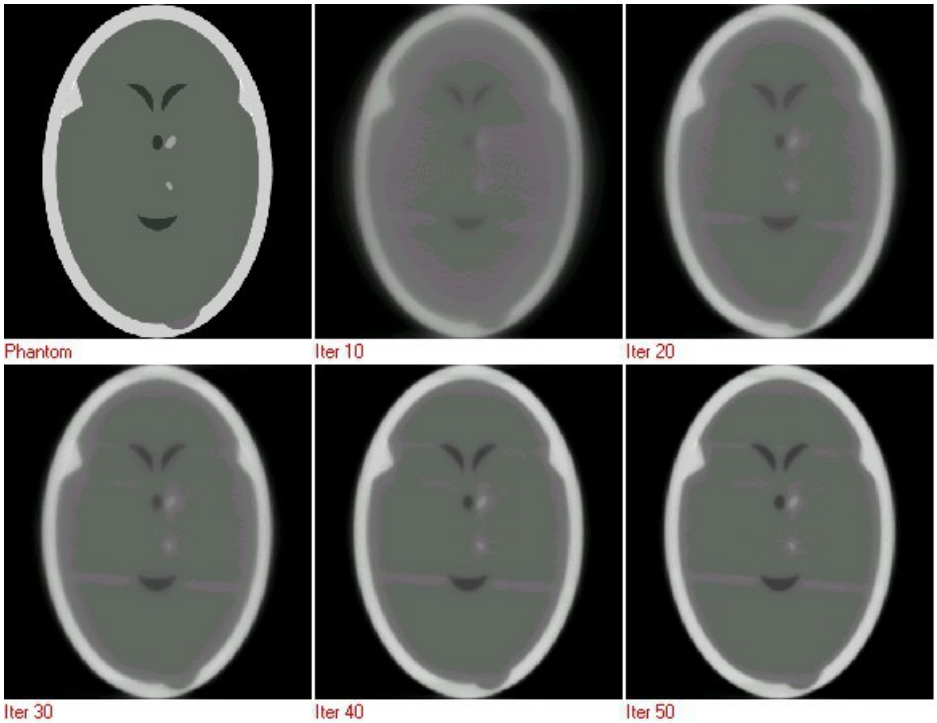

Figura 5.40. Imágenes producidas por ART 1 en caso 2.3 (345×345 pixels, 126.655 ecuacs., 103.454 vars.)

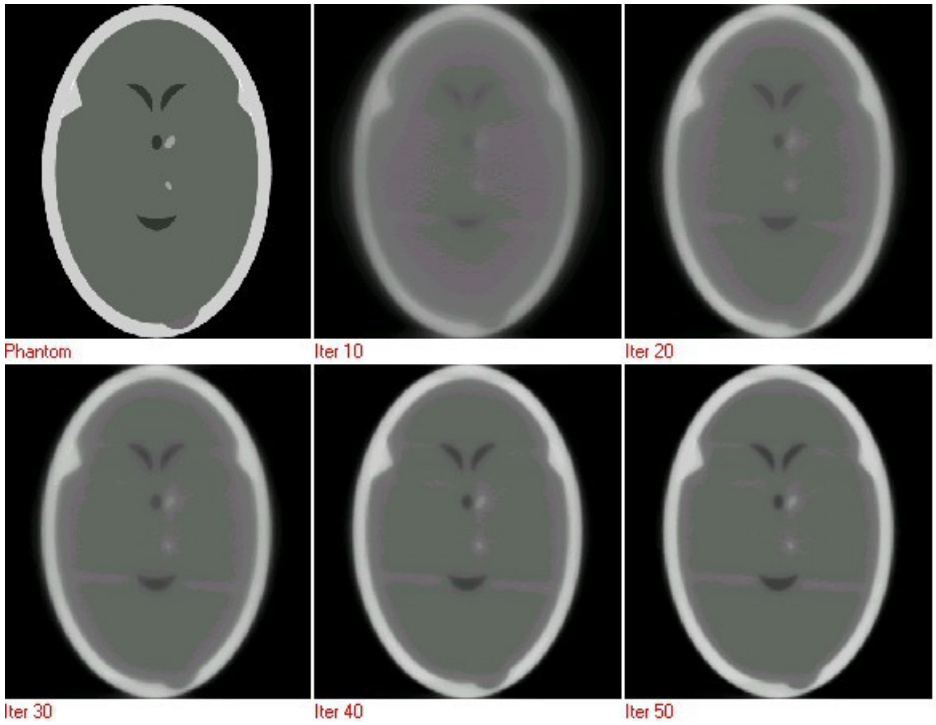

Figura 5.41. Imágenes producidas por CAV 2 en caso 2.3 (345×345 pixels, 126.655 ecuacs., 103.454 vars.)

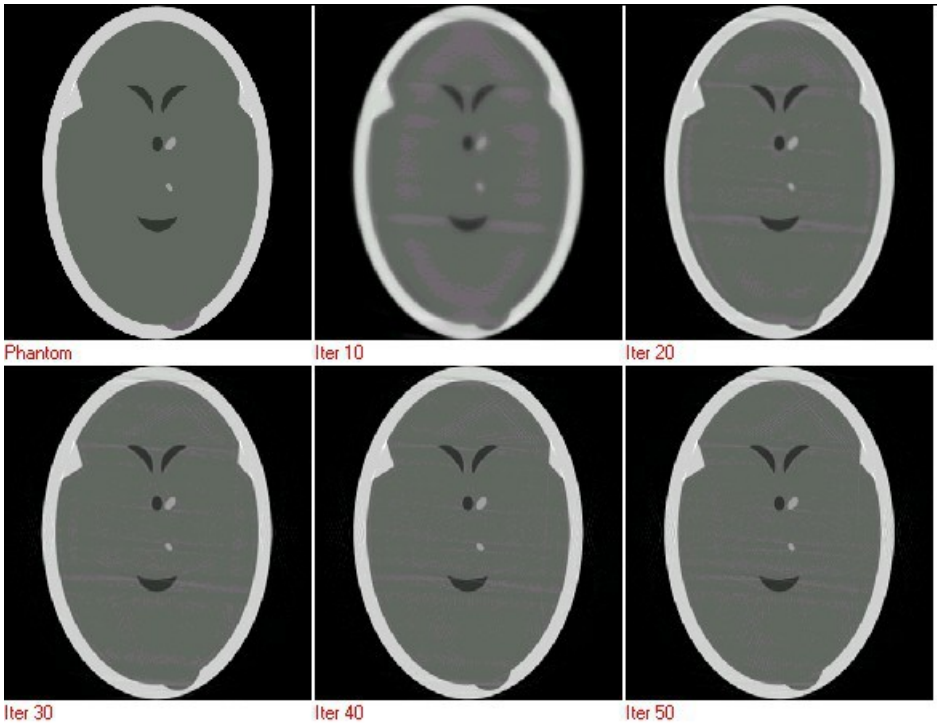

Figura 5.42. Imágenes producidas por ACCAV2 en caso 2.3 (345×345 pixels, 126.655 ecuacs., 103.454 vars.)

