

# Métodos Numéricos

## Segundo Cuatrimestre de 2018

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

### Trabajo Práctico III

#### Tomografía computada

Integrante	LU	Correo electrónico
Teodoro Freund	526/15	tfreund95@gmail.com
Agustín Borgna	79/15	aborgna@dc.uba.ar
Jonathan Scherman	152/15	jscherman@dc.uba.ar
Jessica Singer	177/16	singer.jeess@gmail.com

En este trabajo nos proponemos implementar un programa basado en métodos numéricos conocidos para reconstruir imágenes tomográficas sujetas a ruido, utilizando aproximación por cuadrados mínimos. Para esto, vamos a generar y simular el comportamiento de los rayos utilizados por un tomografo en una imagen base, esto nos dará el resultado ideal que buscamos obtener, es decir, lo que un tomógrafo teórico (imposible) debería retornar. Agregando ruido vamos a simular un tomógrafo del mundo real, donde los resultados están sujetos a influencias externas. Luego, aproximaremos la imagen a través de cuadrados mínimos, esperando conseguir algo similar a lo que empezamos.

**SVD Decomposition    Linear Least Squares    PSNR    Tomography**

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Desarrollo</b>	<b>4</b>
2.1. Discretización del espacio . . . . .	4
2.2. Generación de Rayos . . . . .	4
2.2.1. Generadores de rayos . . . . .	4
2.2.2. Cálculo de incidencias . . . . .	6
2.3. Añadiendo Ruido . . . . .	7
2.4. Cuadrados Mínimos con SVD . . . . .	7
2.5. Condiciones para obtener una matriz SDP . . . . .	8
2.6. Estabilidad de la Solución . . . . .	9
<b>3. Resultados y Discusión</b>	<b>10</b>
3.1. Análisis de los rayos . . . . .	10
3.2. Variando el ruido añadido . . . . .	11
3.3. Variando la cantidad de valores singulares . . . . .	13
3.4. Experimentando con el número de condición . . . . .	15
3.5. Variando cantidad y tipo de rayos . . . . .	17
<b>4. Conclusiones</b>	<b>22</b>
<b>5. Apéndice</b>	<b>23</b>
5.1. Detalles de implementación . . . . .	23
<b>Referencias</b>	<b>24</b>

## 1. Introducción

La tomografía computada es una técnica ampliamente utilizada en la medicina a nivel mundial. En este rubro, se utiliza un tomógrafo que emite rayos X en distintas direcciones a través de un objeto y, utilizando esta técnica, permite reconstruir la densidad del objeto en cuestión calculando la velocidad de los rayos en distintos puntos de su trayectoria, a partir de los tiempos tomados por los rayos en viajar desde su emisor a su receptor.

En el contexto de este TP, nos vamos a centrar en reconstruir imágenes cuadradas usando esta metodología. De esta forma, al recibir nuestra imagen de entrada, lo primero que haremos será discretizarla en una matriz  $A \in \mathbb{R}^{n \times n}$ . Luego, generaremos los rayos a través de la imagen, para lo cual tenemos que definir una cantidad  $m$  de rayos y una forma de distribuirlos en el espacio discretizado  $A$ , y calcularemos el tiempo que demoran en atravesar la imagen ( $t_r$ ). Tenemos entonces que, para cada rayo  $r$ , el tiempo que toma en viajar desde su emisor a su receptor es:

$$t_r = \sum_{i=1}^n \sum_{j=1}^n \frac{\tilde{d}_{ij}}{v_{ij}}$$

donde  $\tilde{d}_{ij}$  corresponde a la distancia que recorre  $r$  en la celda  $A_{ij}$  y  $v_{ij}$  su velocidad. Notemos que  $t_r$  es un dato conocido y  $\tilde{d}_{ij}$  también, por lo que nuestra única incógnita resulta  $v_{ij}$ . De esta manera, generando varios rayos podemos construirnos una matriz  $D \in \mathbb{R}^{m \times n^2}$ , un  $s^t = (v_1^{-1}, v_2^{-1}, \dots, v_{n^2}^{-1})^t \in \mathbb{R}^{n^2}$  y un  $t^t = (t_1, t_2, \dots, t_{n^2})^t \in \mathbb{R}^{n^2}$  de forma tal que las ecuaciones anteriores correspondientes a los rayos resultan equivalentes a resolver el sistema:

$$\begin{pmatrix} - & - & D_1 & - & - \\ - & - & D_2 & - & - \\ & & \vdots & & \\ - & - & D_m & - & - \end{pmatrix} \begin{pmatrix} v_1^{-1} \\ v_2^{-1} \\ \vdots \\ v_{n^2}^{-1} \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_{n^2} \end{pmatrix}$$

donde cada  $D_i$  es la matriz de distancias  $\tilde{D}$  correspondiente al rayo  $i$ , visto como una única fila. Además, le agregaremos ruido a los tiempos para mayor verosimilitud con la realidad.

Luego, bastaría con resolver el sistema  $Ds = t$ , pero, dado que podría no tener solución, lo que buscamos es la solución  $s^*$  de cuadrados mínimos que cumple  $D^t D s^* = D^t t$ . Finalmente,  $s^*$  vista como una matriz de  $n \times n$  será la solución buscada.

## 2. Desarrollo

### 2.1. Discretización del espacio

Lo primero que tenemos que hacer es discretizar el objeto de entrada en una matriz  $A \in \mathbb{R}^{n \times n}$ . Para ello, como nuestro objeto en este caso es una imagen de  $k \times k$  ( $k \geq n$ ), le aplicamos un algoritmo de redimensionamiento.

### 2.2. Generación de Rayos

Simularemos los rayos a partir de una serie de generadores descritos en la sección 2.2.1. A partir de estos rayos, definidos como una recta en el plano, debemos calcular cuáles celdas atraviesan y en cuánto tiempo. Llamamos a esto *incidencia* de los rayos, y describimos el método utilizado en la sección 2.2.2.

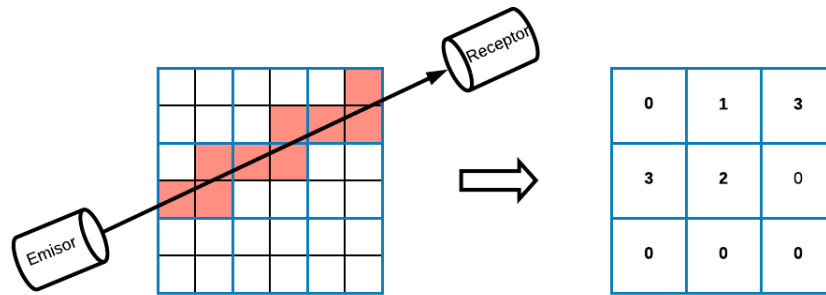


Figura 1: Generando matriz de distancias asociada a un rayo.

#### 2.2.1. Generadores de rayos

**Rayos axiales.** Los rayos axiales nos permitieron comenzar a entender el sistema, y ver distintos modos de avanzar con la implementación. La idea es que cada rayo va a recorrer una fila (o una columna) de punta a punta, pasando sólo por las celdas de esa fila (o columna) de forma recta (sin inclinación). La Figura 2 muestra una posible instanciación de estos rayos.

Estos rayos tienen como ventaja una implementación muy simple, de todas formas, cómo se vera con la experimentación, no son lo suficientemente buenos cómo para reconstruir la imagen, de hecho, nos damos cuenta de que estos rayos solo pueden generar  $2n$  rectas distintas, con las cuales queremos representar una imagen que tiene, informalmente,  $n^2$  cantidad de información. Si bien podemos repetir muchas veces cada rayo para tener un sistema sobredimensionado, muchas de las ecuaciones serán iguales.

