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

Reconstrucción de imágenes tomográficas

Huaier, Damián Fernando damianhuaier@gmail.com
Marenco, Mateo mateomarenco74@gmail.com
Salvia, Daniel Matías danmats140@gmail.com
Togno, Ezequiel ezetogno@gmail.com

En este trabajo implementamos un método para reconstruir imágenes tomográficas. Dicho método busca aproximar las densidades de un objeto estudiado, atravesado por una cierta cantidad de rayos X. Disponiendo de los tiempos y las distancias recorridas por cada rayo se obtienen las densidades como una solución aproximada de un sistema de ecuaciones sobredeterminado. Dicha aproximación se realiza por *cuadrados mínimos*.

En el presente informe se detalla el proceso de simulación tomográfico para generar datos de prueba, la implementación de cuadrados mínimos mediante la descomposición SVD, y la calidad de la imagen reconstruida en relación a la cantidad de rayos y la forma en que son generados, así como la manera en que se divide el objeto de estudio.

Palabras clave: `simulación tomográfica`, `cuadrados mínimos`, `SVD`

Índice

1. Introducción

Nuestro objeto de estudio es, en teoría, un cuerpo continuo, cuadrado y bidimensional, con densidad variable en cada punto. Para reconstruir estas densidades primero se discretiza el cuerpo en cuestión, es decir, consideramos que está dividido en $n \times n$ celdas discretas. Los datos de entrada para nuestro método consisten, por cada rayo k emitido sobre el cuerpo, en:

- El tiempo que tardó el rayo en atravesar el objeto, que llamaremos t_k.
- La distancia que el rayo recorrió en cada celda ij, que llamaremos $d_{ij}^{(k)}$.

Para este trabajo consideraremos que la densidad en una celda ij viene dada por v_{ij}^{-1}, que se entiende por la inversa de la velocidad con la que cualquier rayo atraviesa dicho punto. Teniendo en cuenta esto, notemos que para cada $k = 1 \dots m$, siendo m la cantidad de rayos, se tiene:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}^{(k)} v_{ij}^{-1} = t_k$$

Si consideramos la matriz $D \in \mathbb{R}^{m \times n^2}$ que en cada fila k posee todas las distancias $d_{ij}^{(k)}$ ordenadas primero por i y después por j, y el vector $t = (t_1, \dots, t_m)^t \in \mathbb{R}^m$, entonces el vector de intensidades de cada celda es una solución del sistema:

$$Ds = t \quad s \in \mathbb{R}^{n^2}$$

Dado que no es posible medir el tiempo de los rayos sin errores, los tiempos t_k traerán consigo cierto error, que será la causa de la imposibilidad de encontrar una solución exacta al sistema $Ds = t$. Por esta razón acudimos al método de cuadrados mínimos: se busca s tal que minimice la cuenta $||Ds - t||^2$, o dicho de otra manera, buscamos el vector s que más se "parezca" a una solución del sistema.

Este problema puede resolverse calculando la descomposición SVD[1] de la matriz D. Siendo $D = U\Sigma V^t$ dicha descomposición, podemos escribir:

$$U^t t = \begin{pmatrix} c \\ d \end{pmatrix}$$

con $c \in \mathbb{R}^r$, $d \in \mathbb{R}^{m-r}$ y $r = \text{rg}(D)$. Para hallar la solución primero se debe calcular el vector $y \in \mathbb{R}^{n^2}$ dado por:

$$y = \begin{pmatrix} c_1/\sigma_1 \\ \vdots \\ c_r/\sigma_r \\ y_{r+1} \\ \vdots \\ y_{n^2} \end{pmatrix}$$

con $\sigma_1 \dots \sigma_r$ los valores singulares de D y $y_{r+1} \dots y_{n^2}$ arbitrarios. Finalmente se obtiene la solución $s = Vy$, que minimiza la norma $||Ds - t||^2$ y por lo tanto contendrá los valores aproximados de las densidades de todas las n^2 celdas del objeto [2].

[1] https://davidtabora.files.wordpress.com/2015/01/david_s-_watkins_fundamentals_of_matrix_computat.pdf. Página 262.

[2] En el apéndice puede encontrarse una demostración de que una solución construida de esa manera es solución de cuadrados mínimos.

La calidad de la solución obtenida dependerá de la cantidad de rayos generados, de sus recorridos, del tamaño de la discretización y del nivel de ruido que posean los tiempos t_k. Analizaremos con detalle todos estos aspectos posteriormente, a partir de las experiencias realizadas.

Existe un inconveniente, y es que no se cuenta con datos provenientes de un tomógrafo real. Por lo tanto, en este trabajo trabajaremos con imágenes en escala de grises que representarán el objeto a estudiar. Los píxeles con intensidad más cercana a 255 (la máxima) indicarán zonas de mayor densidad y tendrán un color más claro, mientras que aquellos más cercanos a 0 (la mínima) se corresponderán con zonas de menor densidad y seran de un color más oscuro. Esta será una representación aproximada del cuerpo continuo y bidimensional que mencionamos anteriormente.

Una vez elegida dicha imagen se determina la discretización a utilizar, que consiste en decidir cuántos píxeles contendrá cada celda de la imagen. Posteriormente se realiza una simulación de los rayos X: dado un rayo y una celda ij calcularemos $d_{ij}^{(k)}$ como la cantidad de píxeles atravesados por el rayo en esa celda, es decir, consideramos que la distancia recorrida por cualquier rayo en cualquier pixel es 1. El tiempo que tarda en recorrer esa celda será entonces la suma de las intensidades de los píxeles tocados, resultado que será sumado al tiempo total t_k. Este proceso se realiza para todas las celdas.

Luego de realizar esta simulación añadiremos cierto nivel de ruido a todos los tiempos t_k, es decir, le sumaremos un valor aleatorio, para simular las imprecisiones que surgen de medir esta magnitud.

El objetivo general del trabajo es crear un programa en C++ que tome como dato de entrada una imagen en escala de grises, en formato `csv`, que represente el objeto a analizar, más una serie de parámetros que determinan la discretización a utilizar, la forma de generar los rayos, y el nivel de ruido. El programa realizará la simulación tomográfica y con los datos obtenidos reconstruirá la imagen. Esta nueva imagen será guardada en un `csv` de salida.

Como parte de la experiencia se analizará la calidad de la imagen reconstruida en relación a la imagen original, utilizando la métrica PSNR[3] —Peak signal-to-noise ratio—, que se define como:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}_u^2}{\text{ECM}} \right)$$

donde MAX_u es el rango máximo de la imagen, que en nuestro caso será 255, y ECM es el *error cuadrático medio*:

$$\text{ECM} = \frac{1}{N} \sum_{ij} (u_{ij}^{(0)} - u_{ij})^2$$

siendo $u^{(0)}$ la imagen original, u la imagen reconstruida y N la cantidad de píxeles.

[3] http://www.ni.com/white-paper/13306/en/

2. Desarrollo

2.1. Manejo de las imágenes

Nuestro programa aceptará únicamente imágenes cuadradas en formato `csv`. Estas son representadas como un simple arreglo de arreglos de bytes. En caso de recibir una imagen cuyo tamaño no es divisible por el tamaño de las celdas especificado se presenta un problema puesto que no será posible dividir la misma en celdas cuadradas de manera exacta.

Una posible solución es extender la imagen con píxeles negros o de otra intensidad específica para que pueda ser discretizada. Sin embargo, si pensamos en las celdas que tendrán estos nuevos píxeles, su intensidad resultante luego de la reconstrucción puede llegar a ser muy influenciada por la presencia de dichos píxeles. Por lo tanto, decidimos que estos mismos posean distancia cero y que el tiempo en recorrerlos sea nulo. Es decir, directamente ignoramos su presencia. Visto de otra manera, lo que estamos haciendo es permitir que las celdas que no caben en la imagen sean no cuadradas para que sí lo hagan.

Una vez obtenida la imagen reconstruida, está será de menor tamaño que la original. Para poder obtener el PSNR decidimos agrandar la nueva imagen para que ambas tengan el mismo tamaño. Con el fin de estudiar la calidad de los resultados obtenidos únicamente con nuestro método de reconstrucción, la manera de agrandar la imagen no utilizará interpolación. El método será el más simple posible: cada píxel de ella, que se corresponde con una celda de la discretización, será dividido en tantos píxeles como los que tenía dicha celda, pero con la intensidad proveniente de la reconstrucción.

2.2. Método de simulación de rayos

Lo que haremos es pensar en la imagen original como un área del plano $\mathbb{R}^2$, pero el eje y apuntará hacia abajo.

Consideramos que para simular un rayo se deben especificar dos píxeles del borde de la imagen, $p_0 = (x_0, y_0)$ y $p_1 = (x_1, y_1)$. El rayo generado será una recta que pase por los centros de estos dos píxeles, es decir, por los puntos de $\mathbb{R}^2$ $\tilde{p_0} = (\tilde{x_0}, \tilde{y_0}) = (x_0 + 0.5, y_0 + 0.5)$ y $\tilde{p_1} = (\tilde{x_1}, \tilde{y_1}) = (x_1 + 0.5, y_1 + 0.5)$.

La idea es recorrer todos los píxeles que va atravesando la recta, y por cada uno de ellos sumamos su intensidad al tiempo total del rayo y además sumamos 1 a la distancia recorrida en la celda correspondiente.

La recta estará definida por la ecuación $y = f(x) = ax+b$ siendo $a = (\tilde{y_1}-\tilde{y_0})/(\tilde{x_1}-\tilde{x_0})$ y $b = \tilde{y_0}-a\tilde{x_0}$.

Como pixel de inicio se eligirá entre p_0 y p_1 a aquel que esté más a la izquierda. Para ver como se recorren los píxeles, llamemos (x, y) a las coordenadas (enteras) de este pixel. Lo que haremos es evaluar nuestra función f en el borde derecho del pixel, es decir, obtendremos $f(x + 1)$ y en base a su resultado tomaremos los siguientes caminos:

- $f(x + 1) < y$: pasar al pixel de arriba.
- $f(x + 1) = y$: pasar al pixel que está arriba y hacia la derecha.
- $y < f(x + 1) < y + 1$: pasar al píxel de la derecha.
- $f(x + 1) = y + 1$: pasar al pixel que está abajo y hacia la derecha.
- $y + 1 < f(x + 1)$: pasar al píxel de abajo.