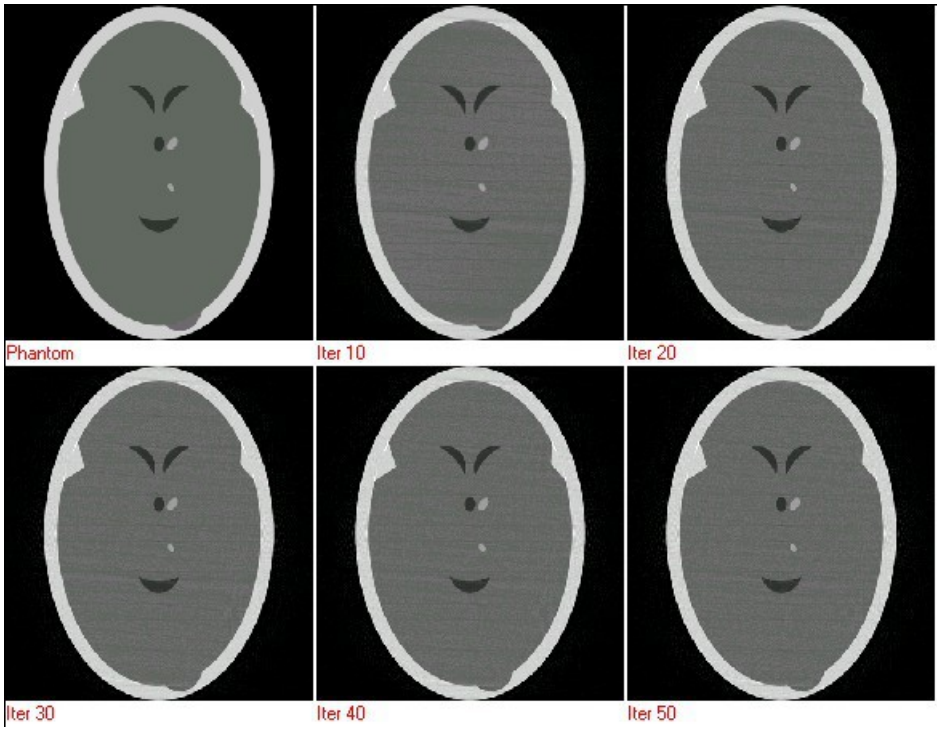

Figura 5.43. Imágenes producidas por ART 0.1 en caso 1.4 (345×345 pixels, 232.275 ecuacs., 119.025 vars.)

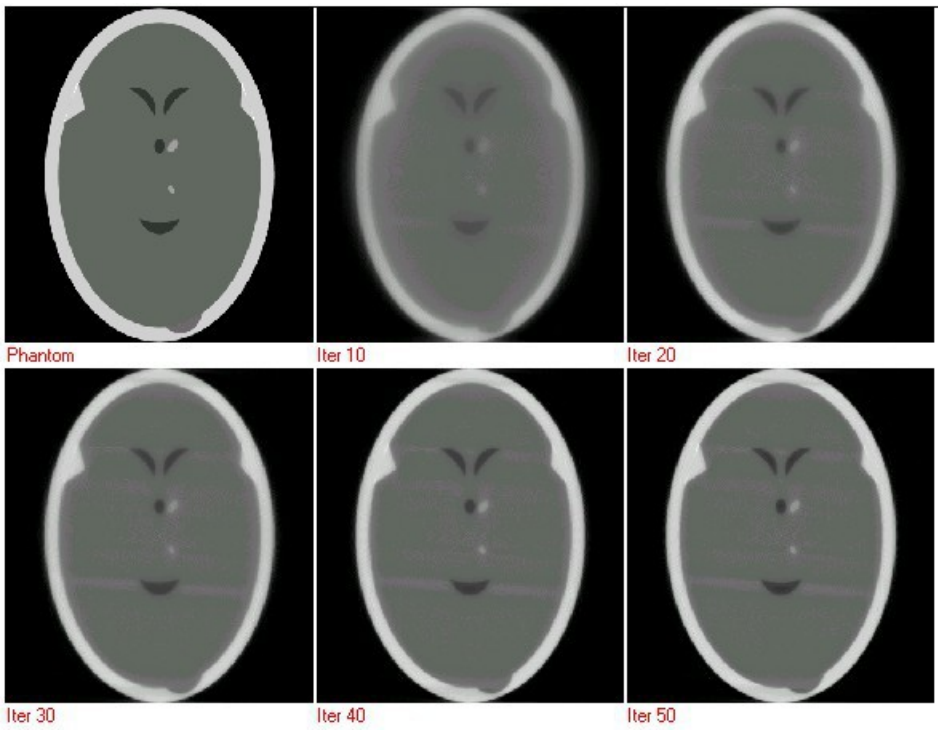

Figura 5.44. Imágenes producidas por CAV 2 en caso 1.4 (345×345 pixels, 232.275 ecuacs., 119.025 vars.)

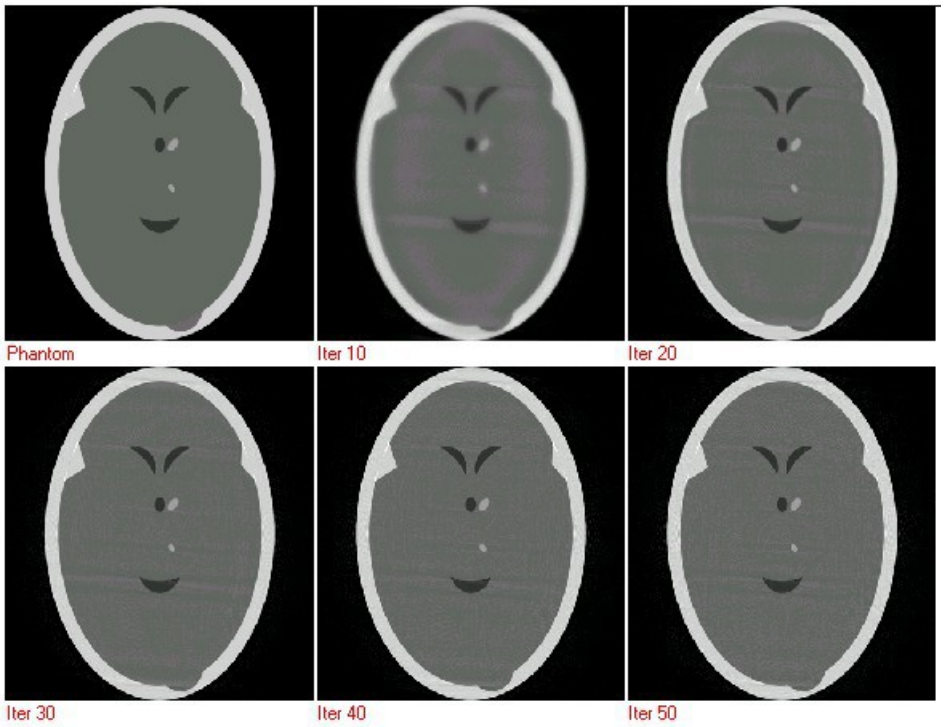

Figura 5.45. Imágenes producidas por ACCAV2 en caso 1.4 (345×345 pixels, 232.275 ecuacs., 119.025 vars.)