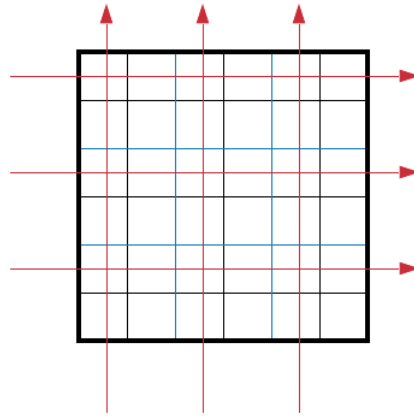


Figura 2

**Rayos laterales.** El problema planteado con los rayos axiales fue la falta de cantidad de rayos distintos que podíamos generar, por lo que una respuesta obvia fue agregarle otra variable a los rayos, una pendiente. La idea principal de los rayos laterales es generar una gran cantidad de rayos distintos, sin perder cierta simpleza en su implementación. Nuestros rayos van a ser rectas que cortan la imagen entrando por uno de sus lados y saliendo por el otro. Para generar  $K^2$  rayos, vamos a tomar  $K$  puntos equidistantes entre vecinos sobre el margen izquierdo y  $K$  sobre el derecho, y vamos a lanzar un rayo desde cada uno de los del margen izquierdo, hacia cada uno de los del derecho.

Rápidamente, se ve que tenemos una enorme cantidad de rectas potenciales distintas, por lo que intuitivamente, tenemos la suficiente cantidad de ecuaciones independientes en nuestro sistema para capturar a toda la imagen. De todas formas, cómo ya se puede ver un poco en la Figura 3, hay zonas que son atravesadas por menos rectas que otras. Esto lo vamos a ver con mayor profundidad cuando experimentemos.

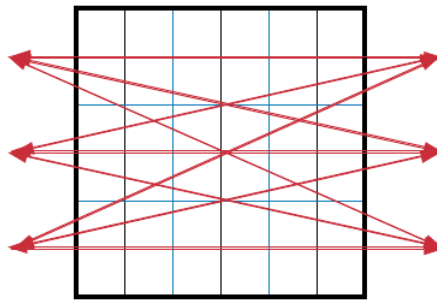


Figura 3

**Rayos aleatorios.** Otra opción que nos planteamos fue usar rayos aleatorios, cómo uno espera, la generación aleatoria no sigue un patrón tan claro, pero cuantos más rayos uno genere, más la imagen abarcará, minimizando las zonas que tienen poco contacto con rayos en nuestros modelos previos. En la experimentación veremos que este modelo obtiene buenos resultados.

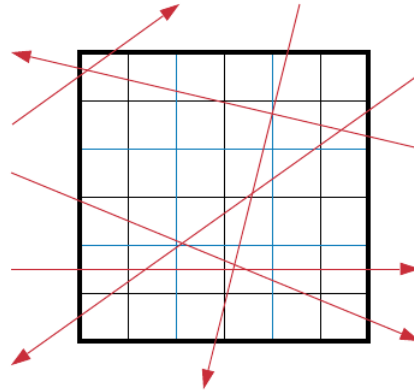


Figura 4

### 2.2.2. Cálculo de incidencias

Para decidir en qué celdas incide cada rayo y cuanto tiempo tarda en atravesarlas utilizamos el algoritmo de rasterizado de líneas con anti-aliasing de Xiaolin Wu [2].

Este genera para cada celda de la imagen un valor de punto flotante entre 0 y 1 que representa la distancia que atraviesa el rayo en la imagen. Un ejemplo se muestra en la figura 5.

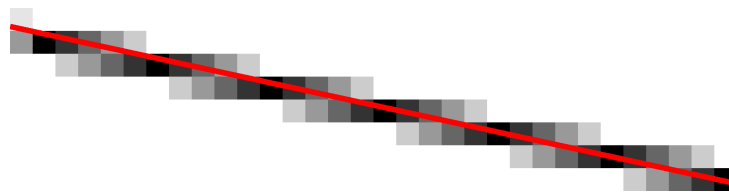


Figura 5: Ejemplo de incidencias de un rayo sobre celdas utilizando el algoritmo de Xiaolin Wu. Colores mas oscuros representan un mayor tiempo incurrido.

Un método similar sería utilizar el algoritmo de Bresenham. Al contrario que en el algoritmo anterior, este devuelve valores binarios para cada celda indicando si esta es atravesada por el rayo, como se puede ver en el ejemplo de la figura 6.

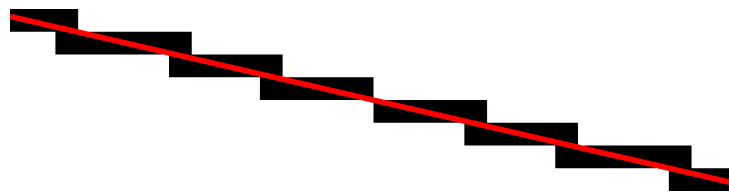


Figura 6: Ejemplo incidencias de un rayo sobre celdas utilizando el algoritmo de Bresenham. Colores mas oscuros representan un mayor tiempo incurrido.

Utilizar un algoritmo que genera en valores reales resulta en una simulación mas realista de los rayos, ya que es una representación mas fiel del tiempo que incurren en atravesar el área demarcada por una celda.

Adicionalmente esto puede aumentar el rango de la matriz de distancias de rayos, ya que el resultado del algoritmo de Xiaoling Wu es mas sensible a perturbaciones en la posición del rayo y por lo tanto rayos similares tienen mas probabilidad de resultar en ecuaciones linealmente independientes.

### 2.3. Añadiendo Ruido

Cómo se pedía en el enunciado, en este trabajo nosotros generamos la respuesta esperada del sistema y le agregamos ruido, para esto planteamos una función de ruido que le agrega a cada valor del vector resultado del sistema un número aleatorio que sigue una distribución normal, donde variamos la desviación estándar para experimentar. Además, agregamos la opción de no agregar ningún tipo de ruido, esto nos permitió evitar el ruido a la hora de implementar la solución.

### 2.4. Cuadrados Mínimos con SVD

Para poder obtener las velocidades de los rayos, debemos resolver el sistema  $D.s = t$ , donde, recordemos,  $D \in \mathbb{R}^{m \times n^2}$  la matriz cuyas filas se corresponden con las  $m$  señales, las columnas se corresponden con las  $n^2$  celdas de la discretización del cuerpo de forma vectorizada, y en cada celda contiene  $d_{k,(i,j)}$ , es decir, la distancia del  $k$ -ésimo rayo en la celda  $(i,j)$ . El vector  $t$  corresponde a los tiempos para cada rayo, y el vector  $s$  nos da entonces la inversa de las velocidades originales en cada celda que representarán las densidades en cada punto del cuerpo escaneado.

Al tener muchas mediciones, es probable que el sistema esté sobredeterminado, y la presencia de errores de medición hace que no sea factible buscar una solución exacta al sistema. Es por esto que buscaremos la solución de *cuadrados mínimos* del mismo.

En este caso, utilizamos la descomposición SVD para facilitarnos los cálculos. La idea fue la siguiente: Inicialmente tenemos la matriz  $D$ , la cual podemos descomponer en  $U.\Sigma.V^t$ , las de la descomposición SVD.

Para esto, debemos buscar los autovalores y autovectores de  $D^t.D \in \mathbb{R}^{m \times m}$  utilizando el *método de la potencia*.

Sin embargo, es probable que, al ser  $A$  una matriz *rala*, la matriz  $D^t.D$  tenga varios autovalores nulos, y el método de la potencia funciona cuando los autovalores tienen un orden estricto.

Más aún, podemos evitarnos estos cálculos si utilizamos la descomposición SVD comprimida <sup>1</sup>. Por lo tanto tendremos que  $D = \tilde{U}.\tilde{\Sigma}.\tilde{V}^t$ , donde  $\tilde{\Sigma} \in \mathbb{R}^{r \times r}$  es una matriz diagonal con los valores singulares  $(\sigma_1, \dots, \sigma_r)$  de mayor a menor en la diagonal, la  $\tilde{V} \in \mathbb{R}^{n^2 \times r}$  es una matriz con los autovectores asociados a cada autovalor en sus columnas, y  $\tilde{U} \in \mathbb{R}^{m \times r}$  son las primeras  $r$  columnas de la  $U$  de la factorización SVD.

De esta forma logramos reducir bastante el costo computacional que lleva calcular autovectores y autovalores, ya que calculamos menos, y el hecho de utilizar la descomposición SVD hace que los cálculos los hagamos sobre  $D^t.D$  la cual *es probable* que tenga menores dimensiones que  $D$ .