Es decir, no hacemos más que saltar al pixel que sea tocado por el rayo. Este proceso se repite hasta que detectemos que ya no estamos más parados en el área de $\mathbb{R}^2$ ocupada por la imagen.

Si el rayo es vertical, es decir $x_0 = x_1$, entonces los puntos exactos por los que pasa la recta, $\tilde{p_0}$ y $\tilde{p_1}$, serán ligeramente corridos (uno hacia la izquierda y el otro hacia la derecha) de manera que la recta

quede levemente "torcida" y así poder calcular los coeficientes a y b de la recta, pero evitando que la misma colisione con píxeles de más.

El nivel de ruido será especificado por una constante positiva R. Luego de simular cada rayo y obtener el tiempo t_k, a este valor se le sumará un número aleatorio en el intervalo $[-R, R]$.

2.3. Representación de matrices

Es claro que la cantidad de píxeles que atraviesa un rayo es escasa en comparación con la cantidad total de píxeles de la imagen. Si la imagen tiene $\ell \times \ell$ píxeles podemos esperar que cada rayo atraviese ℓ píxeles aproximadamente. Como consecuencia tendremos que para cada k habrá muchos $d_{ij}^{(k)}$ nulos, y por lo tanto la matriz D será muy rala.

La decisión más razonable es guardar solo los elementos no nulos de D para utilizar menos memoria y agilizar los cálculos que se deberán realizar posteriormente. Lo que haremos entonces es representar a D como un arreglo de vectores ralos, donde cada vector ralo representa una columna de la matriz. Por su parte un vector ralo se representa como un arreglo de pares, donde cada par contiene un valor no nulo del vector y la posición de este en el vector. Estos pares están ordenados en orden creciente de acuerdo a las posiciones.

La decisión de representar las columnas de D con vectores ralos, en vez de las filas, tiene la siguiente motivación: sabiendo que para la descomposición SVD calcularemos la matriz D^tD y que $(D^tD)_{ij}$ se calcula como el producto interno entre la columna i y la columna j de D, estos productos internos serán obtenidos con mucha rapidez si las columnas son vectores ralos.

Sin embargo, no hay garantías de que D^tD resulte rala, por lo tanto dicha matriz será representada como un arreglo de arreglos de `double`'s. Lo mismo vale para el resto de las matrices que se deban calcular.

2.4. Cálculo de autovalores y autovectores

La descomposición SVD requerirá la obtención de los autovalores y autovectores de D^tD. Para llevar a cabo esta tarea utilizaremos el *método de la potencia con deflación*, aplicable a matrices con base de autovectores ortogonales, como D^tD. En primer lugar consideramos un vector v_0 arbitrario y la sucesión:

$$v_{k+1} = \frac{D^tDv_k}{||D^tDv_k||}$$

para hallar el autovector asociado al autovalor de mayor magnitud λ, que a su vez puede calcularse como $\lambda = v^tD^tDv$ siendo v el autovector obtenido. Una vez obtenidos v y λ se procede a calcular la matriz $D^tD - \lambda vv^t$, a la cual se le aplica nuevamente el método de la potencia para hallar el segundo autovalor y autovector dominante de D^tD. Este proceso se sigue repitiendo hasta hallar todos los autovectores y autovalores.

Es claro que este método puede llegar a ser muy inestable puesto que cada autovalor λ y autovector v que hallemos tendrán un cierto error, debido a cuestiones numéricas y a que quizás no se realizó una cantidad de iteraciones suficiente. Por más que sea aceptable, dicho error se trasladará al cálculo de la nueva matriz durante el paso de la deflación, y por lo tanto los siguientes autovalores y autovectores tendrán aún más error.

Para lidiar con esta situación decidimos adoptar el siguiente criterio de convergencia: la iteración termina cuando el vector actual v_k está lo suficientemente cerca de ser un autovector de D^tD, es decir, cuando:

$$E_k = \frac{||D^tDv_k - \lambda_k v_k||}{||\lambda_k v_k||} \leq \varepsilon_1$$

con $\lambda_k = v_k^t D^t D v_k$ y ε_1 una constante global. Es importante remarcar que esta condición se prueba utilizando $D^t D$ en vez de la matriz actual a la que le estemos aplicando el método de la potencia. Esto asegurará que todos los autovectores y autovalores sean igual de precisos que los primeros.

Debido a que con este criterio puede ocurrir que el error E_k nunca llegue a ser menor o igual a ε_1, nuestra implementación detecta esta situación, viendo que $E_k = E_{k-1}$ (es decir, luego de una iteración el error no mejoró) y en ese caso devuelve el autovalor y autovector actual tal como si el criterio se hubiera cumplido. Si E_k era casi menor o igual a ε_1 entonces el resultado devuelto será aceptable y típicamente no se volverá a repetir la misma situación durante el cálculo de los próximos autovalores y autovectores (esto puede ser corroborado experimentalmente).

Sin embargo, si E_k es demasiado alto (mayor a otra constante global más grande que ε_1, llamada ε_2) entonces nuestra implementación frenará el cálculo de todos los autovalores y autovectores. Además, también se frenará si un autovalor resultó mucho más grande que el anterior, o si un autovalor dio negativo. Esta última condición está impuesta porque $D^t D$ es semi-definida positiva y por lo tanto sus autovalores son no negativos.

Notemos que en esos casos frenar todo el cálculo es lo más sensato, ya que si no lo hacemos los próximos autovalores y autovectores ya no serán válidos puesto que en la deflación habremos usado un λ y un v erróneos.

Para concluir esta sección se describe en el Algoritmo 1 el método propuesto, siendo A la matriz a la cual calcularle el autovalor de mayor magnitud λ junto con su autovector asociado v, y siendo A_0 la matriz con la cual se computará el error E_k en cada iteración. El vector v_0 es el vector inicial con el cual se empieza a iterar, y la variable *outcome* es un indicador de qué tan buenos fueron los resultados.

Algorithm 1: Cálculo de autovalor y autovector dominante.

Input: A, v_0, A_0
Output: λ, v, *outcome*

```
1  v ← v_0
2  for k = 1 ... do
3      for i = 1 ... N do
4          v ← Av / ||Av||
5      λ ← v^t A v
6      E_k ← ||A_0 v − λv|| / ||λv||
7      if E_k ≤ ε_1 then
8          outcome ← "Resultado aceptable"
9          return λ, v, outcome
10     else if E_k = E_{k−1} then
11         if E_k ≤ ε_2 then
12             outcome ← "Resultado casi aceptable"
13         else
14             outcome ← "Resultado inaceptable"
15         return λ, v, outcome
```

Notemos que la instrucción $v \leftarrow \frac{Av}{||Av||}$ se ejecuta una cantidad de veces N antes de evaluar si seguir iterando o no. Esto genera que se gaste menos tiempo calculando el error E_k, el cual requiere una multiplicación de matriz por vector adicional ($A_0 v$).

Nuestra implementación fija los valores $N = 25$, $\varepsilon_1 = 0.01$, y $\varepsilon_2 = 0.1$. Notemos que 0.01 no es un límite muy exigente para ε_1, sin embargo, gracias a que el error se mide utilizando la matriz original, los resultados son bastante aceptables, y al poder utilizar un límite no tan exigente el método termina haciendo menos iteraciones y por lo tanto se mejora el rendimiento. Si midiéramos el error usando la

matriz actual (en el caso del Algoritmo 1, se trata de la matriz A) tendríamos que reducir el valor de ε_1 y por lo tanto el tiempo de procesamiento sería mayor.

Con esta función se calculan todos los autovectores y autovalores de D^tD que sean posibles. Los chequeos de signo y magnitud de los autovalores devueltos, mencionados anteriormente, son llevados a cabo por fuera de esta función.

2.5. Descomposición SVD y cuadrados mínimos

Según se mencionó en la introducción, obtener la solución de cuadrados mínimos resulta muy sencillo conociendo la factorización SVD de la matriz D.

Para obtener esta descomposición decidimos calcular D^tD en vez de DD^t porque la primera será de tamaño $n^2 \times n^2$ y la segunda de $m \times m$, y típicamente la cantidad de rayos m será mayor a la cantidad de celdas n^2, por lo tanto D^tD será más chica. Su cálculo consiste en realizar los productos internos entre las columnas de D para determinar cada $(D^tD)_{ij}$. Esto se hace solo para $i \leq j$, puesto que $(D^tD)_{ij} = (D^tD)_{ji}\ \forall i, j$. Los productos internos serán más rápidos de lo normal gracias a que las columnas son vectores ralos.

Una vez obtenida D^tD se procede a calcular sus autovalores y autovectores. Los autovectores serán las columnas $v_1 \dots v_n$ de V y las raíces cuadradas de los autovalores (que serán positivos puesto que D^tD es semi-definida positiva) serán los valores singulares $\sigma_1 \dots \sigma_r$ con $r = \text{rg}(D)$.

Si recordamos lo que se mencionó en la introducción, podemos deducir que solo nos interesan las primeras r filas de U^t, ya que el resto de las filas solo importan para el resultado del vector d, que no utilizamos. Entonces podemos usar que, para todo i entre 1 y r:

$$Dv_i = \sigma_i u_i$$
$$\implies u_i = Dv_i/\sigma_i$$

Esto nos brinda un método sencillo y eficaz para calcular las filas de U^t que nos importan. Calcular Dv_i en principio parece complicado al no poder operar fácilmente con las filas de D. Sin embargo, podemos pensar que Dv_i es una combinación lineal de las columnas de D, donde los coeficientes son los elementos de v_i. Aprovechando que las columnas son vectores ralos resulta fácil optimizar el cálculo de esta combinación lineal.