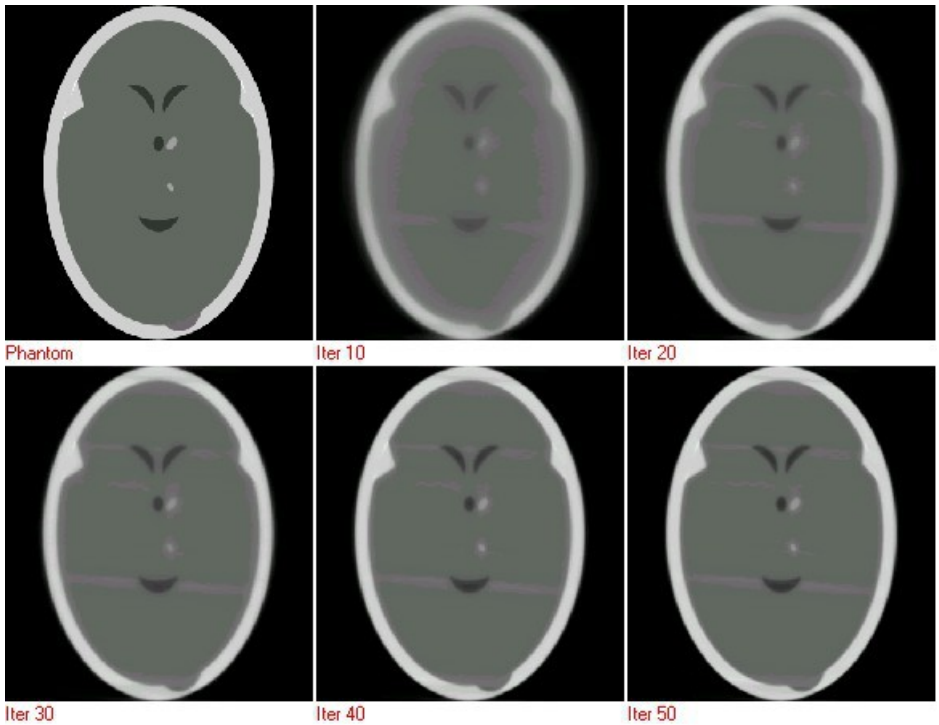

Figura 5.46. Imágenes producidas por ART 1 en caso 2.4 (345×345 pixels, 232.275 ecuacs., 103.454 vars.)

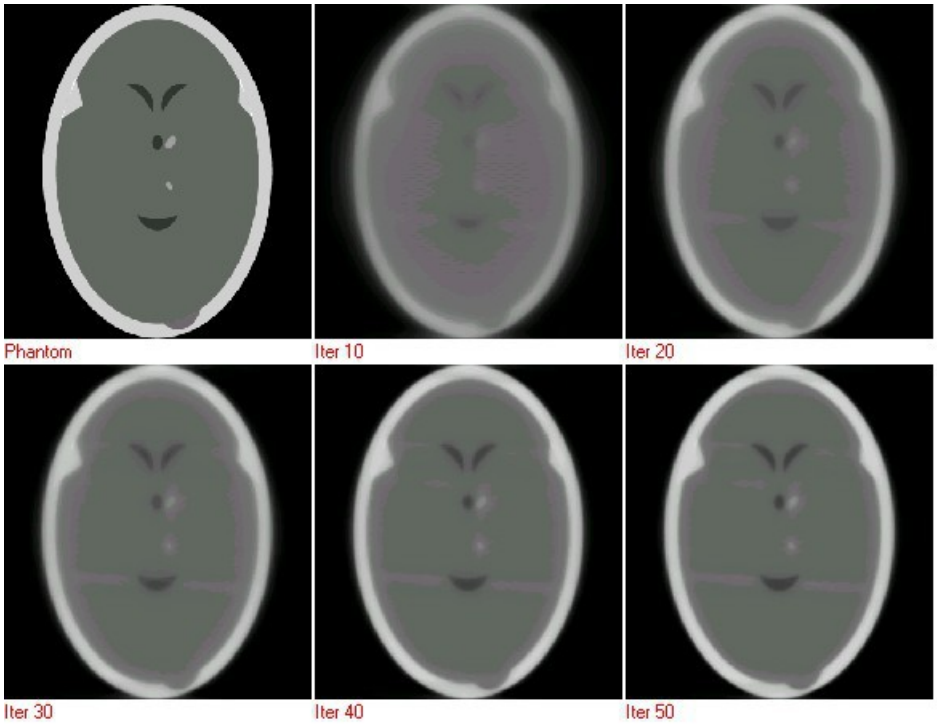

Figura 5.47. Imágenes producidas por CAV 2 en caso 2.4 (345×345 pixels, 232.275 ecuacs., 103.454 vars.)

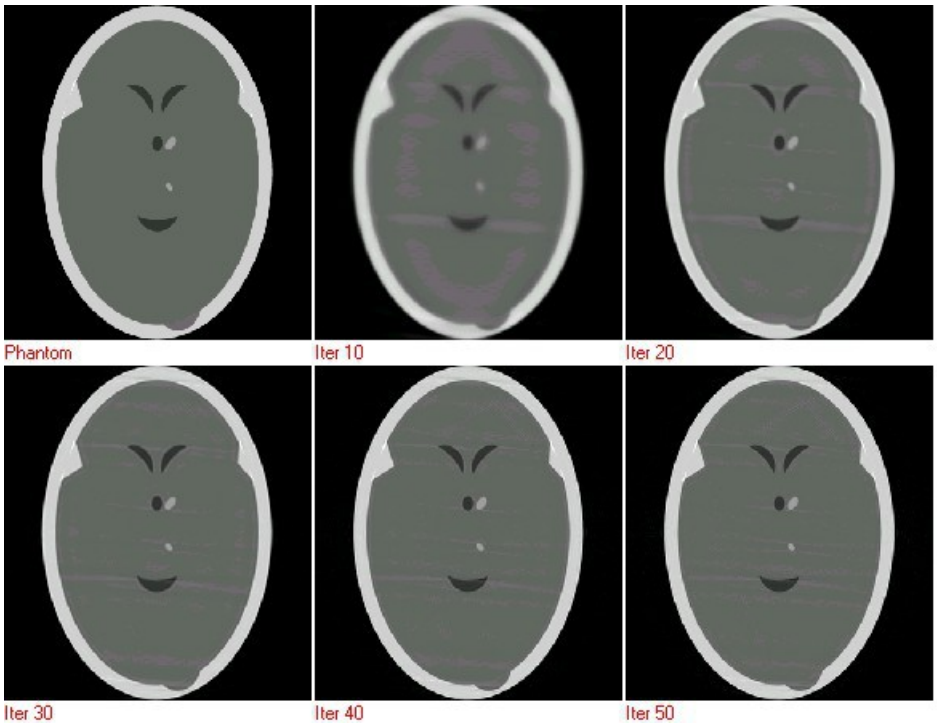

Figura 5.48. Imágenes producidas por ACCAV2 en caso 2.4 (345×345 pixels, 232.275 ecuacs., 103.454 vars.)