Ahora bien, ¿Qué haremos con la descomposición una vez que la tengamos?, ¿Cómo relacionamos esto con cuadrados mínimos? La idea va a ser despejar la *ecuación normal* dada por cuadrados mínimos sin llegar a resolver ese sistema explícitamente. Para ello, seguimos el siguiente razonamiento:

Inicialmente, tenemos la ecuación normal:

$$D^t.Ds = D^t.t \quad (1)$$

Reemplazando por nuestra descomposición SVD comprimida, tenemos:

$$(\tilde{V}.\tilde{\Sigma}.\tilde{U}^t).(\tilde{U}.\tilde{\Sigma}.\tilde{V}^t)s = (\tilde{V}.\tilde{\Sigma}.\tilde{U}^t).t \quad (2)$$

Como  $\tilde{U}$  es ortogonal, nos queda

$$(\tilde{V}.\tilde{\Sigma}^2.\tilde{V}^t)s = (\tilde{V}.\tilde{\Sigma}.\tilde{U}^t).t \quad (3)$$

Ahora notemos que  $\tilde{\Sigma}$  es una matriz diagonal con elementos **positivos** en la diagonal, ya que son los valores singulares. Por lo tanto, su inversa será una matriz diagonal con los mismos elementos *invertidos*,

<sup>1</sup><http://sepwww.stanford.edu/public/docs/sep73/ray1/paper.html/node3.html>

es decir, en lugar de tener a los  $\sigma_i$  en la diagonal, tendrá a  $\frac{1}{\sigma_i}$ ,  $i = 1 \dots r$ .

Tendremos entonces, multiplicando por  $\tilde{V}\tilde{\Sigma}^{-2}\tilde{V}^t$  a izquierda, lo siguiente:

$$s = (\tilde{V}\tilde{\Sigma}^{-2}\tilde{V}^t)(\tilde{V}.\tilde{\Sigma}.\tilde{U}^t).t = (\tilde{V}\tilde{\Sigma}^{-1}.\tilde{U}^t).t \quad (4)$$

Y con estos datos podremos conseguir el vector  $s$  y resolver el sistema.

Nuestro algoritmo hará exactamente eso, y de esta forma conseguiremos una aproximación de las velocidades de los rayos. El pseudocódigo del algoritmo es el siguiente:

---

**Algorithm 1** LSQ With SVD
 

---

```

1: function LSQWITHSVD(Matrix D, vector t)
2:   (U, Σ, V) ← SVDComprimida(D)
3:   Σ-1 ← inversaDiagonalPositiva(Σ)
4:   s ← V.Σ-1.Ut.t
5:   return s
6: end function

```

---

Notemos que la inversa de la  $\Sigma$  se calcula linealmente, ya que es hacer 1 dividido los valores de la diagonal de ella misma.

Para obtener la descomposición de SVD comprimida, tuvimos que obtener los autovalores y autovectores de  $D^t.D$ . Para esto, utilizamos el método de la potencia con deflación. Este método plantea aplicar alguna matriz  $A$   $k$  veces a algún vector aleatorio  $v$ , partiendo de la presunción de que este vector no es ortogonal al autovector de autovalor mayor, se consigue que, cuando  $k \rightarrow \infty$ ,  $A^k v \rightarrow e$ , donde  $e$  es el autovector buscado, normalizado. Es decir, que  $Ae = \lambda e$ , donde  $\lambda$  es el autovalor asociado. Luego, planteamos la matriz  $A^{(1)} = A - \lambda e e^t$ , y notamos que esta nueva matriz tiene todos los autovectores que  $A$ , con los mismos autovalores, excepto que ahora  $e$  tiene a 0 como su autovalor asociado, y repetimos sobre  $A^{(1)}$ .

$$A^{(1)}e = (A - \lambda e e^t)e = Ae - \lambda e(e^t e) = \lambda e - \lambda e = 0e$$

$$A^{(1)}w = (A - \lambda e e^t)w = Aw - \lambda e(e^t w) = \epsilon w - \lambda e 0 = \epsilon w$$

donde  $w$  es el autovector asociado al autovalor  $\epsilon$ , y  $w$  es ortogonal a  $e$ .

## 2.5. Condiciones para obtener una matriz SDP

Para hallar las velocidades de los rayos, debemos resolver el mencionado sistema de ecuaciones lineales  $D.s = t$ . Este sistema tiene un significado *físico* además de uno matemático; estamos despejando la inversa de las velocidades de los rayos.

Estas velocidades deberían existir y ser únicas, por supuesto. Esto se traduce en que el sistema debería tener solución única.

Sin embargo, como vimos, los errores de precisión pueden traer como consecuencia que el sistema no tenga solución, y por esto es que se utiliza la aproximación por cuadrados mínimos de la solución del sistema.

Ahora bien, ¿podría pasar que el sistema tenga más de una solución utilizando cuadrados mínimos? La respuesta es que sí.

Como sabemos, utilizando cuadrados mínimos la solución es única si y sólo si la matriz, en este caso  $D$ , tiene rango máximo (ya que esto implica directamente que  $D^t.D$  es inversible). Notemos que en ese



caso  $D^t \cdot D$  va a ser, más aún, simétrica definida positiva.

Esto es puesto que  $D^t \cdot D$  es obviamente simétrica, y por el *Teorema de la Dimensión* el núcleo de una matriz de rango máximo es el subespacio nulo. Esto trae como resultado que, tomando  $x \in \mathbb{R}^{n^2}$  cualquiera,

$$x^t \cdot D^t \cdot D \cdot x = (x^t \cdot D^t) \cdot (D \cdot x) = \|D \cdot x\|_2^2 \neq 0 \quad \forall x \neq 0, \text{ ya que } \text{Nu}(D) = \{0\}$$

Sin embargo, nada nos asegura que en nuestro sistema  $D$  tenga rango máximo, esto dependerá de los valores que tomen los rayos, de la forma en que los mandemos y de que los errores numéricos no estropeen las cuentas.

Como sabemos, cada celda  $(i, (j,k))$  de  $D$  corresponde a la distancia que recorre el rayo  $i$ -ésimo en la celda  $(j,k)$  de la discretización.

Nuestras matrices por regla general tendrán más de  $n^2$  rayos, para poder capturar correctamente la imagen. Por lo tanto, que  $D$  sea de rango máximo indicaría que su rango es  $n^2$ . Para esto, sus *columnas* deben ser linealmente independientes, lo cual, gráficamente, se puede pensar como que no hayan dos posiciones de la imagen tales que los distintos rayos hayan recorrido, en ellas, la misma distancia. Por último, si bien en la teoría esto tiene sentido, en nuestro caso contamos con mediciones que conllevan cierto error (en nuestro caso, al estar simulando, añadimos el error artificialmente). El ruido añadido a las mediciones hace que estas definiciones sean imprecisas, ya que existirá siempre un error inherente al realizar los cálculos.

## 2.6. Estabilidad de la Solución

En nuestro problema de hallar las velocidades de los rayos, debemos tener en cuenta que nos manejamos con una aritmética finita y por ende vamos a tener irremediablemente errores de precisión.

En base a esto, podemos, además de ver experimentalmente los resultados obtenidos y medir los errores con alguna medida estándar como PSNR, adelantarnos y ver qué *número de condición* tiene nuestra matriz  $D$ , para poder medir la estabilidad de nuestra solución obtenida y el impacto de los errores de cálculo.

Por la naturaleza del número de condición <sup>2</sup> de una matriz dada, éste se puede expresar como el mayor valor singular,  $\sigma_1$ , dividido el menor valor singular,  $\sigma_r$ . Ahora bien, en nuestra matriz  $D$ , ¿Qué significan los valores singulares? Nuestra matriz refleja la información captada por los rayos; en este sentido, mayor cantidad de valores singulares implica una mayor *cantidad* de información captada. Si tenemos pocos rayos o éstos no están bien distribuidos, podemos pensar que captaremos sólo algunas partes de la imagen que no alcanzarán para reconstruirla de forma eficaz, y veremos experimentalmente que esto estará relacionado con la cantidad de valores singulares que encontramos en la matriz  $D$ .

Siguiendo esa línea, si captamos un mayor rango de valores singulares de la matriz, estaremos de cierta manera complejizando o mejorando la información captada por los rayos. Esto nos plantea un *trade off* entre número de condición y cantidad de información de los rayos.