Una vez obtenidas U^t y V y los valores singulares, se procede a calcular el vector $c = U^tt$, con el cual obtendremos el vector y según figura en la introducción. Los valores $y_{r+1} \dots y_n$ son arbitrarios, puesto que cualquier elección dará lugar a una solución de cuadrados mínimos (es decir, las soluciones son infinitas, justamente porque $r = \text{rg}(D) < n$). Por convención, decidimos dejar todos ellos como cero. La solución final se obtiene haciendo el producto $s = Vy$, que puede entenderse como una combinación lineal de las columnas de V.

Existe un problema, y es que es posible que no obtengamos todos los autovalores y autovectores de V, según nuestra implementación del método de la potencia. En este caso lo que hacemos es quedarnos con los ℓ autovalores y autovectores que hayamos podido calcular, y en este caso las matrices U y V serán más chicas y dispondremos solamente de ℓ valores singulares. Como consecuencia los vectores c e y no tendrán r elementos si no ℓ, y el resultado final será una combinación lineal de las ℓ columnas de V.

Visto de otra manera, lo que sucede es que obtenemos una combinación lineal de no todas las columnas de la verdadera matriz V, puesto que las otras no las tenemos porque no las hemos podido calcular. Si las columnas que faltan son solo unas pocas podemos esperar que el resultado s sea más o menos aceptable.

Durante este proceso también calcularemos $\kappa(D^tD)$, el número de condición de D^tD. Dado que esta matriz es simétrica y semi-definida positiva sus autovalores, que ya habremos calculado, serán iguales a sus valores singulares. Sabiendo que el número de condición se puede calcular como el valor singular más grande dividido el más chico, $\kappa(D^tD)$ será obtenido dividiendo su autovalor más grande por el más chico.

El número de condicion de D^tD resulta de particular interés ya que toda solución de cuadrados mínimos del sistema $Ds = t$ es solución de las *ecuaciones normales*:

$$D^tDs = D^tt$$

Cuanto más cercano a 1 sea $\kappa(D^tD)$, más estable será la solución obtenida. Por otro lado, si $\kappa(D^tD)$ es muy grande la solución será muy inestable y la matriz estará más cerca de ser singular. En efecto, si la matriz es singular entonces el autovalor más chico será cero y el número de condición no podrá calcularse. Este viene a ser el peor escenario en cuanto a estabilidad.

Por lo tanto una condición necesaria para no caer en este caso es que D^tD sea inversible, lo que es equivalente a pedir que D^tD sea SDP (ya que sabemos que es simétrica semi-definida positiva), a que la solución de las ecuaciones normales sea única, y a que la solución de cuadrados mínimos de $Ds = t$ sea única. Esta última condición es equivalente a que las columnas de D sean linealente independientes, y su cumplimiento o no dependerá de la discretización utilizada y principalmente de la forma de generar los rayos.

2.6. Hipótesis sobre los resultados

En primer lugar, es claro que cuanto menor sea el tamaño de las celdas, mayor será la calidad de la reconstrucción. Sin embargo resulta difícil trabajar con tantas celdas debido al tiempo de ejecución, puesto que la matriz D^tD será muy grande. Esto no solo implica que se deberán hallar más autovalores, si no que además las iteraciones del metodo de la potencia serán más costosas. Dependiendo del contexto de uso se puede discretizar el cuerpo en partes muy grandes para priorizar el rendimiento, o bien se puede optar por obtener una reconstrucción de mayor calidad dividiendo el cuerpo en partes más pequeñas.

Por otro lado, más allá del tamaño de celda elegido, no cualquier forma de generar los rayos resultará efectiva. Lo mínimo que se debe pedir para obtener una solución estable es que la cantidad de rayos sea mayor a la cantidad de celdas, es decir, que D tenga más filas que columnas, y que además toda celda sea atravesada por al menos un rayo. Si cualquiera de esas dos condiciones no se cumplen entonces las columnas de D no serán l.i. Sin embargo, esto no garantiza que, en caso de cumplirse, sean l.i.

Resulta difícil imaginarse una forma de generar rayos que garantice independencia lineal. Lo mejor que se puede hacer es generar la mayor cantidad de rayos posibles, en diversas direcciones, de manera aleatoria o no.

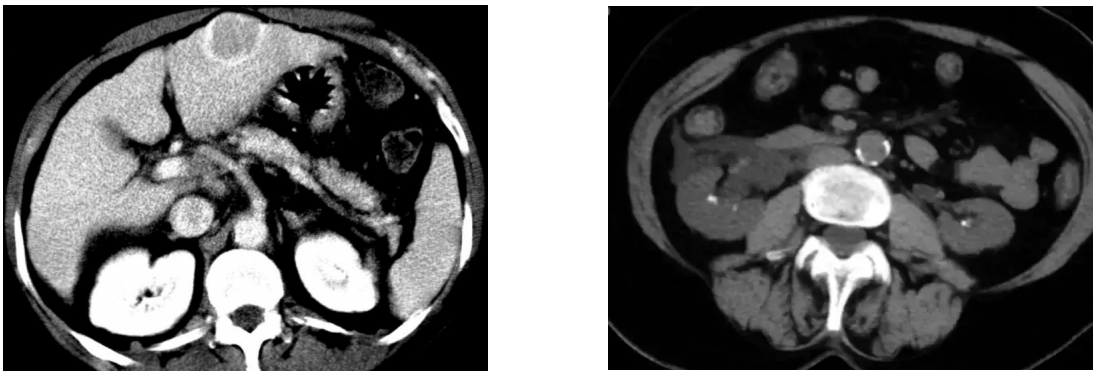
También es importante que no solo cada celda sea atravesada por al menos un rayo. Se necesita que sea atravesada por varios rayos que en conjunto logren pasar por todos sus píxeles. Si hay píxeles que no son atravesados estaremos descartando la información que aportan, al momento de reconstruir la imagen.

Con respecto al rendimiento, la cantidad de rayos generados influirá en tres partes de nuestra implementación: la simulación tomográfica, el cálculo de D^tD, y la obtención de las filas de U^t. Si estamos trabajando con imágenes muy grandes deberemos ser cuidadosos a la hora de elegir la forma de generar los rayos puesto que podríamos generar un *bottleneck*. De más está decir que, si se tratara de un caso real, habría un límite en la cantidad de radiación a generar ya que esta puede ser altamente nociva.

Por último, un parámetro más a tener en cuenta es el nivel de ruido que se agregará a los tiempos t_k. Naturalmente, a mayor cantidad de ruido, peor será la calidad de la imagen reconstruida, y más aún si el número de condición de D^tD es muy alto. Sin embargo no tendrá efecto alguno en el rendimiento. De hecho, nuestra implementación permite realizar todo el proceso de simulación y reconstrucción para varios niveles de ruido a la vez.

2.7. Propuesta de experimentación

En primer lugar experimentaremos con diferentes tamaños de celda para las imágenes `tomo2.csv` y `tomo3.csv`, de tamaño 549×549 y 452×452 píxeles respectivamente. En la Figura 1 mostramos ambas



(a) `tomo2` **(b)** `tomo3`

Figura 1: Imágenes de prueba `tomo2` y `tomo3`.

Figura 2: Imagen de prueba `phantom.csv`.

imágenes pero en sus tamaños originales, donde no son cuadradas.

Como nuestro objetivo será estudiar cómo varía tanto la calidad según nuestra apreciación subjetiva como el valor del PSNR en función del tamaño de celda, dejaremos fijos el método de generación de rayos y el nivel de ruido. El primero será un método con el cual se generan 20000 rayos aleatorios (será descripto en breve), y el nivel de ruido será de 0.0. Además, veremos si el tamaño de celda influye en el número de condición $\kappa(D^tD)$.

Por otro lado, para las imágenes `tomo2.csv`, `phantom.csv` (mostrada en la Figura 2) y `tomo.csv` (igual a `tomo2` pero redimensionada a 100×100 píxeles) probaremos los siguientes métodos de generación de rayos:

- Rayos verticales, horizontales y diagonales: consiste en generar todos los rayos posibles que sean

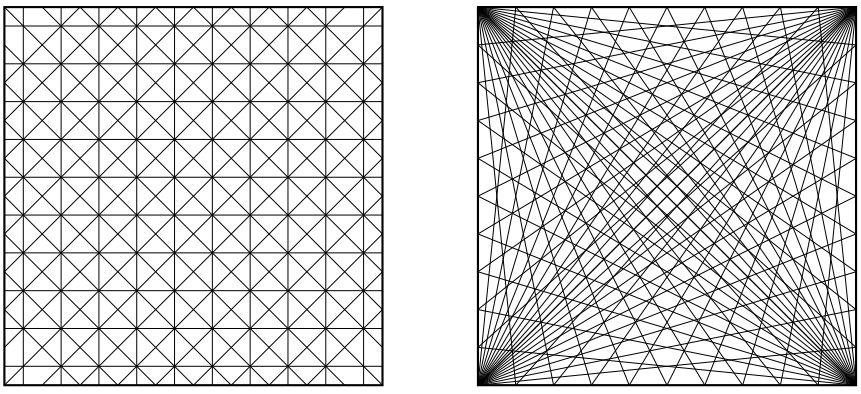


Figura 3: Rayos horizontales, verticales y diagonales (izquierda), y rayos que barren la imagen (derecha).

horizontales, verticales y diagonales con pendiente 1.

- Rayos que barren la imagen: consite en pararse en cada una de las cuatro esquinas de la imagen y desde ese punto generar rayos que vayan a todos los píxeles del borde (exceptuando los bordes que comparten dicho punto).
- Rayos aleatoros: dada una cantidad de rayos deseada m, cada rayo aleatorio se genera de la siguiente manera:
  - Se elige aleatoriamente si el rayo viajará del borde superior al inferior, del izquierdo al derecho, del superior al derecho, del derecho al inferior, del inferior al izquierdo o del izquierdo al superior.
  - Se eligen dos puntos aleatorios, uno de cada borde elegido, que determinarán el rayo.
- Método "completo": se generan todos los rayos posibles que pasen por dos píxeles que vivan en bordes opuestos de la imagen.

Por último tomaremos la imagen `tomo2.csv` y estudiaremos el PSNR obtenido en función del nivel de ruido agregado, así como la calidad de la imagen reconstruida para algunos niveles de ruido, con el objetivo de medir el impacto que tiene este parámetro.