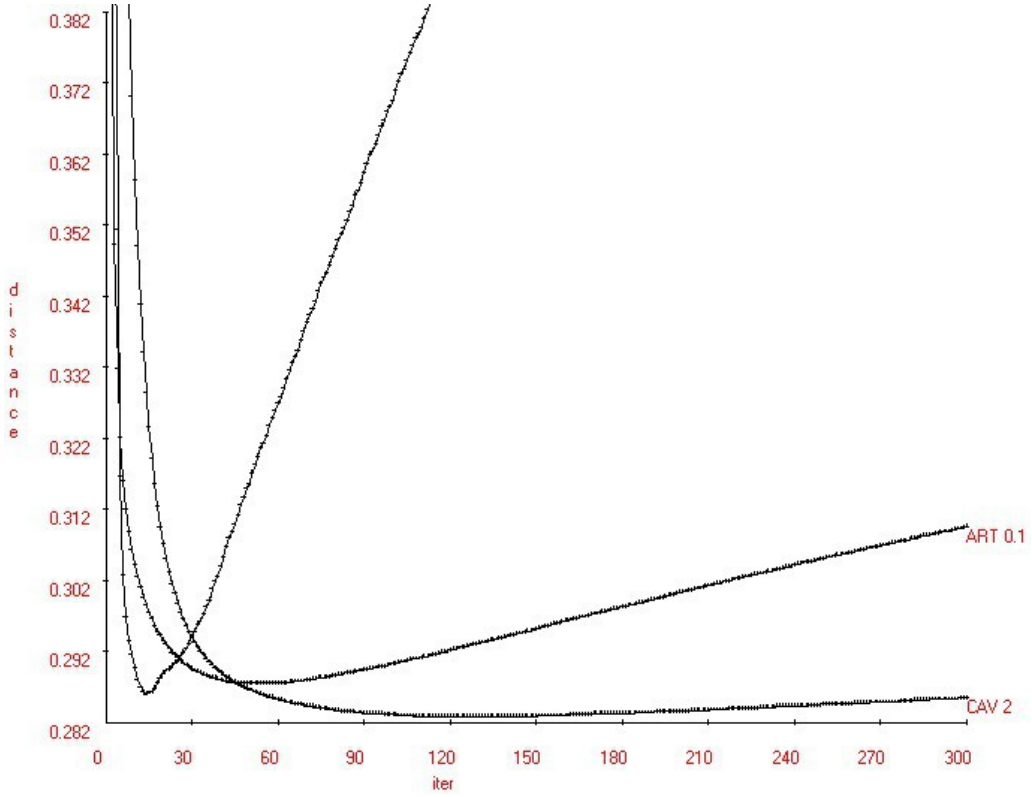

Figura 5.49. Distancia para caso 1.1 (300 iteraciones, 115 × 115 pixels, 13.137 ecuaciones, 13.225 vars.).

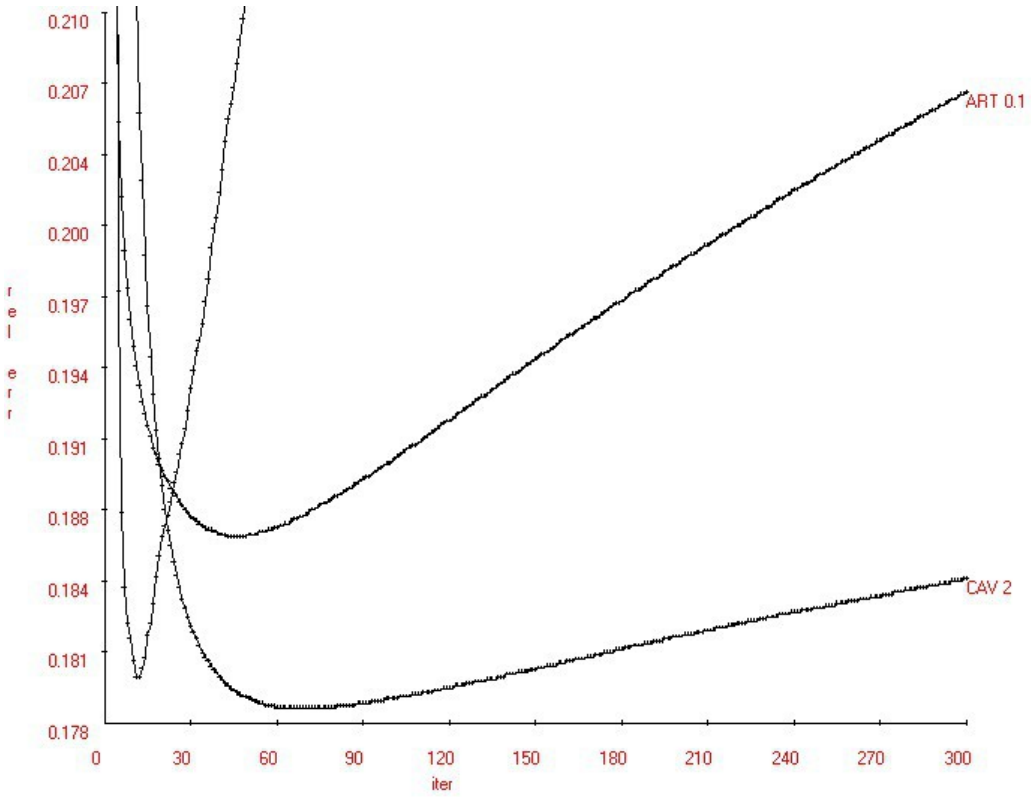

Figura 5.50. Error Relativo para caso 1.1 (300 iteraciones, 115 × 115 pixels, 13.137 ecuaciones, 13.225 vars.)

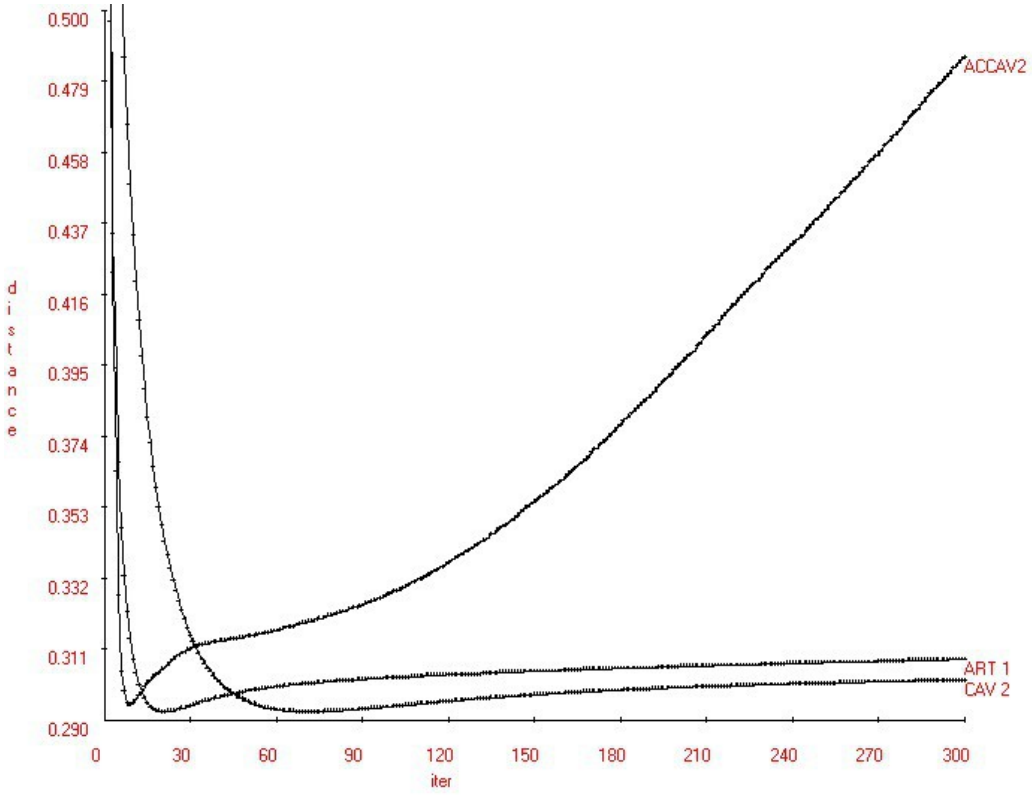

Figura 5.51. Distancia para caso 2.1 (300 iteraciones, 115 × 115 pixels, 13.137 ecuaciones, 11.600 variables.)