Pensemos lo siguiente: si no tuviéramos ningún rayo, la matriz  $D$  sería la matriz nula. La estabilidad de una solución hallada en un sistema con la matriz  $D$  sería genial, sin embargo, no estaríamos capturando ninguna imagen. A medida que arrojamamos más rayos y mejores distribuidos, el espectro de valores singulares tendría sentido que aumente, y por lo tanto el número de condición, calculado en base a estos, aumentaría también, dando como resultado un sistema poco estable, pero pareciera ser que es el "*precio*" que hay que pagar para obtener información de la imagen. Utilizamos este disparador para experimentar sobre la calidad de la imagen con distintos rayos y el número de condición obtenido en cada caso.

<sup>2</sup> Belsley, David A.; Kuh, Edwin; Welsch, Roy E. (1980). "The Condition Number". Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. New York: John Wiley Sons.

### 3. Resultados y Discusión

#### 3.1. Análisis de los rayos

En esta sección analizaremos, visualmente, a los distintos rayos y al cubrimiento de la imagen que logran. Para eso, las Figuras 7, 8 y 9 muestran heatmaps donde, cuanto más blanco está algún pixel, más veces pasan rayos por ese punto. Además, en rojo, se ven los rayos instanciados.

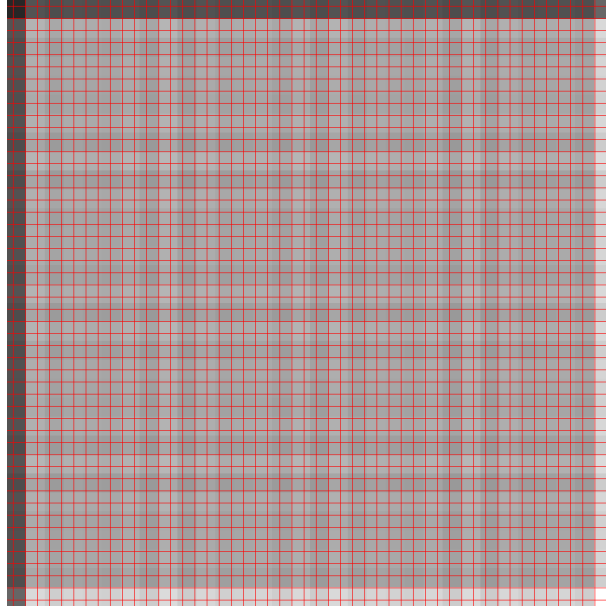


Figura 7: Densidad de 100 rayos axiales, cuanto más negro, menos representa estos rayos ese pixel.

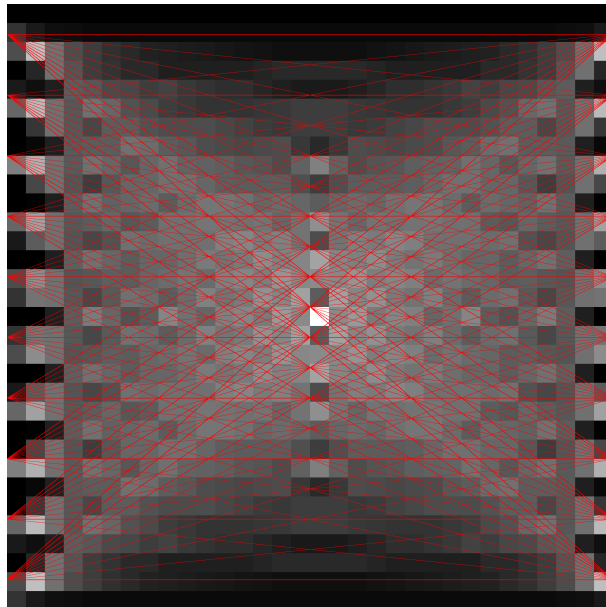


Figura 8: Densidad de 100 rayos laterales, cuanto más negro, menos representa estos rayos ese pixel.

Para comparar, vamos a notar dos cosas, la primera, que la densidad en los pixeles de los rayos axiales (por lo menos para 100 rayos) es mucho más distribuida que para los otros rayos, que tiene algunos puntos muy representados, y otros con pocos rayos incidentes. En la sección 3.4, vemos que los rayos axiales presentan un bajo número de condición, la razón puede estar relacionada a esta distribución pareja.

Además, notamos, en la Figura 8, que hay secciones de la imagen que están completamente negras, cómo por ejemplo los márgenes superior e inferior, luego notaremos que, cuando reconstruyamos las imágenes con este método, dichas partes no serán fieles y tendrán mucho error, que será inherente a la cantidad de rayos que utilizemos.

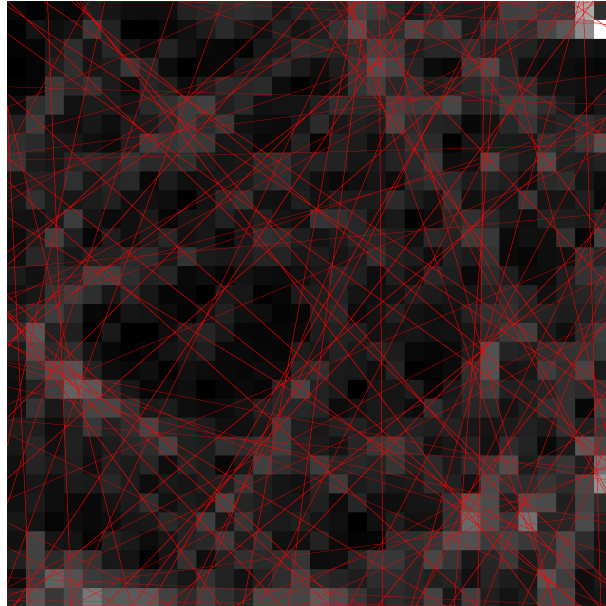


Figura 9: Densidad de 100 rayos random, cuanto más negro, menos representa estos rayos ese pixel.

### 3.2. Variando el ruido añadido

Como vimos, al simular los rayos del tomógrafo agregamos un error, o ruido, para condecirnos con la realidad de la inexactitud de las mediciones. Esto trae aparejado un error de cálculos inherente a los algoritmos y métodos numéricos que utilizemos. En esta sección experimentaremos variando el ruido y veremos cómo afecta a nuestros resultados.

En el siguiente experimento (figura 10) variamos, para la imagen `tomo3.png`, la cantidad de rayos enviados, todos de tipo *random*, y la cantidad de ruido midiendo la misma como el desvío estándar (es decir, aumentamos el desvío estándar de la variable aleatoria agregada al vector como se describe en el Desarrollo de este trabajo). En cada iteración, medimos el valor del error del resultado utilizando el PSNR.



Figura 10

Podemos ver que existe una clara relación entre el aumento del error y una peor calidad de imagen, aunque a medida que aumentamos la cantidad de rayos el error pareciera afectar cada vez menos el resultado final.

En el siguiente experimento (figura 11) medimos más detalladamente para 6000 rayos aleatorios, la variación del PSNR con el error.

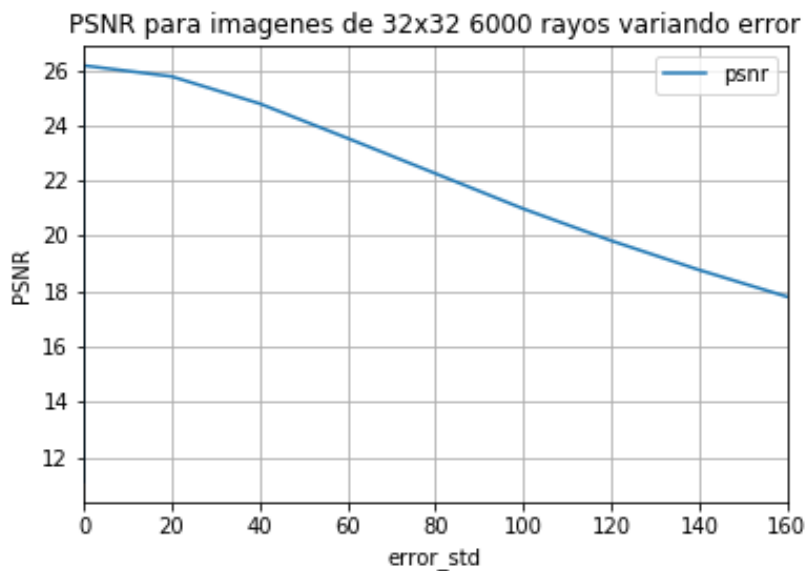



Figura 11

Al igual que antes, vemos que parecería haber una relación lineal entre el aumento del error y la calidad de la imagen, aunque en valores muy pequeños de error no parecería afectar demasiado.