3. Resultados

En las figuras mostramos los diversos resultados obtenidos según la propuesta de experimentación.

Los resultados obtenidos con el método de rayos horizontales, verticales y diagonales son los siguientes:

- `phantom.csv`: PSNR = 15.2316.
- `tomo3.csv`: PSNR = 17.0948.

Con el método de barridos:

- `phantom.csv`: PSNR = 12.4005
- `tomo3.csv`: PSNR = 13.9602

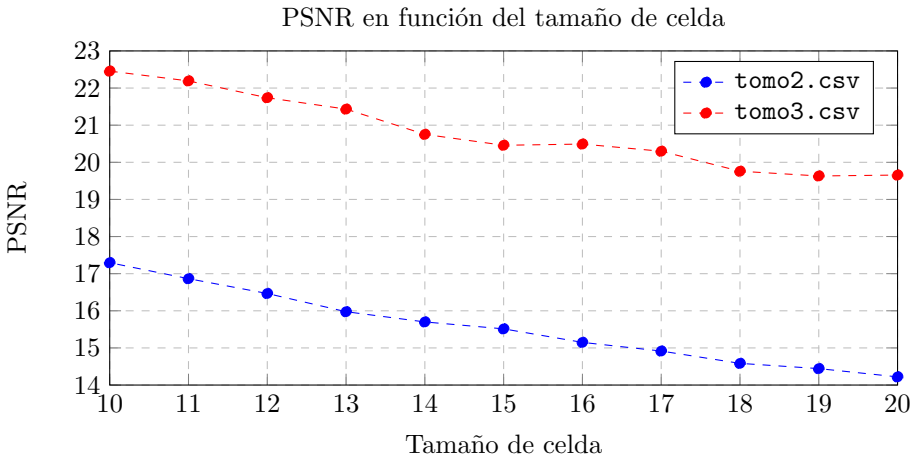


Figura 4: PSNR en función del tamaño de celda, para las imágenes `tomo2.csv` y `tomo3.csv`. Nivel de ruido 0.0, 20000 rayos aleatorios.

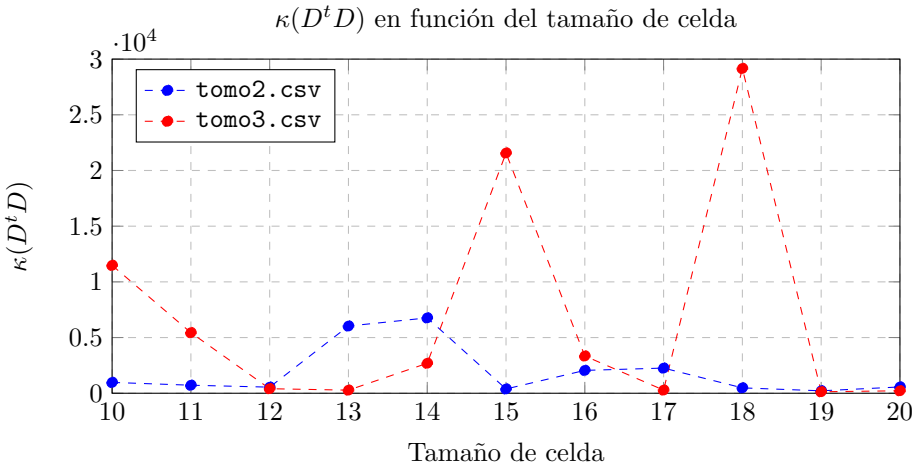
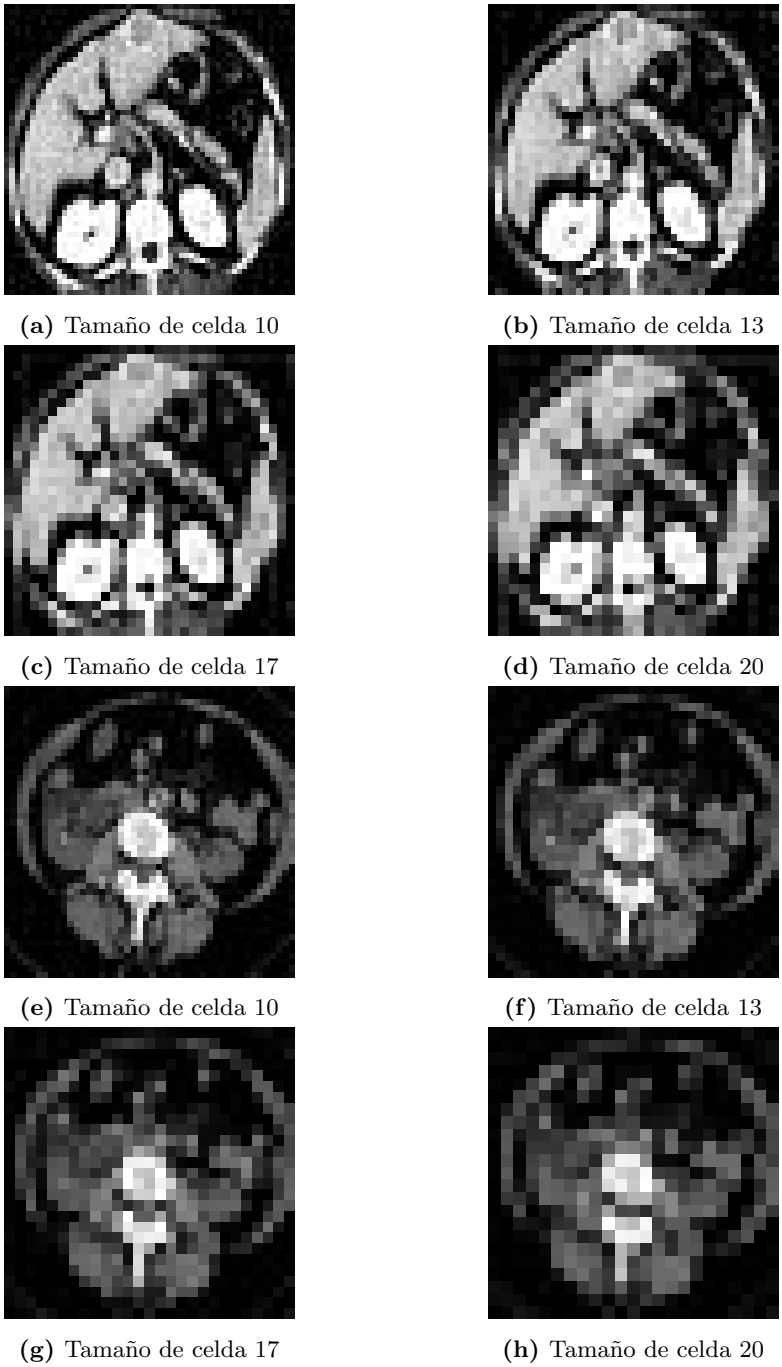


Figura 5: $\kappa(D^tD)$ en función del tamaño de celda, para las imágenes `tomo2.csv` y `tomo3.csv`. Nivel de ruido 0.0, 20000 rayos aleatorios.



(a) Tamaño de celda 10

(b) Tamaño de celda 13

(c) Tamaño de celda 17

(d) Tamaño de celda 20

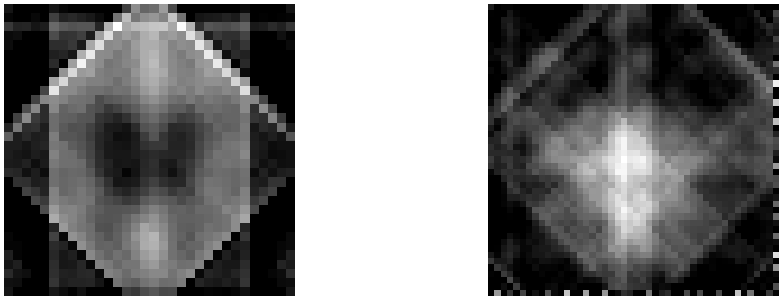
(e) Tamaño de celda 10

(f) Tamaño de celda 13

(g) Tamaño de celda 17

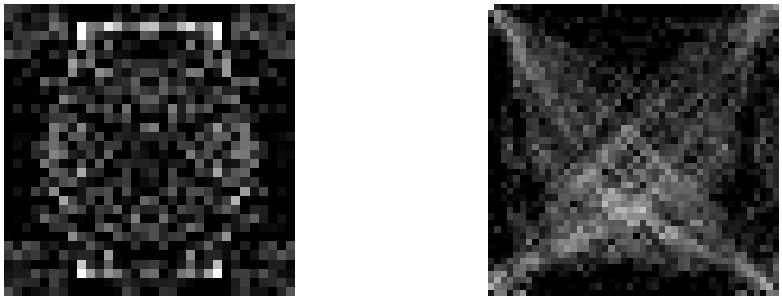
(h) Tamaño de celda 20

Figura 6: Imágenes `tomo2.csv` y `tomo3.csv` reconstruidas con diferentes tamaños de celda, nivel de ruido 0.0 y 20000 rayos aleatorios.



(a) `phantom.csv`, con tamaño de celda 8 **(b)** `tomo3.csv`, con tamaño de celda 10

Figura 7: Imágenes `phantom.csv` y `tomo3.csv` reconstruidas con rayos horizontales, verticales y diagonales. Nivel de ruido 0.0.



(a) `phantom.csv`, con tamaño de celda 8 **(b)** `tomo3.csv`, con tamaño de celda 10

Figura 8: Imágenes `phantom.csv` y `tomo3.csv` reconstruidas con rayos que barren la imagen desde las cuatro esquinas. Nivel de ruido 0.0.

Figura 9: Imagen `tomo.csv` reconstruida con el método "completo", tamaño de celda 2 y nivel de ruido 0.0.