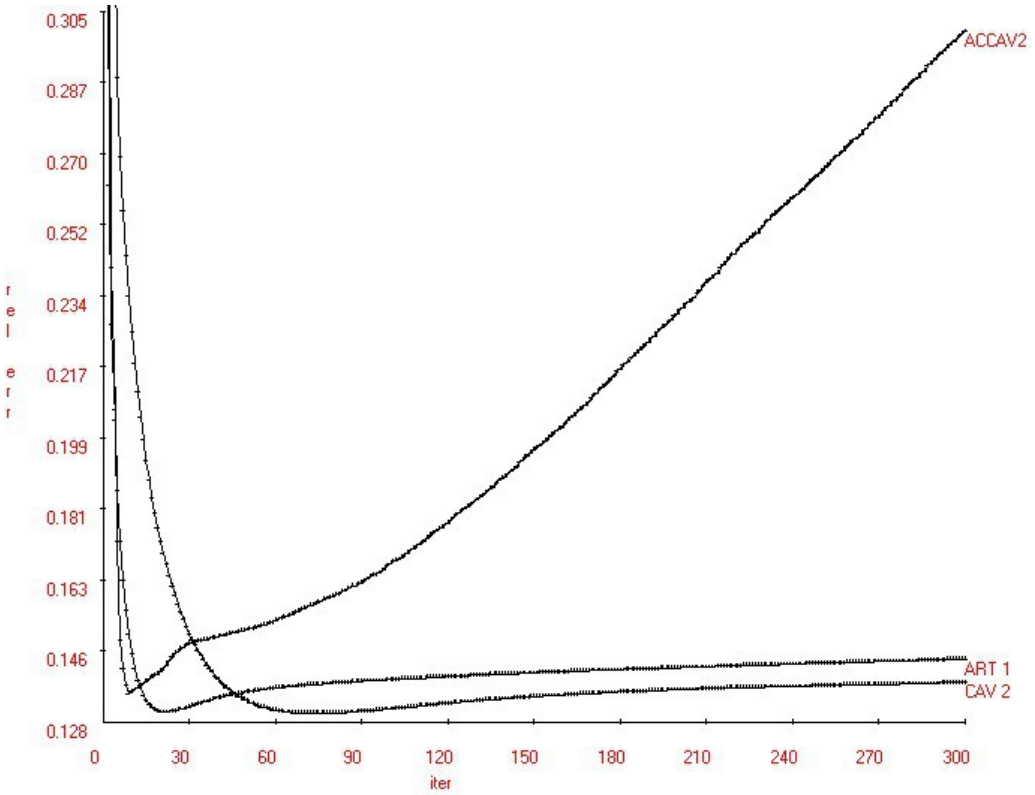

Figura 5.52. Error Relativo para caso 2.1 (300 iteraciones, 115 × 115 pixels, 13.137 ecuaciones, 11.600 vars.)

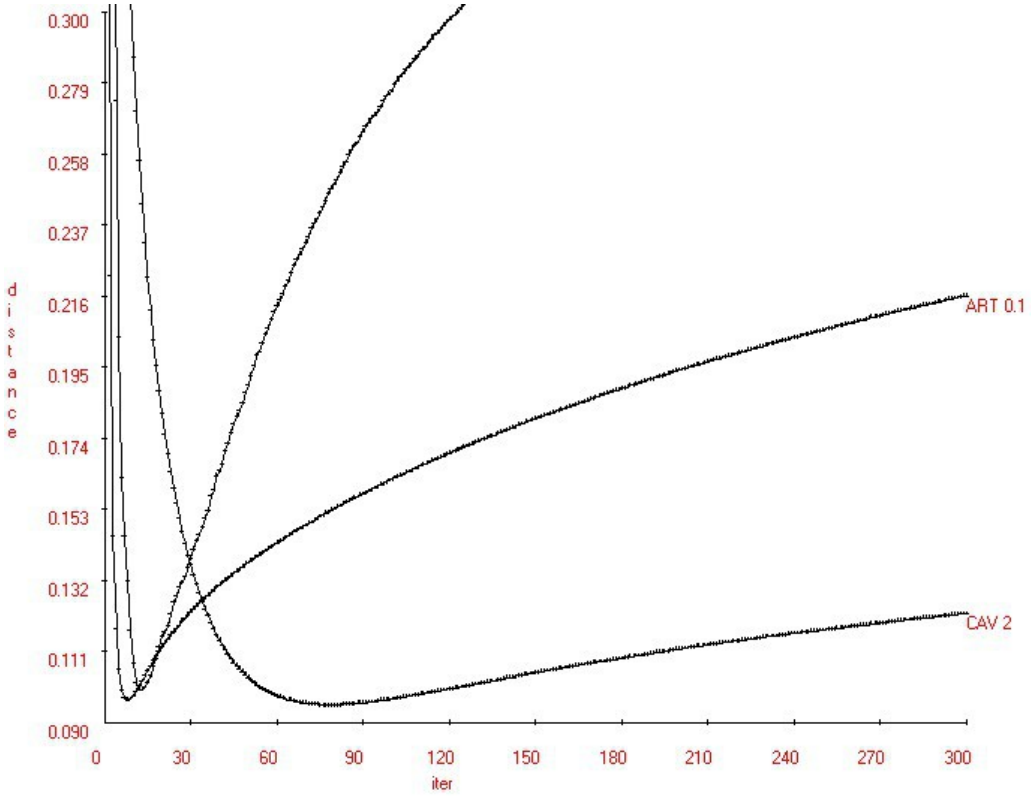

Figura 5.53. Distancia para caso 1.2 (300 iteraciones, 115 × 115 pixels, 26.425 ecuaciones, 13.225 variables.)