Por último, en el siguiente cuadro incluimos algunas imágenes del resultado con distintos niveles de ruido. Se puede ver cómo al principio, con un error añadido con desvío estándar de 40, no parece afectar mucho a la imagen, y a medida que aumenta se hace más clara la presencia de ruido, aunque nunca deja de ser reconocible la imagen, como sí veremos en otros experimentos.

Cuadro 1: Imágenes según el desvío estándar del error

Error std	Tipo de Rayos	Resultado
0	Random	
40	Random	
100	Random	
160	Random	

### 3.3. Variando la cantidad de valores singulares

En esta sección experimentaremos variando la cantidad de valores singulares que tomamos. La idea de esto es que quizás, autovalores pequeños aportan más ruido que información al sistema, y perderlos puede llegar a aportar claridad a la imagen generada.

En nuestro algoritmo, tenemos que buscar los autovalores *positivos* de  $D^t.D$ , para obtener los valores singulares y poder hacer la descomposición SVD comprimida de  $D$ . Esto lo hacemos utilizando el método de la potencia, que va retornando los autovalores de mayor a menor.

Por cuestiones relativas a la aritmética finita de las computadoras, no podemos seguir buscando autovalores hasta que alguno de 0, sino que compararemos que éstos sean mayores a algún  $\alpha$  que tendremos parametrizado.

Una cuestión a tener en cuenta es que aumentar el  $\alpha$  produce menos iteraciones del método de la potencia, disminuyendo en algunos casos considerablemente el tiempo de cómputo, con lo cual estaríamos interesados en encontrar el  $\alpha$  más alto posible que produzca resultados *razonables*.

En el siguiente experimento (figura 12), tomamos la imagen `tomo3.png` en 32x32px, y corrimos el algoritmo con 4000 rayos aleatorios. Variamos el  $\alpha$  desde 0 hasta 30, de a 1. Para cada imagen generada, medimos el error con la original utilizando el PSNR.

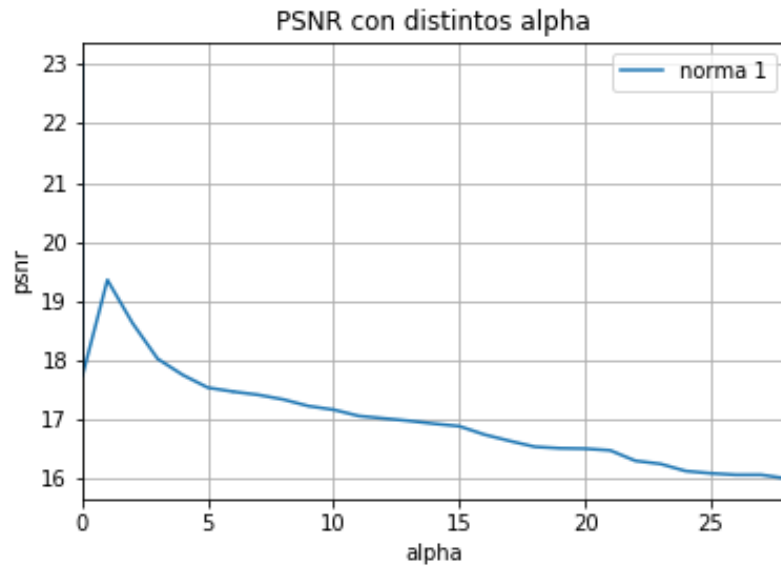
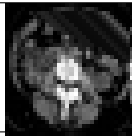
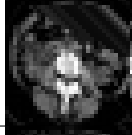
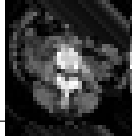
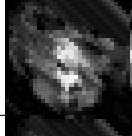
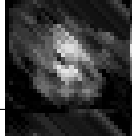
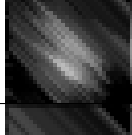
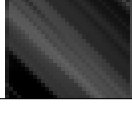


Figura 12

Podemos ver que al aumentar  $\alpha$  un poco, los valores parecen tener un pico al principio, y luego rápidamente comienza a haber más error consistentemente. Esto sugiere que valores muy pequeños de autovalores podrían llegar a aportar más ruido que otra cosa, pero ya con un valor de  $\alpha$  tan pequeño como 1, estamos, quizás, tomando menos autovalores de los que deberíamos.

A continuación incluimos algunas imágenes generadas haciendo zoom en los  $\alpha$  más pequeños:

Cuadro 2: Imágenes variando alpha

Alpha	Tipo de Rayos	#Autovalores	Resultado
<b>0.001</b>	Random	793	
<b>0.01</b>	Random	769	
<b>0.1</b>	Random	668	
<b>0.6</b>	Random	420	
<b>1</b>	Random	344	
<b>10</b>	Random	143	
<b>100</b>	Random	59	

Podemos ver cómo los primeros valores de  $\alpha$  no tienen mucha diferencia en la calidad de la imagen; sin embargo entre 0.1 y 0.001 hay más de 100 autovalores hallados de diferencia, lo cual ya es una diferencia en el costo computacional considerable. Esto nos da la pauta de que los autovalores más pequeños al menos no hacen mucha diferencia en la imagen.

Podemos ver cómo ya al aumentar más el  $\alpha$  la cantidad de autovalores hallados baja considerablemente y la imagen generada no se parece en nada a la original.

### 3.4. Experimentando con el número de condición

En esta sección veremos cómo afecta el número de condición de la matriz  $D$  a los resultados obtenidos mediante la factorización SVD.

Como vimos en la sección anterior, el número de condición se puede expresar en función del mayor y el menor valor singular,  $\kappa(D) = \sigma_1/\sigma_r$ . Esto trae como resultado una relación entre la cantidad y la forma de los rayos que mandamos, la cual impacta en la cantidad de valores singulares que va a tener la matriz.

Nuestra motivación es ver cómo se relacionan los números de condición de los 3 tipos de rayos que generamos con la calidad de la imagen obtenida, y ver qué opción de rayos resulta más conveniente teniendo en cuenta el trade off entre calidad de la imagen y estabilidad a la hora de resolver el sistema.

En este primer experimento (figuras 13 y 14) medimos para los 3 tipos de rayos el número de condición de la  $D$  generada y la calidad del resultado medida con PSNR, variando la cantidad de rayos que enviamos.