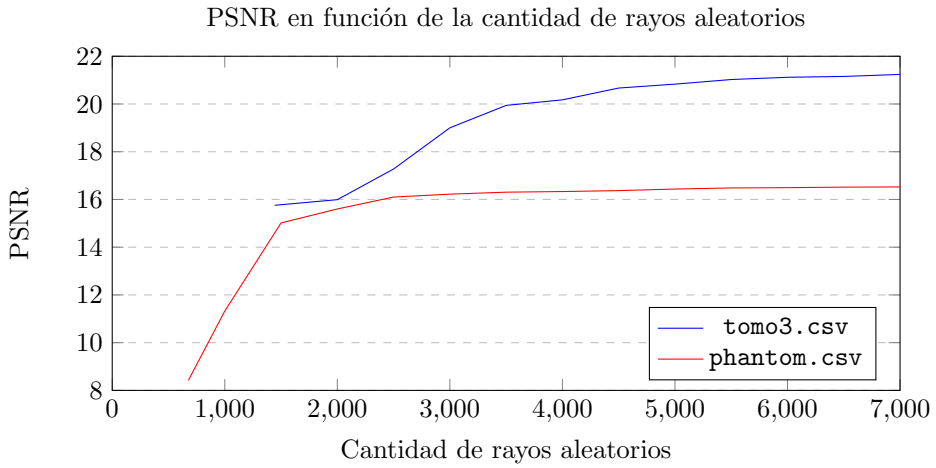


Figura 10: PSNR en función de cantidad de rayos aleatorios, para las imágenes `tomo3.csv` y `phantom.csv`, tamaños de celda 12 y 10 respectivamente, y nivel de ruido 0.0.

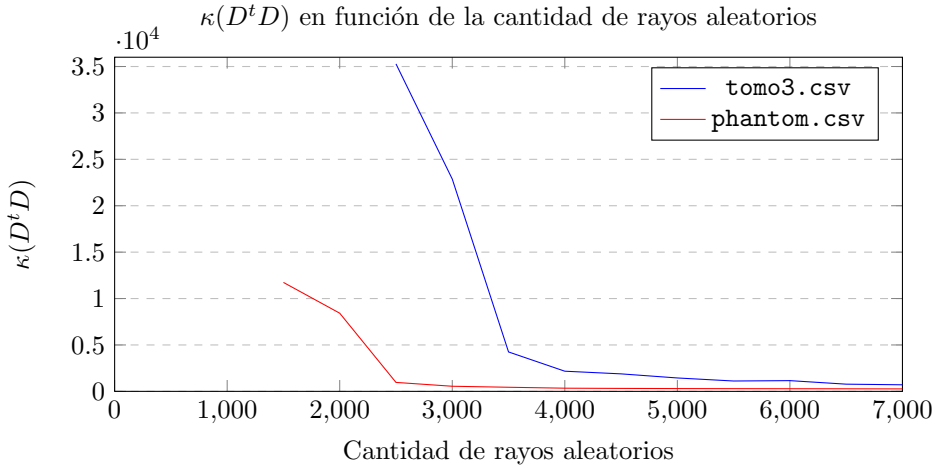
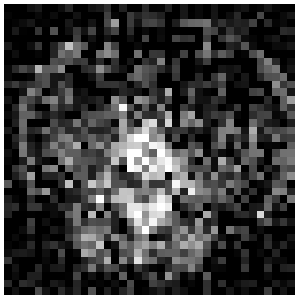
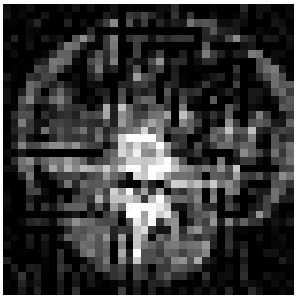


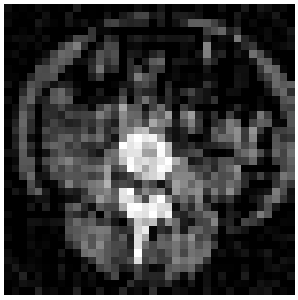
Figura 11: $\kappa(D^tD)$ en función de cantidad de rayos aleatorios, para las imágenes `tomo3.csv` y `phantom.csv`, tamaños de celda 12 y 10 respectivamente, y nivel de ruido 0.0.



(a) 1444 rayos aleatorios (D resulta cuadrada).

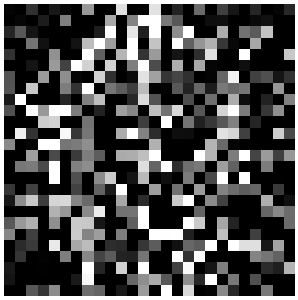


(b) 2500 rayos aleatorios.

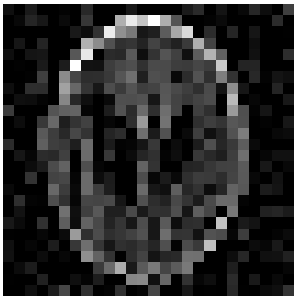


(c) 4000 rayos aleatorios.

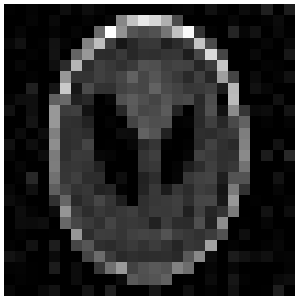
(d) 7000 rayos aleatorios.



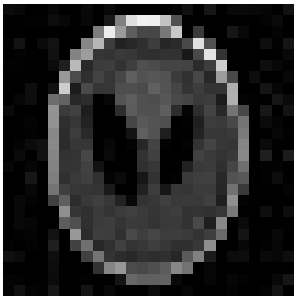
(e) 676 rayos aleatorios (D resulta cuadrada).



(f) 1500 rayos aleatorios.



(g) 4000 rayos aleatorios.



(h) 7000 rayos aleatorios.

Figura 12: Imágenes `tomo3.csv` y `phantom.csv` reconstruidas con tamaños de celda 12 y 10 respectivamente, nivel de ruido 0.0. Se muestran resultados para diferentes cantidades de rayos aleatorios.

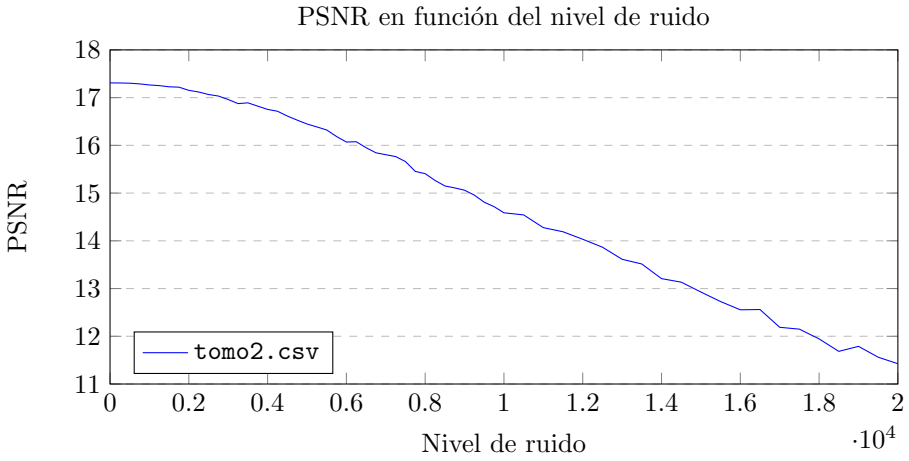


Figura 13: PSNR en función del nivel de ruido agregado a los tiempos de los rayos, para la imagen `tomo2.csv`, tamaño de celda 10 con 20000 rayos aleatorios.

(a) Nivel de ruido: 0.

(b) Nivel de ruido: 4000.

(c) Nivel de ruido: 8000.

(d) Nivel de ruido: 12000.

(e) Nivel de ruido: 16000.

(f) Nivel de ruido: 20000.

Figura 14: Imagen `tomo2.csv` reconstruida con tamaño de celda 10 y 20000 rayos aleatorios. Se muestran resultados para diferentes niveles de ruido.

4. Discusión

4.1. Tamaño de celda

Tal como se esperaba, en la Figura 4 observamos que el PSNR es mayor cuando las celdas son más chicas, o dicho de otra manera, cuanto las imágenes originales son divididas en más celdas. Esta propiedad es independiente de la imagen en cuestión, puesto que lo observamos tanto para `tomo2.csv` como para `tomo3.csv`. De más está decir que la Figura 6 muestra claramente las diferencias de calidad.

Un aspecto positivo es que a mayor tamaño de celda la reconstrucción es ciertamente de menor calidad, pero sin embargo la apariencia, a grandes rasgos, de la imagen reconstruida no cambia. Podría haber sucedido que al aumentar levemente el tamaño de celda la distribución de las intensidades cambiara, generando que zonas que antes eran oscuras ya no lo sean y viceversa; sin embargo no es este el caso, y es por esto que el PSNR tampoco disminuye abruptamente. Este resultado está fuertemente ligado al método de generación de rayos utilizado, que estudiaremos posteriormente.

En la Figura 5 observamos que no parece haber correlación entre el número de condición de D^tD y el tamaño de las celdas. Muy al contrario, el valor de $\kappa(D^tD)$ tiene "saltos" repentinos cuando vemos distintos tamaños de celda. Podemos pensar que dicho parámetro no influye fuertemente en la forma que adquirirá la matriz (a excepción de su tamaño, por supuesto) y en su proximidad a una matriz singular, a diferencia de la forma de generación de rayos, que como veremos posteriormente, juega un rol importante en este aspecto.

4.2. Rayos verticales, horizontales y diagonales

En la Figura 7 observamos con claridad que las reconstrucciones están lejos de ser aceptables. En cada caso la distribución de intensidades respeta en cierto grado la de la imagen original, en el sentido de que, por ejemplo, para `tomo3.csv`, el área del centro es notablemente más clara que el resto, tal como en la imagen real. Sin embargo no se logran recuperar las formas de cada parte de la imagen ya que las reconstrucciones son muy difusas.