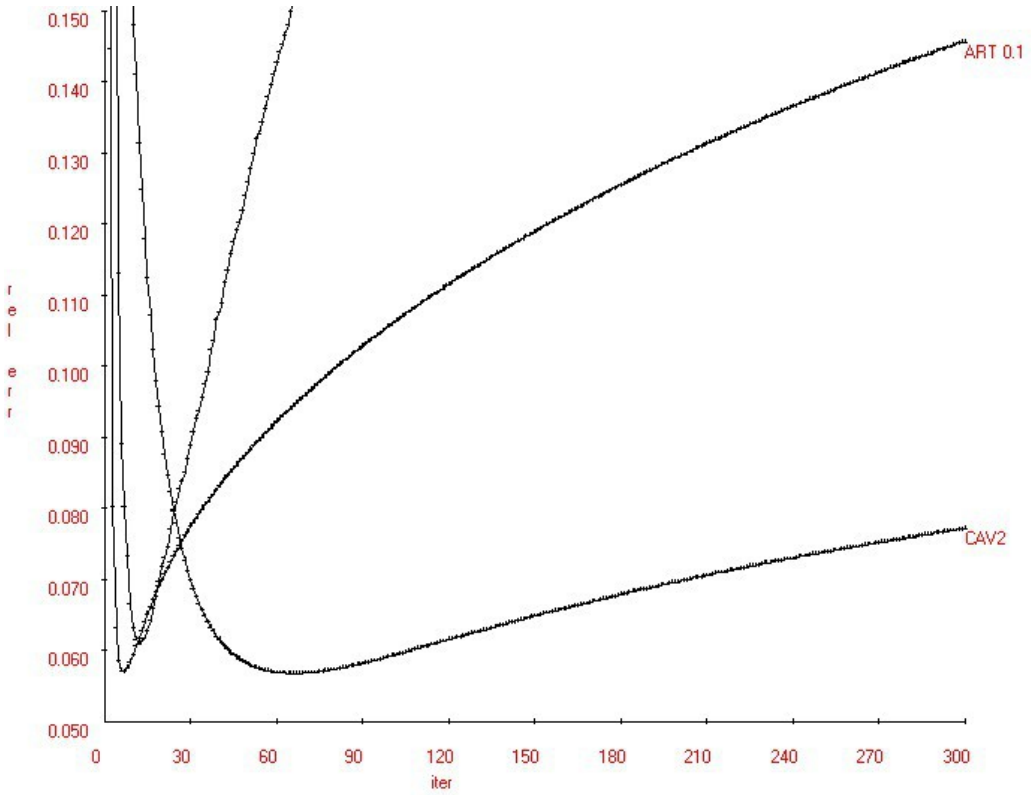

Figura 5.54. Error Relativo para caso 1.2 (300 iteraciones, 115 × 115 pixels, 26.425 ecuaciones, 13.225 vars.)

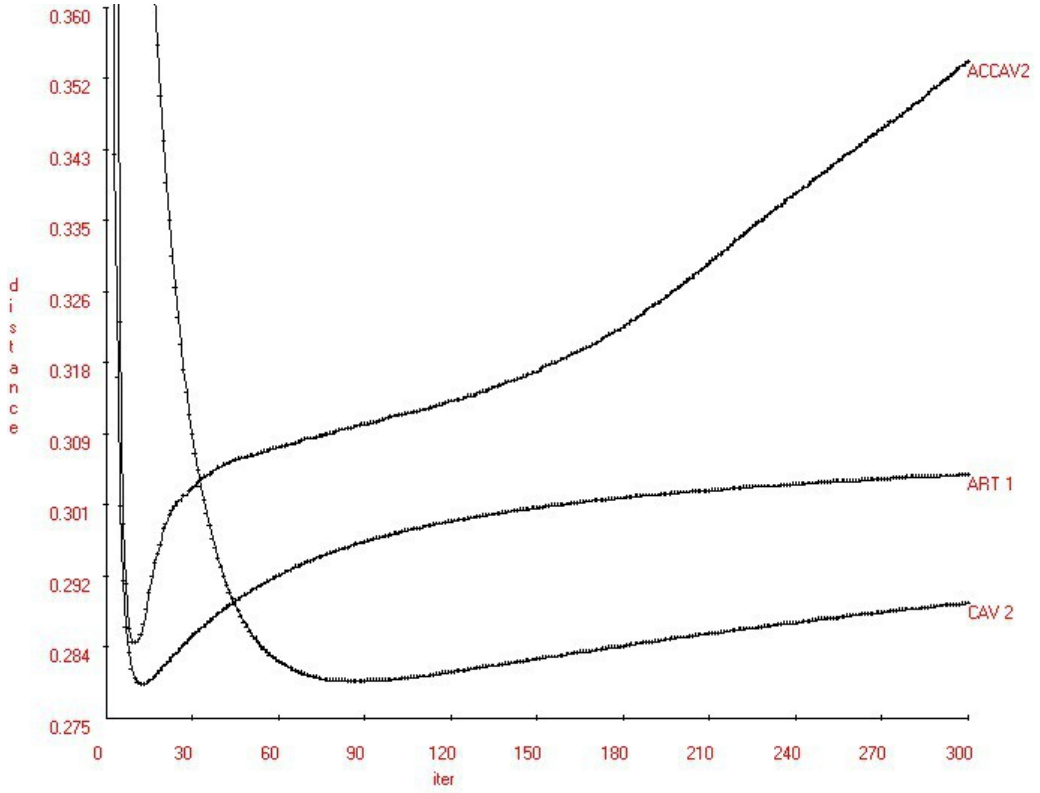

Figura 5.55. Distancia para caso 2.2 (300 iteraciones, 115 × 115 pixels, 26.425 ecuaciones, 11.600 variables.)

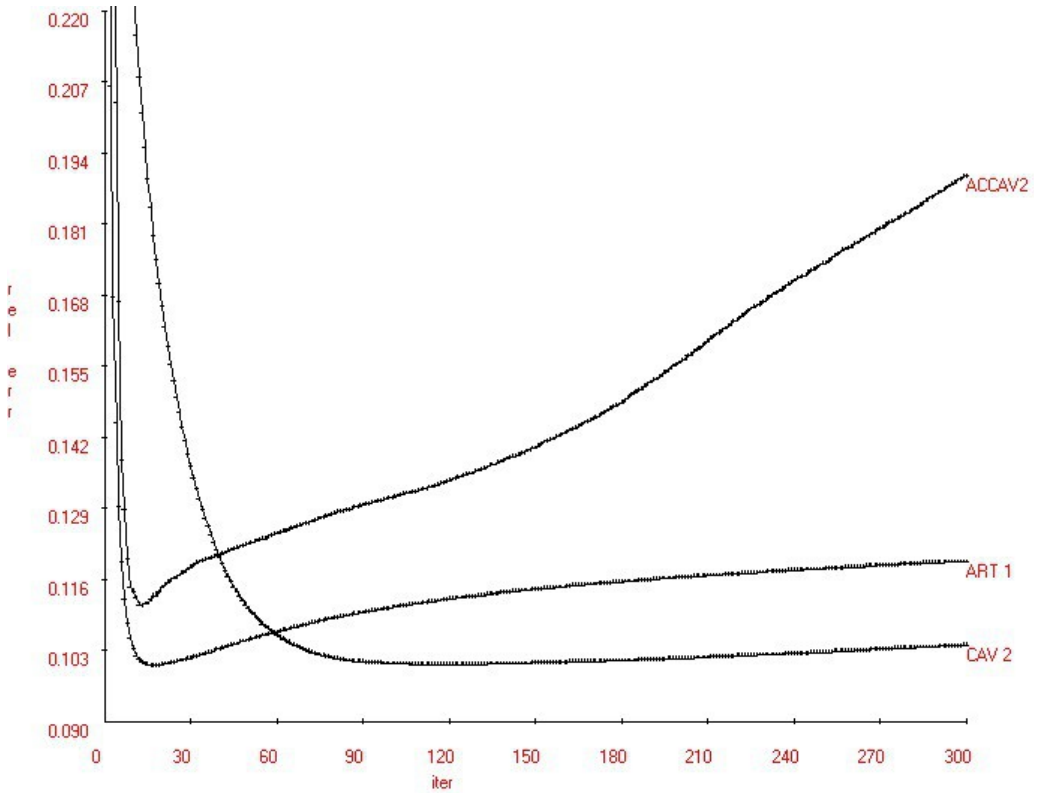

Figura 5.56. Error Relativo para caso 2.2 (300 iteraciones, 115 × 115 pixels, 26.425 ecuacs., 11.600 vars.)