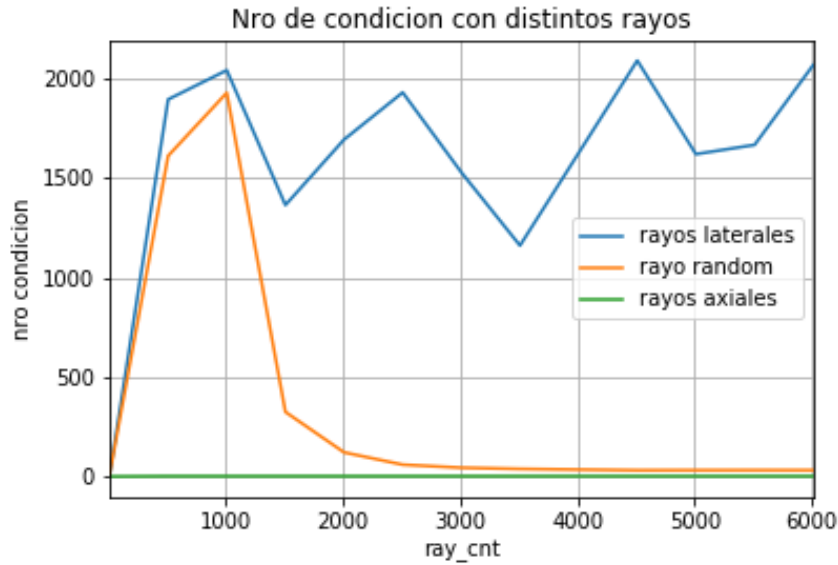


Figura 13

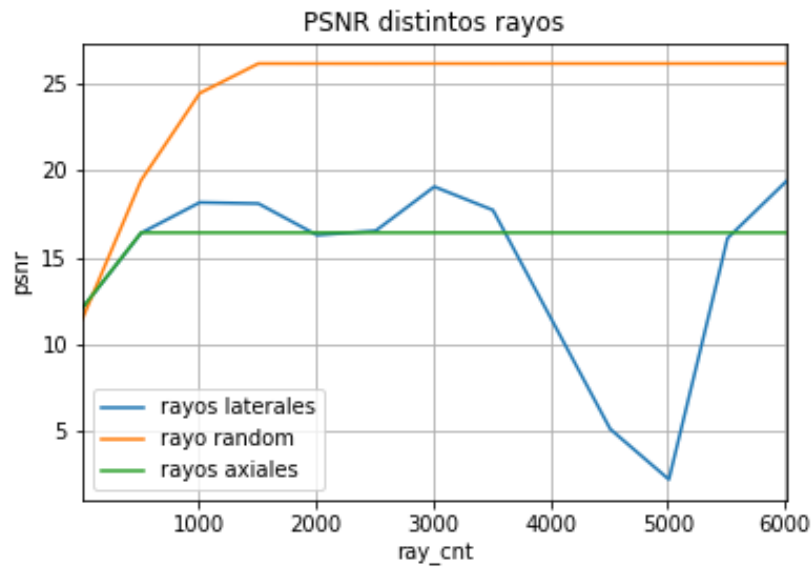


Figura 14

Podemos ver que si bien los rayos axiales tienen un número de condición muy bajo, su calidad es bastante mala y de hecho esto se condice con nuestra idea original de que al tener un número de condición bajo, no estamos tomando tanta información de la matriz y por lo tanto la imagen que conseguiremos tendrá una mala calidad.

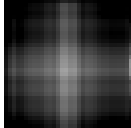
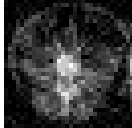
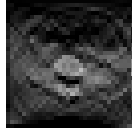
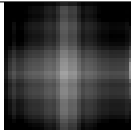
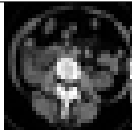
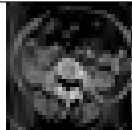
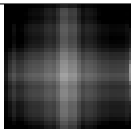

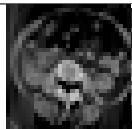
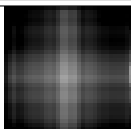


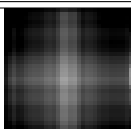

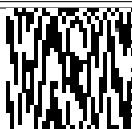
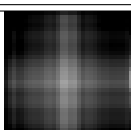

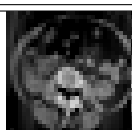
Por otra parte, los rayos laterales por momentos tienen mejor calidad y por momentos peor, y esto se entiende muy bien cuando vemos el número de condición que se va formando, el más alto de los 3, generando *outliers* por la inestabilidad del sistema.

Por último, los rayos random comienzan teniendo un número de condición parecido a los de los rayos laterales, pero luego disminuye enormemente aunque no llega a ser tan pequeño como el de los rayos axiales. Esto nos da la pauta de que los rayos random son mejores que los otros ya que tienen mejor PSNR y mejor número de condición, es decir, dan mejor calidad y producen sistemas estables.



Como último experimento, decidimos mostrar algunas de las imágenes generadas en las distintas corridas variando la cantidad de rayos:

Cuadro 3: Imágenes según alpha

cant rayos	Rayo Axial	Rayo Random	Rayo Lateral
510			
1510			
2510			
4010			
5010			
6010			

Podemos ver cómo los rayos axiales producen siempre una imagen muy parecida, donde no se distingue mucho qué hay. Los rayos laterales parecen ir mejorando pero en con 5010 rayos hay un outlier que produce una imagen inentendible.

Los rayos random son los ganadores indiscutidos tanto en calidad como en consistencia; no parece haber outliers lo cual tiene sentido dado el bajo número de condición, y su calidad va mejorando a medida que aumenta la cantidad de rayos.

### 3.5. Variando cantidad y tipo de rayos

El objetivo de esta experimentación es evaluar la repercusión en el resultado de la cantidad de rayos generados sobre el objeto y cuál es la mejor forma de generarlos. En particular, como fue mencionado anteriormente, implementamos 3 tipos de rayos: los axiales, laterales y aleatorios. Para los experimentos, decidimos evaluar los rayos utilizando las siguientes imágenes:

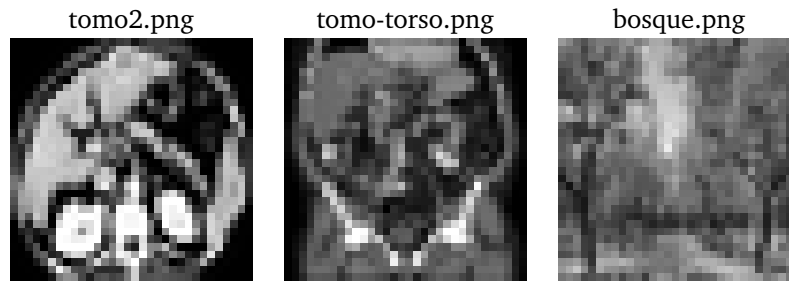


Figura 15: Imágenes utilizadas en exp. (32x32)

Las imágenes *tomo2.png* y *tomo-torso.png* casi que no tienen contenido relevante en los bordes, por lo que decidimos incluir la imagen *bosque.png* para obtener un panorama del comportamiento del algoritmo en éstos.

Para el experimento, corrimos el algoritmo en cada imagen para cada tipo de rayo y variando la cantidad de rayos de 100 a 20000, dando saltos de a 1000. Además, utilizamos error gaussiano con desvío estándar de 1 para los tiempos y  $\alpha = 0,0001$ . Para cada combinación de parámetros, tomamos el tiempo y el psnr de la imagen resultado con respecto a la original. Comenzamos mostrando los resultados sobre la imagen *tomo2.png* que fueron los siguientes:

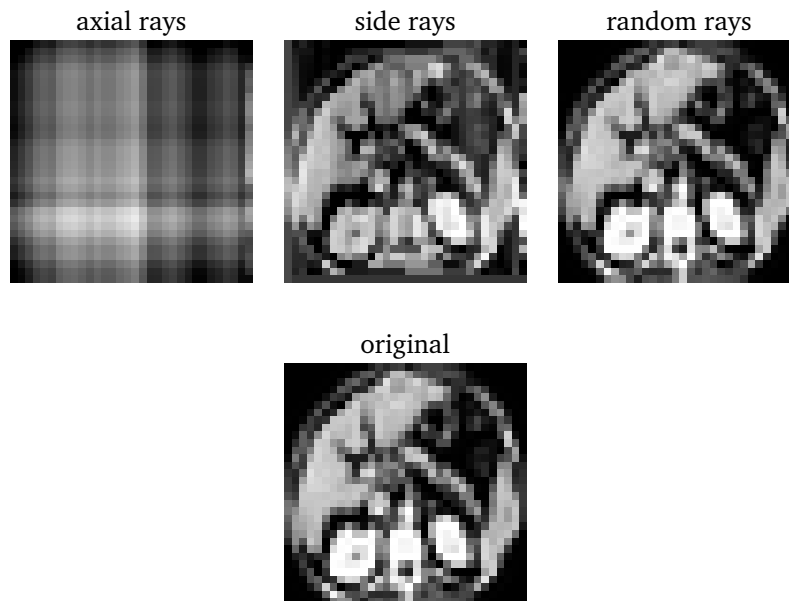


Figura 16: Resultados en tomo2.png de 32x32 con 20k rayos.