Si prestamos más atención se observa la aparición de ciertas rectas difusas con direcciones verticales, horizontales y diagonales de pendiente uno, tal como los rayos generados, y que parecerían seccionar la imagen. Podemos concluir que, debido a cómo armamos los rayos, solo se detectan cambios abruptos de las intensidades de los píxeles en estas tres direcciones, y es esa la razón por la cuál observamos estas rectas. Dicho de otra manera, al irradiar la imagen en solo tres direcciones no estamos capturando la variación de densidad en otras direcciones.

Esto constituye un claro indicio de que los mejores métodos de generación de rayos serán aquellos que generen la mayor variedad de direcciones, siempre y cuando se logren atravesar la mayor cantidad de píxeles posibles.

La diferencia de calidad puede corroborarse con el PSNR obtenido. Por ejemplo, si miramos el PSNR obtenido al reconstruir `tomo3.csv` con tamaño de celda 10 y rayos aleatorios, en el gráfico de la Figura 4, vemos que que está entre 21 y 22, mientras que el PSNR obtenido con rayos verticales, horizontales y diagonales es 17.0948.

Para `phantom.csv` y `tomo3.csv` la cantidad total de rayos generados en este caso fueron 1530 y 2706 respectivamente, que son mayores a la cantidad de celdas, 1024 y 2116. Sin embargo, nuestra implementación del método de la potencia devolvió únicamente 183 y 283 autovalores y autovectores para la matriz D^tD. Lo que sucede es que D^tD no tiene rango completo en ninguno de los dos casos y por lo tanto $\kappa(D^tD)$ no está definido.

Para corroborar esto se guardó la matriz D^tD de `phantom.csv` en un archivo aparte, y con un script de Python se obtuvieron sus autovalores ordenados con la función `eigh` de la biblioteca Numpy. En efecto, se comprobó que el autovalor número 183 era `2.79957053e+02`, mientras que el siguiente era `16308375e-11`, que es tan bajo que puede considerarse cero. Esto no fue exactamente así para el caso de

`tomo3.csv`, dado que se pudo comprobar que existían unas decenas más de autovalores no nulos, y que nuestra implementación falló en calcular. Sin embargo, la mayoría de los autovalores de D^tD en este caso también resultaron nulos, por eso es que tampoco tiene rango completo.

Concluimos entonces que esta forma de emitir rayos es propensa a generar una matriz D^tD singular, lo cual implica que las soluciones obtenidas serán muy inestables.

4.3. Rayos que barren la imagen

Este caso es muy similar al anterior, en el sentido de que los fenómenos observados son los mismos. En la Figura 8 nuevamente observamos resultados inaceptables, y también vemos, sobretodo para `tomo3.csv`, rectas difusas que se asemejan a los rayos generados.

En este caso podemos afirmar que sí existen más direcciones que en el caso anterior, donde sólo había tres. Sin embargo, todos los rayos parten de cuatro posibles puntos, que son las esquinas de la imagen, y esto genera que la variedad de direcciones sea insuficiente para cada celda (por ejemplo, no existe ningún rayo vertical y horizontal que divida la imagen en dos partes). Esto explica los defectos observados.

Los valores de PSNR son incluso peores que los obtenidos con rayos verticales, horizontales y diagonales, marcando una desventaja frente al caso anterior.

Para `phantom.csv` no se pudieron calcular todos los autovalores, solo se calcularon 790. Con Numpy se verificó que el número de condición de D^tD es muy alto, 1578918. Para `tomo3.csv` solo se obtuvieron 1088 autovalores, y también se verificó el número de condición con Numpy, que resultó ser 6470105.

Si bien las matrices D^tD no resultaron singulares como antes, $\kappa(D^tD)$ resulta muy alto y por ende las soluciones son más inestables.

4.4. Rayos aleatorios y método completo

En la Figura 10 observamos que el PSNR obtenido mejora al aumentar la cantidad de rayos aleatorios, para ambas imágenes. Para `tomo3.csv` el PSNR obtenido con 7000 rayos es prácticamente igual al obtenido con 20000 rayos, como se observa en la Figura 4. Esto indica que la velocidad de crecimiento del PSNR disminuye a medida que la cantidad de rayos aleatorios aumenta.

En la Figura 12 vemos cómo aumenta la calidad, según la propia percepción humana, a mayor cantidad de rayos. En las reconstrucciones defectuosas se puede notar un cierto efecto de ruido, es decir, hay píxeles que resultan muy blancos y otros muy oscuros, sin seguirse ningún patrón en particular. Lo importante es que casi todas las imágenes (siendo la reconstrucción de `phantom.csv` con 676 rayos una clara excepción) se asemejan mucho a la original, principalmente porque no encontramos los defectos que surgen con los otros dos métodos de generación de rayos. No solo coinciden en gran medida las intensidades, si no que las formas y los bordes de cada parte de la imagen original son recuperados con considerable precisión.

Esto se debe a que ahora sí resulta que cada celda es atravesada por muchos rayos en diferentes direcciones. Esta variedad en los recorridos de los rayos permite capturar la mayor información de la imagen original, teniendo en cuenta las variaciones de las intensidades de los píxeles en todas las direcciones. Por supuesto, si no proporcionamos la cantidad suficiente de rayos, la reconstrucción será más defectuosa, tal como pudimos observar. Pero estos defectos no son más que intensidades erróneas en píxeles aleatorios de la imagen.

Otro resultado que mejora cuando aumenta la cantidad de rayos es el número de condición de D^tD. En la Figura 11 podemos observar que disminuye abruptamente al principio, sin embargo, una vez que empezamos a generar más de 4000 rayos parece estabilizarse, es decir, la velocidad de decrecimiento tiende a ser cero. Por lo tanto podemos concluir que a mayor cantidad de rayos aleatorios, más estable será la solución obtenida, indepenidentemente del nivel de ruido utilizado.

En la Figura 9 podemos ver la imagen `tomo.csv` reconstruida con el método completo de generación de rayos. En este caso todo rayo que viaje de un lado de la imagen al lado opuesto es generado. Es

claro que el requisito que ya sabemos que debe cumplirse —que cada celda sea atravesada por muchos rayos con diversas direcciones— se cumple para este método. Y es natural que la imagen obtenida sea bastante precisa, como con el método aleatorio. Sin embargo, la clara desventaja es que la cantidad de rayos a generar puede llegar a ser masiva con una imagen como `tomo2.csv` o `tomo3.csv`. Esto implica serias consecuencias en cuanto al tiempo de procesamiento y a la memoria utilizada por el programa. Por eso es que concluimos que el método aleatorio es superior, con el cual se pueden obtener resultados casi idénticos con muchos menos rayos. Además, el número de condición $\kappa(D^tD)$ en este caso resultó ser 10052, un valor bastante alto. De todas formas sería necesario probar este método con imágenes de diferentes tamaños para poder afirmar algo sobre cómo influye en el número de condición, pero resulta muy complicado debido a los inconvenientes recién mencionados.

4.5. Nivel de ruido

En la Figura 13 observamos que el PSNR empieza variando un poco para los primeros niveles de ruido, sin embargo, luego del nivel 3000 aproximadamente, el PSNR comienza a decrecer linealmente, indicando que las reconstrucciones son cada vez más erróneas.

Según los resultados de la Figura 14, el efecto producido es similar al que observamos cuando probamos reconstrucciones con escasos rayos aleatorios, a pesar de que ahora estamos estudiando un parámetro completamente diferente y que no influye en la matriz D. Vemos que a medida que el nivel de ruido aumenta surgen píxeles distribuidos uniformemente en toda la imagen cuyas intensidades no coinciden con las de los píxeles que los rodean.

Esto era un comportamiento esperable dado que cada tiempo de rayo t_k viene determinado por la suma de las intensidades de todos los píxeles que atravesó. Si perturbamos un t_k de manera aleatoria estaremos de alguna manera indicando que las intensidades de los píxeles atravesados por el rayo en cuestión son distintas de las de la imagen original, y como estas perturbaciones son independientes entre sí, no hay razón para creer que la imagen reconstruida cambiará de una manera particular. Al contrario, lo que sucede es que las intensidades de píxeles individuales cambian aleatoriamente.

Durante esta experiencia observamos qué tan grandes resultaron ser los tiempos t_k. Algunos resultaron un poco mayores a 100000, mientras que otros rondaban 30000. En cualquier caso, es claro que un nivel de ruido como 20000 es un nivel muy alto para este caso, puesto que en relación con los t_k originales el cambio puede ser muy drástico. Sin embargo, en la imagen reconstruida no se perdió completamente la esencia de la imagen original. Para niveles de ruido más bajos, como 4000, que resultan más razonables si pensamos en mediciones reales, el impacto del ruido es mínimo.

Lo que influye notoriamente en este aspecto es el número de condición de D^tD, puesto que dicho valor determina qué tanta variación habrá entre dos soluciones s_1 y s_2 de los sistemas $D^tDs = D^tt_1$ y $D^tDs = D^tt_2$ respectivamente, siendo t_1 cercano a t_2. Notemos que ambos sistemas son ecuaciones normales asociadas a dos problemas de cuadrados mínimos $Ds = t_1$ y $Ds = t_2$. Por lo tanto, con un número de condición de D^tD muy alto las soluciones al problema de cuadrados mínimos típicamente serán muy diferentes ante pequeñas perturbaciones en los tiempos. Ya vimos que con el método de generación de rayos aleatorios los valores de $\kappa(D^tD)$ tienden a ser más bajos que con otros métodos, y gracias a eso podemos observar en este caso que con niveles de ruido bajos las intensidades de los píxeles en la imagen reconstruida no cambian abruptamente.

5. Conclusión

En primer lugar, es necesario remarcar que las tomografías computadas que se realizan a diario involucran un proceso muchísimo más complejo que el descripto en este trabajo. Por lo tanto, estableceremos las conclusiones del mismo entendiendo que el objetivo de los métodos propuestos es simular la emisión de rayos X sobre una imagen de manera simplificada, y con los datos obtenidos reconstruir la misma con la mayor precisión posible.