Appendix: Extensions to SNARK93 for blobs support

- In a run of SNARK93 it can be used either blobs or pixels, but it is not possible in a single run to execute one algorithm using pixels and another using blobs. The reason for this implementation decision is the fact that SNARK93 was designed to create the RECFIL file at the beginning of a run, but such file structure had to be extended to support the storage of reconstructions achieved using blobs. The new structure is therefore able to *either* contain reconstructions using pixels *or* reconstructions using blobs, but not both at the same time. It is still possible to modify the RECFIL file structure to hold hybrid reconstructions, but if someone is interested in developing this improvement, he has to be careful in modifying all the routines that perform a search in such file.

  Another tip to consider in allowing hybrid reconstructions is the fact that certain arrays, such as RECON, LIST and WEIGHT presently have their space allocated depending of it is a pixel or blob reconstruction. The whole way of handling the decisions if certain tasks has to be performed in a blob way or a pixel way has to be changed, because presently it is being decided by checking at any moment the truth-value of the USING_BLOB flag. Some other considerations not listed here have also to be taken into account in making this change.

- The command to indicate that the reconstruction should be done using blobs is *BLOBS*. This command has to be placed in the Snark93 input file after the create input sequence commands (see Snark93 Manual 5.4) and before the *picture* and *projection* commands (see Snark93 Manual 5.5 – 5.6.) For its use in SuperSnark, this command has to be included in the DATA file (see Snark93 Manual 8.2.3) following the mentioned order. The syntax of this command is as follows:

  BLOBS ***a alfa gridType samplingDistance***

  where:

  a and *alfa* are the blob radius and taper parameters respectively. (The blobs used will be of order m=2.)

  gridType should be '**hexagonal**' because that is the only type of grid implemented so far. (To implement other types of grid the following subroutines of the module "blobs.f" should be modified accordingly: blob, getBlobIdxsForIntervalInRow, getBlobCenter, as well as the array gridTypeNames and the parameter gridTypesCount.)

  samplingDistance is the distance between blob centers in each grid row. (In the hexagonal grid case the even number rows have their first blob center at x = 0, and the odd number rows have their first blob center at x = samplingDistance / 2. The i-th row is at y = i * samplingDistance * sqrt(3) / 6.)

- When visualizing an image using blobs with SnarkDisplay, the density value of each pixel in the image being displayed is computed in the following way: Let *sF* be the scale factor chosen for visualization, then to each phantom pixel corresponds sF^2 pixels in the visual representation of the image. The density value p_j of the j-th pixel in the representation is computed as the following sum of the blob contributions at the center point of the pixel:

$$p_j = \sum_{k=1}^{B} c_k \cdot b\left(\left\| r_j - x_k \right\| / sF\right)$$

  where B is the number of blobs, c_k is the k-th blob coefficient, b(.) is the blob function, r_j is the j-th pixel center point, and x_k is the k-th blob center point.

  To compute the low, high and average density of the reconstructed image, the density used in this calculation for each "pixel" of the reconstruccion is computed as the average of the corresponding sF^2 pixels in the visual representation. The high, low and average density values shown in the Panel III prior to the showing of the picture are useless, because such values were computed using the blobs'

coefficients as if they were pixel coefficients. The right values will be computed and correctly displayed right after the image is shown.

- **Compiling Enviroments:**
  - SnarkDisplay / SnarkEval and SuperSnark were all compiled using MS Visual C++ 6.0
  - Snark93 was compiled using MS Fortran PowerStation 4.0

- **Notes:**
  - SuperSnark didn't need to be modified in order to support blobs.
  - ray.f was not implemented to support blobs.
  - If blobs are being used, then the function WRAY calls BWRAY (the version of WRAY for blobs.)
  - None of the built-in algorithms included in Snark93 were modified to support blobs. The user interested in running any of such algorithms using blobs has to modify its source code so that it uses BLOBS_COUNT as the dimension of the reconstruction image, (instead of AREA), besides performing any other changes that might be necessary.
  - The algorithm CAV was implemented for both pixels and blobs. Its name for execution via the EXECUTE command is "CAV" and its source code can be found in the module "cav.f". It requires the value of λ as a parameter.
  - The algorithm row-action ART was implemented for both pixels and blobs in the function Alg2 and its source code is in the module "alg2.f". It requires the value of λ as a parameter.
  - In the module "blobs.f" are located the implementation of the Kaiser-Bessel window functions and its integrals and most of the subroutines and functions needed for blobs support.
  - For SnarkDisplay and SnarkEval to be able to function, it is necessary the installation of the *MyGraphCONTROL.ocx* component via its setup program. It will install the required support DLL´s and will register the component (via a call to regsvr32). For best visualization using these tools, the screen resolution has to be set to 1024×768 pixels.

- **Common Block *BLOBS* variables´ description in SNARK93:**
  - BLOB_RADIUS is the blob a (radius) parameter,
  - BLOB_ALFA is the blob α (taper) parameter,
  - GRID_TYPE is the type of grid being used,
  - GRID_1ST_ROW_Y is the y coordinate of the first row of the grid,
  - GRID_ROW_DISTANCE is the distance in the y-axis between grid rows,
  - GRID_ROWS_COUNT is the number of rows in the grid,
  - GRID_SAMPLING_DISTANCE is the distance between two consecutive blob centers in a row of the grid,
  - BLOBS_PER_GRID_ROW is the number of blobs per grid row,
  - BLOBS_COUNT is the total number of blobs,
  - LINE_INTEGRALS_TABLE(1000) is an array containing the values of the line integral of the blob function for 1000 values of distance (ranging from 0 to a) of the line to the blob center,
  - HEXAGONAL_GRID_TYPE is an id for the hexagonal grid type,
  - USING_BLOBS is a logical flag indicating if blobs are being used for representing the images,
  - W2_TABLE(1000) is an array containing the values of the Kaiser Bessel window function w2 for 1000 values of distance (ranging from 0 to a) of the point to the blob center.

Bibliografía:

[BRO/93] J. A. Browne, G. T. Herman, D. Odhner. *SNARK93: A Programming System for Image Reconstruction from Projections*. Department of Radiology, University of Pennsylvania, Medical Image Processing Group, Technical Report MIPG198. 1993.

[CEN/83] Y. Censor. *Finite Series-Expansion Reconstruction Methods*. Proceedings of the IEEE, vol 71, pp. 409-419, March 1983.

[CEN/97] Y. Censor and S. A. Zenios. *Parallel Optimization: Theory, Algorithms, and Applications*, pp. 262-268. Oxford University Press, New York, USA. 1997.

[CEN/00] Y. Censor, Dan Gordon, and Rachel Gordon. *Component Averaging: An Efficient Iterative Parallel Algorithm for Large and Sparse Unstructured Problems*. Parallel Computing 27 (2001), pp. 777-808. 2000.