Lo primero que se puede observar es que los rayos axiales dan pésimos resultados (al menos para esta cantidad de rayos). En cuanto a los laterales y random, ambos arrojan un resultado bastante bueno aunque en los laterales se puede ver que no logra determinar bien los colores blancos fuertes en la parte inferior central. Veamos los tiempos y psnr al variar la cantidad de rayos:

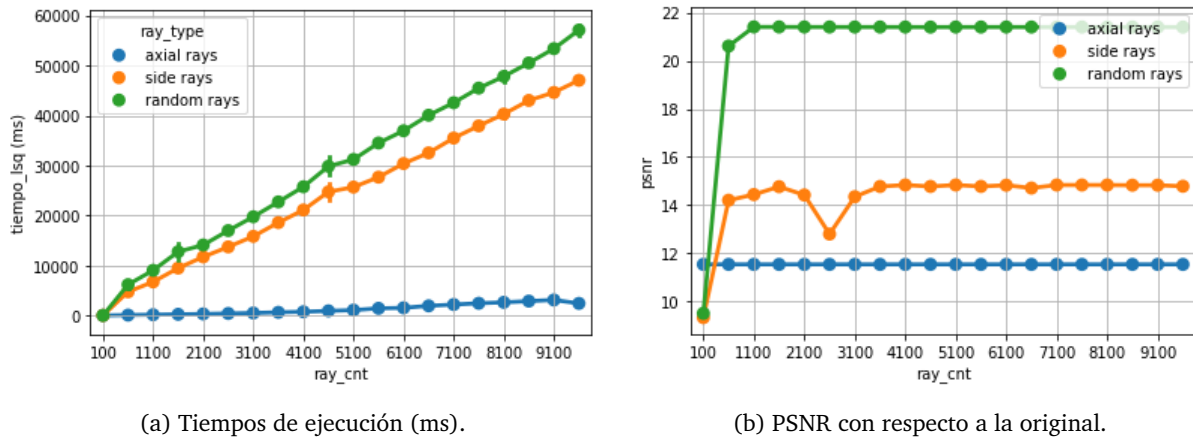


Figura 17: Resultados de variar la cant. de rayos para diferentes tipos de rayos sobre tomo2.png

Lo primero que se puede observar es que los tiempos de los rayos axiales y laterales son mucho mayores que los axiales. Esto debe estar relacionado con el rango de la matriz  $D$  resultante; en el caso de los rayos axiales, el rango se mantiene bajo porque permite mandar hasta 64 rayos distintos (32 por filas y 32 por columnas). El rango de esta matriz influye en los tiempos considerablemente porque se reducen los cálculos al tomar la SVD comprimida en el algoritmo.

En cuanto a los resultados obtenidos, los rayos random muestran mucho mejores resultados – con respecto al psnr – que los otros. Además, los resultados no mejora después de los 1100 rayos aproximadamente. Probablemente esto tenga que ver con que para este momento la matriz  $D$  alcanza el rango máximo y entonces el valor para cada celda ya esta determinado.

Repetimos el mismo experimento pero ahora con la imagen *tomo2-torso.png*. Evaluemos primero los resultados de cada tipo de rayo al mandar 20k rayos.

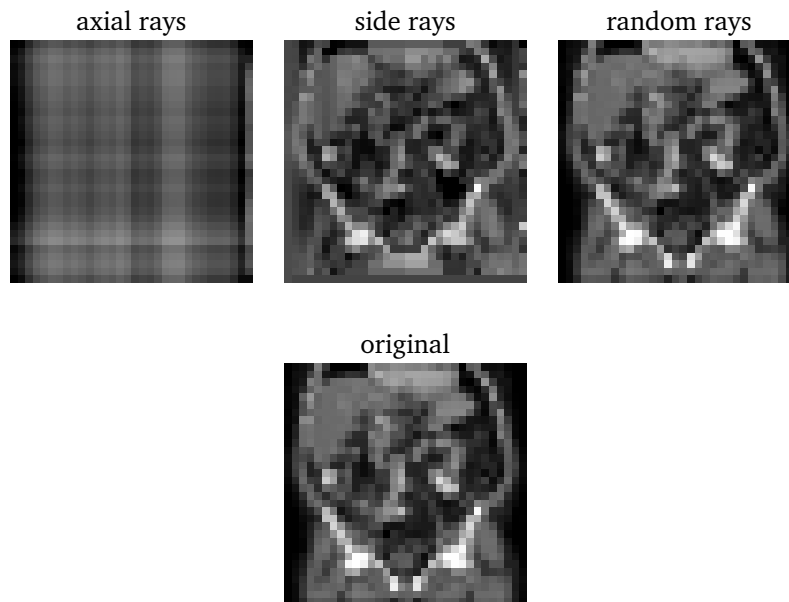


Figura 18: Resultados en tomo-torso.png de 32x32 con 20k rayos.

En este caso, se puede apreciar que la imagen generada por rayos laterales no logra capturar bien los bordes de la imagen original. En cuanto a los axiales, obtuvimos un resultado análogo al caso anterior. Por último, los random dan muy buenos resultados en este caso también. Veamos los tiempos y psnr obtenidos al variar la cantidad de rayos sobre esta imagen.

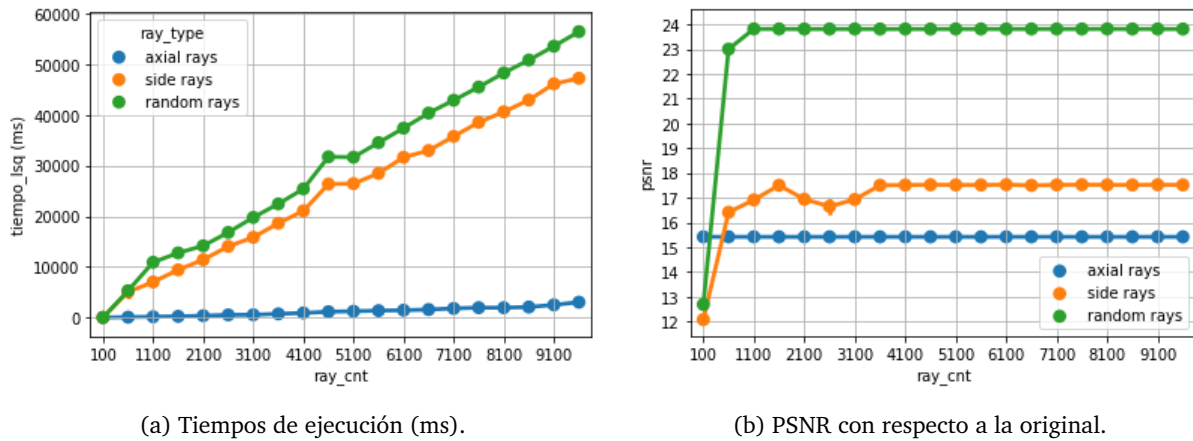


Figura 19: Resultados de variar la cant. de rayos para diferentes tipos de rayos sobre tomo-torso.png

Los resultados se ven análogos al caso anterior. Algo interesante que vemos en este caso y vimos en el caso anterior también es que hay un pico alrededor del 2200 en la figuras 19b y 17b. En otra experimentación vimos que el número de condición de los rayos laterales era muy malo, por lo cual podría explicar que un error numérico empeore mucho la solución.

Por último, veamos que sucede en la imagen *bosque.png* al generar 20k rayos:

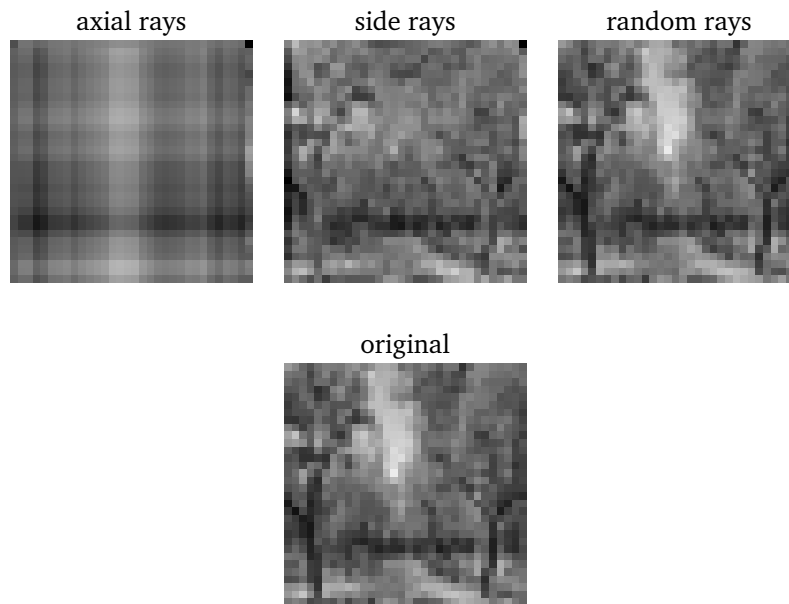


Figura 20: Resultados en bosque.png de 32x32 con 20k rayos.