En este sentido, es claro que la manera en que se emiten los rayos resulta decisiva a la hora de evaluar qué tan buena es la reconstrucción. Vimos que el método de generación de rayos aleatorios resultó ser aquel que genera el sistema de ecuaciones más representativo, en comparación con los otros métodos propuestos. El hecho de que las celdas de la imagen sean atravesadas por varios rayos y en diversas direcciones resultó ser lo que diferencia a este método del resto, ya que permitió obtener reconstrucciones donde se preserva bien la distribución de las intensidades de los píxeles y las formas de las distintas partes de la imagen original.

El tamaño de celda elegido para la discretización es sin dudas un parámetro decisivo. Celdas muy chicas garantizan una mejor calidad de la imagen generada, pero se debe decidir hasta qué punto uno está dispuesto a sacrificar velocidad de procesamiento por calidad.

La descomposición SVD de D resulta apropiada para hallar fácilmente la solución de cuadrados mínimos del sistema $Ds = t$, puesto que no implica que se deban realizar cálculos numéricamente inestables, suponiendo que ya contamos con la descomposición. Sin embargo, es necesario contar con un método eficaz para la obtención de los autovalores y autovectores de D^tD (o DD^t). El método de la potencia con deflación no es la mejor opción ya que no solo es poco eficiente en cuanto a tiempo de procesamiento, si no que además requiere el cumplimiento de ciertas hipótesis y puede haber un considerable arrastre de error en los cálculos, debido a los problemas comunes de la aritmética finita. En el desarrollo del trabajo propusimos un método que logra en cierta medida paliar estos problemas que pueden surgir, sin descuidar el rendimiento, sin embargo está lejos de ser perfecto.

A continuación enunciamos posibles extensiones que podrían hacerse:

- Obtención de autovalores y autovectores con otro algoritmo.
- En vez de utilizar la descomposición SVD, hallar la solución de cuadrados mínimos resolviendo las ecuaciones normales mediante las factorizaciones QR y Cholesky de D^tD, los métodos iterativos de Jacobi y Gauss-Seidel, y eliminación gaussiana.
- Nuevos métodos de generación de rayos.

Para finalizar, concluimos que teniendo un buen método de generación de rayos, la aproximación de la solución mediante cuadrados mínimos constituye un método muy efectivo que permite obtener reconstrucciones satisfactorias de la imagen original.

6. Apéndice

A: Enunciado del TP

Métodos Numéricos
2^o Cuatrimestre 2018
Trabajo Práctico 3

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

Tomografía computada

Introducción

El objetivo del trabajo práctico es evaluar un método para reconstruir imágenes tomográficas sujetas a ruido, utilizando aproximación por cuadrados mínimos.

En muchos ámbitos de la medicina se trabaja con estudios que buscan obtener los mejores resultados posibles con instrumentos que poseen error. Ejemplo de esto son las tomografías computadas, donde se atraviesa el objeto de estudio con rayos X. Luego, según la proporción de la radiación que llega al otro lado, se consigue estimar la densidad del objeto atravesado.

Dado que cada envío de un rayo atraviesa una sección distinta del objeto se realizan múltiples disparos con diferentes ángulos para tratar de capturar todas las partes del objeto. Como además los instrumentos tienen errores de medición es recomendable realizar repetidas veces el mismo envío para tratar de disminuir los efectos negativos asociados. Dada una cierta granularidad elegida para representar la imagen se tomará luego una serie de mediciones con los rayos y se determinará un sistema de ecuaciones. La particularidad de dicho sistema es que será sobredeterminado, debido a que se toman más mediciones que cantidad de incógnitas. Para conocer los niveles de densidad en cada parte del objeto en la discretización basta entonces resolver este sistema, para lo que se usarán los métodos numéricos vistos en la materia.

Así como en muchas aplicaciones, no siempre es posible contar con datos reales ya sea porque son difíciles o caros de conseguir o, como en este caso, porque no es razonable realizar sucesivas tomografías sobre un individuo por la cantidad de radiación que podría recibir. En este trabajo realizaremos una *simulación* sobre una imagen sintética que consideraremos proveniente de un tomógrafo lo que nos permitirá además juzgar que tan buena es la aproximación obtenida.

Método de reconstrucción

El análisis tomográfico de una sección (que suponemos bidimensional y cuadrada) de un cuerpo consiste en emitir señales de rayos X que lo atraviesan en diferentes direcciones, midiendo el tiempo[1] que tarda cada uno en atravesarlo.

Supondremos que el cuerpo se discretiza en $n \times n$ celdas cuadradas. Si d_{ij}^k es la distancia que recorre el k-ésimo rayo en la celda ij (notar que $d_{ij}^k = 0$ si el rayo no pasa por esta celda) y v_{ij} es la velocidad de la señal del rayo en esa celda, entonces el tiempo de recorrido de la señal completa es:

$$t_k = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}^k v_{ij}^{-1}. \tag{1}$$

Como resultado del análisis tomográfico, se tienen mediciones del tiempo que tardó cada señal en recorrer el cuerpo. Si se emitieron m señales, entonces el resultado es un vector

[1] Esto es una simplificación, en realidad se mide la intensidad de los rayos.

$t \in \mathbb{R}^m$, tal que t_k indica el tiempo de recorrida de la k-ésima señal. Sea $D \in \mathbb{R}^{m \times n^2}$ una matriz cuyas filas se corresponden con las m señales y cuyas columnas se corresponden con las n^2 celdas de la discretización del cuerpo de forma vectorizada, y tal que la fila k contiene los valores d_{ij}^k en las columnas correspondientes a cada celda. Entonces, las velocidades de recorrida originales v_{ij}^{-1} se pueden reconstruir resolviendo el sistema de ecuaciones $Ds = t$. El vector solución $s \in \mathbb{R}^{n^2}$ contiene los valores inversos de las velocidades originales en cada celda que representarán las densidades en cada punto del cuerpo escaneado. Notar que estos valores serás las incógnitas de nuestro problema y por lo tanto no hará falta invertirlos.

Un problema fundamental que debe considerarse es la presencia de errores de medición en el vector t de tiempos de recorrido. Dado que el sistema estará en general sobredeterminado, la existencia de estos errores hará que no sea posible encontrar una solución que satisfaga al mismo tiempo todas las ecuaciones del sistema. Para manejar este problema, el sistema $Ds = t$ se resuelve utilizando el método de aproximación por *cuadrados mínimos*, obteniendo así una solución que resuelve de forma aproximada el sistema original.

Proceso de simulación tomográfico

Debido a la imposibilidad de contar con un tomógrafo real y acceder a los datos crudos del mismo, realizaremos un proceso de simulación tomográfico el cual consistirá en generar datos de prueba partiendo de imágenes sintéticas que consideraremos reales, es decir, serán la sección de nuestro objeto a escanear[2]. Para los fines de este procedimiento, se considerará que el valor de cada píxel de la imagen corresponde al dato real de densidad del cuerpo, es decir, a la velocidad (inversa) de recorrida de las señales de rayos X al atravesar ese píxel.

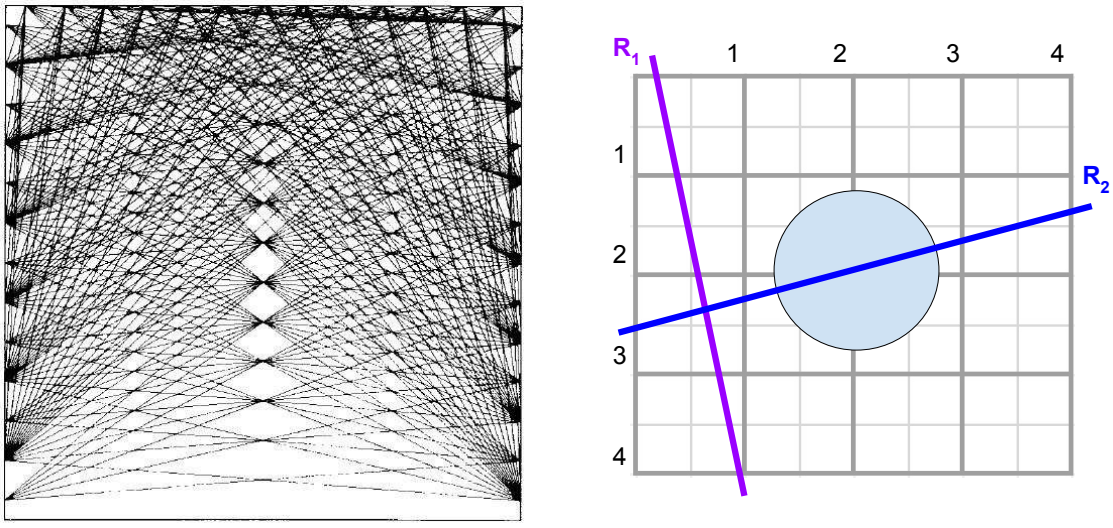
Generación de rayos

Parte del proceso de simulación tomográfico, consiste en generar rayos sobre la imagen real o cuerpo a escanear de distinta forma. Distintos tomógrafos generan rayos de diversas formas, algunos generan rayos de forma axial, otros rayos paralelos, etc. En la Figura 1(a) se muestra una posible distribución de estos rayos.

Una vez definida la forma de generar rayos, se debe definir la discretización del espacio a escanear. Este modelo así definido permite representar muchos tipos de tomógrafos distintos con sólo variar la forma de los rayos y la discretización. Dado que poseemos la imagen real sobre la cual realizaremos nuestra simulación, primero generaremos los rayos sobre ella para poder medir el tiempo t_k de cada rayo k y la longitud del mismo rayo en cada celda de la grilla discretizada d_{ij}^k.

Veamos un ejemplo, supondremos que tenemos una imagen de real de tamaño 8×8 y la discretizamos en una grilla de 4×4 (ver Figura 1(b)). Para los rayos R_1 y R_2, obtendremos los valores $t_1 = 0$ y $t_2 = 15$ resultado de la suma de las intensidades de los píxeles que el rayo R_1 y R_2 toca en la imagen original, respectivamente. Luego, obtendremos los siguientes valores de distancias para cada rayo (donde los que no se indican son iguales a cero). Rayo R_1: $d_{11}^1 = 2$, $d_{21}^1 = 3$, $d_{31}^1 = 2$ y $d_{41}^1 = 2$. Rayo R_2: $d_{31}^2 = 3$, $d_{32}^2 = 2$, $d_{23}^2 = 2$ y $d_{24}^2 = 3$.