[EDH/87] P. Edholm y G.T. Herman. *Linograms in Image Reconstruction from Projections*. IEEE Trans. on Medical Imaging MI-6, pp 301-307. 1987.

[FUR/93] S. S. Furuie, G. T. Herman, T. K. Narayan, P. Kinahan, J. S. Karp, R. M. Lewitt y S. Matej. A *Methodology for Testitng for Statistically Significant Differences Between Truly 3-D PET Reconstruction Algorithms*. 1993 International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Snowbird, Utah, USA. 1993

[GAR/01] E. Garduño and G. T. Herman. *Optimization of Basis Functions for Both Reconstruction and Visualization*. Electronic Notes in Theoretical Computer Science 46. 2001.

[GIL/72] P. Gilbert. *Iterative Methods for the Three-Dimensional Reconstruction of an Object from Projection.*. J. Theor. Biol. 36, pp 105-117. 1972.

[GOR/74] R. Gordon y G. T. Herman. *Three Dimensional Reconstruction from Projections: a Review of Algorithms*. In International Review of Cytology. G. H. Bourne and J. F. Danielli, Eds. Academic Press, New York, pp 111-151. 1974.

[HER/76a] G. T. Herman y A. Lent. *Iterative Reconstruction Algorithms*. Comp. Biol. Med. 6, pp 273-294. 1976.

[HER/76b] G. T. Herman, A. V. Lakshminarayanan y A. Naparstek. *Convolution Reconstruction Techniques for Divergent Beams*. Comp. Biol. Med. 6, pp 259-271. 1976.

[HER/76c] G. T. Herman y A. Lent. *Quadratic Optimization for Image Reconstruction, Part I*. Comp. Graph. Image Proc. 5, pp 319-332. 1976.

[HER/78] G. T. Herman y S. W. Rowland. *SNARK77 – A programming System for Image Reconstruction from Projections*. Department of Computer Science, State University of New York at Buffalo, Amherst, N.Y. Technical Report No. 130. 1978.

[HER/80] G. T. Herman. *Image Reconstruction from Projections: The Fundamentals of Computerized Tomography*, Academic Press, New York, 1980.

[HER/89a] G. T. Herman y K. T. D. Yeung. *Evaluators of Image Reconstruction Algorithms*. Med. Image Processing Group, Dept. of Radiology, Univ. of Pennsylvania, Philadelphia, PA. Tech. Rep. No. MIPG151. 1989.

[HER/89b] G. T. Herman, R. M. Lewitt, D. Odhner y S. W. Rowland. *SNARK89 – A Programming System for Image Reconstruction from Projections*. Med. Image Processing Group, Dept. of Radiology, Univ. of Pennsylvania. Tech. Rep. No. MIPG160. 1989.

[HER/91] G. T. Herman y D. Odhner. *Performance Evaluation of an Iterative Image Reconstruction Algorithm por Positron Emission Tomography*. IEEE Trans. on Medical Imaging 10, pp 336-346. 1991.

[HER/01] G. T. Herman, S. Matej, and M. Carvalho. *Algrebraic Reconstruction Techniques Using Smooth Basis Functions for Helical Cone-Beam Tomography*. Inherently Parallel Algorithms in Feasibility and Optimization and their Applications, pp. 307-323. 2001.

[JAC/98] F. Jacobs, S. Matej, and R. Lewitt. *Image Reconstruction Techniques for PET*. Department of Electronics and Information Systems, University of Ghent, Medical Image and Signal Processing Group, technical report R9810 and Department of Radiology, University of Pennsilvania, Medical Image Processing Group, technical report MIPG245. 1998.

[JAC/99] F. Jacobs and I. Lemahieu. *Iterative Image Reconstruction From Projections Based On Generalised Kaiser-Bessel Window Functions*. 1st World Congress on Industrial Process Tomography, Buxton, Greater Manchester, April 14-17, 1999, p. 427. 1999.

[LEW/83] R. M. Lewitt. *Reconstruction Algorithms: Transform methods*. Proceedings of the IEEE, vol 71, pp. 390-408, March 1983.

[LEW/90] R. M. Lewitt. *Multidimensional Digital Image Representations Using Generalized Kaiser-Bessel Window Function*. J. Opt. Soc. Am., vol 7, pp. 1834-1846, October 1990.

[LEW/92] R. M. Lewitt. *Alternatives to Voxels for Image Representation in Iterative Reconstruction Algorithms*. Physics in Medicine and Biology, vol 37, pp 705-716. 1992.

[MAR/74] R. B. Marr. *On the Reconstruction of a Function on a Circular Domine from a Sampling of its Line Integrals*. J. Math. Anal. Appl. 45, pp 357-374. 1974.

[MAT/96a] S. Matej and J. A. Browne. *Performance of a Fast Maximum Likelihood Algorithm for Fully 3D PET Reconstruction*. Series of Computational Imaging and Vision: Three-dimensional Image Reconstruction in Radiology and Nuclear Medicine (P. Grangeat and J.-L. Amans, eds.), pp297-316, Kluwer Academic Publishers. 1996.

[MAT/96b] S. Matej and R. M. Lewitt. *Practical Considerations for 3-D Image Reconstruction Using Sperically Symmetric Volume Elements*. IEEE Transactions on Medical Imaging, vol 15, pp 68-78, February 1996.

[MOU/89] R. Mould. *Introduction to Medical Statistics*. Adam Hilger, Bristol, England. 1989.

[PER/75] R. M. Perry, M. D. Altschuler y B. R. Altschuler. *Medical Image Reconstruction: Multiangular Sectional Roengenography by Computer*. National Center for Atmospheric Research, Boulder, Colorado. Technical Note No. 108. 1975.

[SCO/00] H. Scolnik, N. Echebest, M. T. Guardarucci, M. C. Vacchino. *Acceleration Scheme for Parallel Projected Aggregation Methods for Solving Large linear Systems*. A aparecer en: Annals of Operations Research. Baltzer Science Publishers (Kluwer Academic Publishers Company). Entregado para su publicación en Noviembre 2000.

[SCO/01] H. Scolnik, N. Echebest, M. T. Guardarucci, M. C. Vacchino. *New Optimized and Accelerated PAM Methods for Solving Large Non-Symmetric Linear Systems: Theory and Practice.* En Inherently Parallel Algorithms in Feasibility and Optimization and their Applications, D. Butnariu, Y. Censor and S. Reich (Editors), Studies in Computational Mathematics, Volume 8, Elsevier Science, Amsterdam, April 2001.

[SCO/02] H. Scolnik, N. Echebest, M. T. Guardarucci, M. C. Vacchino. *A Class of Optimized Row Projection Methods for Solving Large Non-symmetric Linear Systems*. En Applied Numerical Mathematics, 41, pp 499-513, 2002.

[SHE/82] L. A. Shepp y Y. Vardi. *Maximum Likelihood Reconstruction in Positron Emission Tomography*. IEEE Trans. on Medical Imaging 6, pp 113-122. 1982.

[SMI/73] P. R. Smith, T. M. Peters y R. H. T Bates. *Image Reconstruction from Finite Numbers of Projections*. J. Phys. A: Math. Nucl. Gen. 6, pp 361-382. 1973.

[WAT/44] G. N. Watson. *Theory of Bessel Functions.* 2nd edition, Cambridge U. Press, Cambridge, 1944.