Se puede notar mejor como los rayos laterales no logran capturar bien bordes ni el centro de la imagen original; más precisamente, los troncos de los árboles y el cielo. Por el contrario, los random tienen bastante buen resultado y los axiales malos, como antes. Veamos los tiempos y psnr al variar la cantidad de rayos sobre esta imagen:

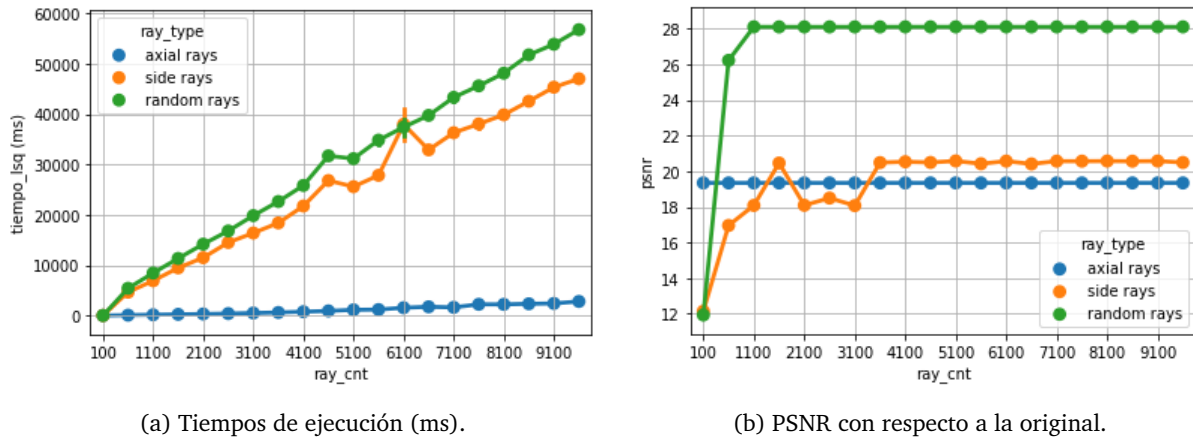


Figura 21: Resultados de variar la cant. de rayos para diferentes tipos de rayos sobre bosque.png

Los resultados vuelven a ser análogos al anterior. En este caso se puede notar mejor la inestabilidad de los rayos laterales con respecto a los errores.

En conclusión, vimos que los rayos random parecen ser los más efectivos en todos los casos. En el siguiente gráfico dejamos un resultado final que compara los resultados de éste tipo de rayos en todas las imágenes de muestra:

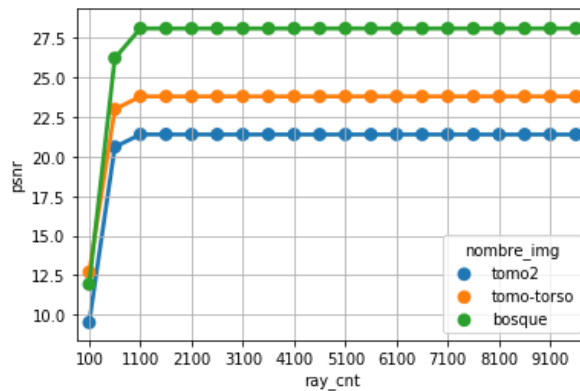


Figura 22: PSNR al variar la cant. de rayos para rayos random sobre distintas imágenes.png

Observamos que *bosque.png*, la imagen que parecía ser más compleja por su distribución de contenido y detalle, es la que otorga mejores resultados. Le siguen *tomo-torso.png* y luego *bosque.png*. Es importante notar que, en ese mismo orden, están menormente concentrados los contenidos, es decir, en el caso del bosque tiene contenido relevante en cualquier parte de la imagen; en el caso de la tomografía 2, no tiene prácticamente contenido en los bordes (está más bien centrada) y la tomografía del torso es un intermedio entre ambas. Luego, en los primeros casos se puede pensar que siempre que se generen rayos estos van a ser "útiles", o sea, nunca va a pasar que se genere un rayo que no aporte nada como podría pasar en *tomo2.png* si el rayo no pasa por el objeto.

## 4. Conclusiones

En este trabajo nos planteamos simular el comportamiento de un tomógrafo a partir de un modelo, donde lo propuesto es lanzar rayos que atraviesen cierta imagen y medir el tiempo que les toma y qué píxeles atraviesan. Luego, resolvimos un sistema de cuadrados mínimos lineales, donde el resultado es un vector que podemos pensarlo como una imagen.

Los resultados fueron muy buenos y nos permitieron experimentar cómo si tuviésemos un tomógrafo propio, cambiando la disposición de los rayos, la cantidad, el tiempo utilizado en resolver nuestro sistema obtenido, etc. Con esto, logramos comenzar con una disposición de los rayos trivial y muy simple, y evolucionarla a una más prometedora. Además, evaluamos un modelo de rayos pseudoaleatorios, que observamos que se comportaba MUY bien si utilizábamos una cantidad grande de rayos.

Además de todo esto, experimentamos utilizando descomposiciones SVD reducidas, tomando solo valores singulares grandes, esto mejoraba el tiempo de cómputo, pero también empeoraba (rápidamente) la calidad de la imagen resultante. Este trade-off plantea una pregunta interesante, sobretodo cuando se compara con el TP anterior y su método similar, PCA. ¿Qué tanto nos importa computar rápido? ¿Qué tan exacto tiene que ser el resultado para considerarlo valioso? ¿Qué nos importa más en el ámbito de la medicina (tomógrafos)? ¿Y en el de análisis de sentimiento (Twitter, Facebook, ...)? Si bien la idea no es responder estas preguntas, un análisis cómo el hecho en este trabajo nos permitiría argumentar mejor las respuestas.

A lo largo del trabajo utilizamos PSNR como medida de error, y obtuvimos valores entre 15 y 30, que, según el libro de Salomon [1], son valores aceptables para compresión de imágenes. Cómo mostramos en la experimentación, para cada rayo este valor variaba, a medida que cambiábamos sus parámetros, tendiendo a alcanzar un máximo cuantos más rayos utilizábamos. De nuevo, sin perder de vista al objeto de estudio, hay lugar para preguntarse ¿es gratis lanzar más y más rayos?. Sobretodo cuando cada uno de estos atraviesa el cuerpo de una persona.

Por todo lo hablado arriba, consideramos que disponer de un simulador de tomógrafos como este, podría ser útil a la hora de analizar su funcionamiento y posibles mejoras.

## 5. Apéndice

### 5.1. Detalles de implementación

#### Matrices ralas

Destacamos que, en algunas situaciones críticas donde consideramos que valía la pena, utilizamos matrices ralas (heredadas de los trabajos anteriores). Esto significa mejoras tanto en la complejidad espacial como en la temporal.

#### Cache

Uno de los detalles particulares de nuestra implementación es el uso de una caché, la idea es aislar el cómputo que más demora (la descomposición SVD) y guardarlo para ser utilizada luego. La gran ventaja de hacer esto es que, luego, podemos correr nuestro programa muy rápidamente.

La razón que nos motivó a hacerlo, vino de notar que nuestra matriz  $D$ , sobre la cuál queremos resolver cuadrados mínimos lineales, no depende de la imagen, sino simplemente de su tamaño y de la disposición de los rayos, pudiendo computar esto una sola vez y luego ejecutarlo para muchas imágenes de forma barata.

## Referencias

- [1] Salomon, David. (2007). Data Compression: *The Complete Reference*. 281.
- [2] Wu, Xiaolin (1991). "Fast Anti-Aliased Circle Generation". In James Arvo. Graphics Gems II. San Francisco: Morgan Kaufmann. pp. 446–450. ISBN 0-12-064480-0.