[2]Con el presente trabajo se adjuntan imágenes de prueba, sin embargo cada grupo puede proponer las suyas.



(a) (b)

Figura 1: Ejemplo de generación de señales de rayos X.

Procedimiento

1. Leer desde un archivo una imagen con los datos reales de un objeto de estudio.
2. Definir un tamaño de discretización $n \times n$ para la imagen a reconstruir.
3. Generar un conjunto relevante de rayos y calcular sus tiempos de recorrida exactos.
4. Perturbar los tiempos de recorrida con ruido aleatorio.
5. Guardar para cada rayo los tiempos t_k y los valores d_{ij}^k
6. Ejecutar el método propuesto en la sección anterior sobre los datos perturbados, con el objetivo de reconstruir el cuerpo original. La imagen resultante se debe guardar en un archivo de salida.

Enunciado

El trabajo práctico consiste en implementar un programa en `C++` que simule el proceso de tomografía y reconstrucción. Para el mismo, se **deberán** generar datos de prueba simulando el proceso de tomografía computada descripto en el presente trabajo práctico. Dado que el proceso consiste en incorporar ruido a las mediciones, se desea evaluar la calidad de reconstrucción comparada con la imagen real.

El programa **deberá** tener como parámetros (al menos) los nombres de los archivos de la imagen de entrada (a utilizar en la simulación) y de salida de la imagen reconstruida, junto con un parámetro que permita especificar el nivel de ruido a introducir en la imagen. La cantidad y tipo de parámetros puede modificarse por cada grupo.

Se **deben** probar con distintas estrategias geométricas para generar las señales de rayos X, y distintas discretizaciones. Buscar la que mejor reconstrucción posea[3]. Ejemplo de generación de rayos pueden ser: paquetes de señales radiando de puntos fijos o puntos de inicio móviles, señales partiendo de los cuatro lados de la imagen o sólo de algunos lados, variación de rango de ángulos de salida de las señales, de forma aleatoria, etc. Reportar también los casos fallidos y justificar las causas de fallo.

Para resolver cuadrados mínimos de **debe** utilizar la descomposición SVD.

Además, se **debe** medir el número de condición κ_2 de la matriz asociada al sistema de ecuaciones normales $D^tDx = D^tt$ que resuelve el problema de cuadrados mínimos, para analizar la estabilidad de la solución. **Definir** condiciones para que D^tD sea SDP y posea un número de condición bajo. **Justificar** teóricamente y corroborar experimentalmente.

Por último, este trabajo **tendrá** una presentación oral frente al curso que será evaluada como una parte adicional de la nota. Para la misma, cada grupo diseñará una presentación con diapositivas incluyendo los desarrollos y resultados que considere interesantes del propio informe, y dispondrá de 15 minutos para exponerlo. Ver sección "Pautas para la exposición oral".

Experimentación

La particularidad de este trabajo práctico es que cada grupo **debe** proponer nuevas experimentaciones basadas en lo pedido en la sección de Enunciado y lo que el grupo considere relevante. Para guiar la experimentación, a continuación se proponen algunas líneas de trabajo las cuales no son exhaustivas pero que pueden servir como disparadores de nuevas experimentaciones.

Se desea analizar el impacto del nivel y tipo de ruido introducido en función de la calidad de reconstrucción. También, la discretización utilizada jugará un rol importante en la calidad de reconstrucción y en el tiempo de procesamiento.

Para medir el error de la imagen reconstruida se propone utilizar el PSNR, definido como:

$$\texttt{PSNR} = 10 \cdot \log_{10}\left(\frac{\texttt{MAX}_u^2}{\texttt{ECM}}\right)$$

donde $\texttt{MAX}_u$ define el rango máximo de la imagen y $\texttt{ECM}$ es el *error cuadrático medio*, definido como:

$$\frac{1}{N}\sum_{i,j}(u_{ij}^0 - u_{ij})^2$$

donde N es la cantidad de píxeles de la imagen, u^0 es la imagen *ideal* y u es la imagen que reconstruimos. Tener en cuenta que previo a aplicar el PSNR ambas imágenes deben poseer las mismas dimensiones. Si bien esta métrica se condice en gran medida con la calidad percibida por las personas, una variación homogénea de las intensidades en los píxeles de la imagen reconstruida, producirá altos valores de ECM y por ende bajos valores de PSNR. Para lidiar con este problema se sugiere utilizar de forma complementaria alguna métrica que considere errores *normalizados*.

[3]Para facilitar la implementación de esta función se adjunta un código Matlab/Octave de ejemplo.

Analizar el condicionamiento de la matriz y su relación con la forma y cantidad de rayos generada.

Puntos opcionales no obligatorios

Utilizar alguna de las siguientes opciones para resolver cuadrados mínimos y comparar con el método requerido:

1. Factorización QR.
2. Cholesky sobre las ecuaciones normales.
3. Gauss-seidel sobre las ecuaciones normales.

Notar que en el procedimiento de simulación descripto anteriormente, la alteración de los tiempos de recorrida solamente afecta al término independiente en el sistema de ecuaciones normales. Por la tanto, teniendo la factorización LU, es posible experimentar con muchos niveles de ruido de forma más eficiente. Mencionar en qué casos esto es conveniente.

En caso de que el sistema tenga un número de condición alto, se pueden intentar esquemas de regularización para mejorar la estabilidad de la solución obtenida.

Pautas para la exposición oral

Como se mencionó anteriormente, además del informe del trabajo práctico, se deberá crear una presentación con diapositivas la cual cada grupo deberá exponer en a lo sumo 15 minutos en la fecha indicada por los docentes. La presentación y exposición será evaluada y formará parte de la nota del trabajo práctico. Para la misma se deberán seguir las siguientes pautas:

- No sobrepasar los **15 minutos** de exposición.
- La exposición es requisito para aprobar el TP3 pero la misma no implica garantía de aprobación de todo el trabajo práctico.
- La exposición puede ser de la totalidad o de un subconjunto de los integrantes, y esta decisión queda a elección de cada grupo. Una vez finalizada la misma, se llevará a cabo un coloquio donde los integrantes del grupo responderán a las preguntas realizadas.
- Cabe mencionar que los docentes podrán elegir qué alumno debe responder, con lo cual es importante que todos los integrantes estén al tanto de todas las decisiones tomadas.
- Ver consejos para la creación de presentaciones en la sección “Descargas” del sitio de la materia.

Fecha de entrega

- *Formato Electrónico:* domingo 2 de diciembre hasta las 23.59 hs, enviando el trabajo (informe + código) a la dirección `metnum.lab@gmail.com`.

- El subject del email debe comenzar con el texto `[TP3]` seguido de la lista de apellidos de los integrantes del grupo separados por punto y coma `;`.
  Ejemplo: `[TP3] Lennon; McCartney; Starr; Harrison`

- Se ruega no sobrepasar el máximo permitido de archivos adjuntos de 20MB. Tener en cuenta al realizar la entrega de no ajuntar bases de datos disponibles en la web, resultados duplicados o archivos de backup.

- *Formato físico:* no es necesario.
- *Recuperatorio:* jueves 20 de diciembre hasta las 23.59 hs, enviando el trabajo corregido a la dirección `metnum.lab@gmail.com`
- *Exposición oral:* viernes 21 de diciembre desde las 17hs.
- Pautas de laboratorio:
  `https://campus.exactas.uba.ar/pluginfile.php/109008/course/section/16502/pautas.pdf`

Importante: El horario es estricto. Los correos recibidos después de la hora indicada serán considerados re-entrega.

B: Demostraciones relevantes

- Obtención de la solución de cuadrados mínimos del sistema $Ds = t$ mediante la descomposición SVD de D:

  En primer lugar supondremos que $D \in \mathbb{R}^{n^2 \times m}$ es tal que $r = \text{rg}(D) < n^2$.

  Se quiere minimizar $||Ds - t||^2$. Consideremos la descomposición SVD de D, $D = U\Sigma V^t$. Dado que U^t es ortogonal:

  $$||Ds - t||^2 = ||U^t(U\Sigma V^t s - t)||^2 = ||\Sigma V^t s - U^t t||^2$$

  Dado que V^t es inversible (por ser ortogonal), el problema de encontrar s tal que minimice $||\Sigma V^t s - U^t t||^2$ es equivalente a encontrar $y \in \mathbb{R}^{n^2} = \text{Im}(V^t)$ tal que minimice $||\Sigma y - U^t t||^2$ y luego obtener $s = Vy$. Si escribimos:

  $$U^t t = \begin{pmatrix} c \\ d \end{pmatrix}$$

  con $c \in \mathbb{R}^r$ y $d \in \mathbb{R}^{m-r}$, entonces:

  $$||\Sigma y - U^t t||^2 = \left\| \begin{pmatrix} \sigma_1 y_1 \\ \vdots \\ \sigma_r y_r \\ 0 \\ \vdots \\ 0 \end{pmatrix} - \begin{pmatrix} c \\ d \end{pmatrix} \right\|^2 = \left\| \begin{pmatrix} \sigma_1 y_1 - c_1 \\ \vdots \\ \sigma_r y_r - c_r \end{pmatrix} \right\|^2 + ||d||^2$$

  donde $\sigma_1 \dots \sigma_r$ son los valores singulares de D (las entradas positivas de la diagonal principal de Σ). Si tomamos:

  $$y = \begin{pmatrix} c_1/\sigma_1 \\ \vdots \\ c_r/\sigma_r \\ y_{r+1} \\ \vdots \\ y_{n^2} \end{pmatrix}$$

  con $y_{r+1} \dots y_{n^2}$ arbitrarios entonces se minimiza $||\Sigma y - U^t t||^2$. Entonces obteniendo $s = Vy$ logramos minimizar $||\Sigma V^t s - U^t t||^2 = ||Ds - t||^2$.

  Si $r = n^2$ entonces solo podremos obtener un único y y por ende un único s, lo cual tiene sentido puesto que al ser D de rango completo la solución de cuadrados mínimos es única.