DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

Tomografía computada

Métodos Numéricos
Primer Cuatrimestre de 2018

Integrante	LU	Correo electrónico
Serio, Franco Agustìn	215/15	francoagustinserio@gmail.com
Ruiz Ramirez, Emanuel David	343/15	ema.capo2009@hotmail.com
Goldstein, Brian Uriel	27/14	brai.goldstein@gmail.com
Fadel, Uriel	104/14	uriel.fadel@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción Teórica

Palabras Clave: Error Cuadrático Medio - Cuadrados Mínimos Lineales - Discretización **Resumen:** Se propone un método de reconstrucción de imágenes computarizadas. Dada una imagen se trata de producir distintos rayos que pasen en distintos ángulos. Sabiendo por qué píxeles pasa el rayo y su intensidad, podemos obtener el tiempo que le toma al rayo su recorrido por la imagen. A éste tiempo se le agrega ruido para luego recalcular las intensidades de los píxeles por donde pasa cada rayo emitido con el método de cuadrados mínimos lineales.

1.1. Cuadrados Mínimos Lineales

Al tener en general un sistema sobredeterminado, es decir hay mas ecuaciones que incógnitas, se utiliza el método de Cuadrados Mínimos para aproximar la solución. Lo que hace éste método es minimizar la suma de los cuadrados de los residuos, que es la diferencia entre el valor observado y el valor que provee la función, por una familia de funciones, que en el caso de ésta materia, van a ser siempre funciones lineales (pero se podría aproximar por otro tipo de funciones). En el caso de éste trabajo práctico, se usa para aproximar el vector de velocidades, o sea el vector con la intensidad de los píxeles con el vector tiempos ya con ruido. La resolución de cuadrados mínimos lineales se puede hacer usando varios métodos vistos en la materia como descomposición QR, SVD o mediante ecuaciones normales. En este trabajo práctico se usará la última opción.

1.2. Error Cuadrático Medio

Su definición es $\frac{1}{n^2}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$. Mide el promedio de los errores al cuadrado. Éste método lo utilizamos para medir el error entre el vector velocidades y nuestra solución, que es un vector de velocidades v' resultado de $DX = T'$ donde T' es el vector de tiempos con un ruido aleatorio. Desde luego, valores cercanos al cero son mejores. La desventaja de usar éste método suele ser que es muy vulnerable a outliers.

1.3. Discretización

En este trabajo práctico, luego de procesar la imagen dada, se hace una discretización de la misma, que es la subdivisión en una cantidad de celdas determinadas por un $n \in \mathbb{Z}$ que nos viene como parámetro. Dado que la imagen puede no ser cuadrada, lo primero que se hace es hacerla cuadrada tomando el mínimo entre el ancho y el largo de la imagen, para luego dividirla en n^2 celdas.

2. Desarrollo

2.1. Análisis de algoritmos utilizados

2.1.1. darRayos

Esta función establece dos puntos por donde pasará un rayo y esto se repite varias veces para poder crear mas rayos. Estos puntos varían según cual sea la estrategia de lanzamiento:

- Rayos aleatorios: se producen pares de puntos en los diferentes bordes de la imagen.
- Rayos canónicos: se producen todos los puntos de a pares cada punto en bordes opuestos de manera que la recta que pase por ellos tenga pendiente 0. En particular se generan en los bordes superior e inferior, pero si se decide agregar los rayos traspuestos se tendrán los rayos que cruzan del borde izquierdo al derecho.
- Rayos catetos opuestos: se producen todos los puntos de pares, cada uno de los puntos de un par estarán, uno en un borde y el otro en el borde opuesto.En particular se generan en los bordes superior e inferior pero si se decide agregar los transpuestos, se tendrán también los rayos que cruzan de borde izquierdo al derecho.
- Rayos custom: Se puede cargar un archivo CSV con los rayos que el usuario desee (el csv indicara los pares de puntos de comienzo y fin del rayo). También se podrán agregar los rayos traspuestos.

En la siguiente sección se habla de cuantos rayos se lanzan según el tipo elegido.

2.1.2. simulacionRayos

Esta es la principal función en éste trabajo práctico. Se tiene un vector de tuplas que representan los puntos por los que pasan los m rayos. Por cada par de puntos se calculan 2 rayos, el rayo que pasa por esos puntos, y su traspuesto, **si el flag de agregar transpuestos esta en 1**. Esto se hace tratando de imitar la figura 1. Por cada rayo, se llena una fila de la matriz distancias y las nuevas posiciones de cada píxel por donde pasa el rayo en nuestra discretización. También se calcula el tiempo que le toma al rayo ir de un extremo al otro de la imagen, este calculo es la suma de intensidades de los píxeles por los cuales pasa el rayo y a este tiempo se le suma un ruido aleatorio que va desde 0 hasta un valor máximo que se pasa como parámetro por consola.

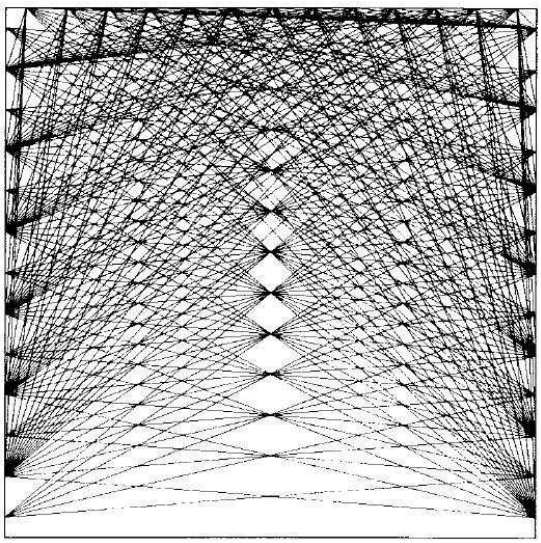


Figura 1: Ejemplo de simulación de rayos.

2.1.3. Cuadrados Mínimos

En la última parte del trabajo práctico lo que se hace es la reconstrucción de la imágen con la técnica de cuadrados mínimos para resolver la ecuación lineal $D^tDx = D^tb$ (éstas son las ecuaciones normales) donde $D \in \mathbb{R}^{mxn^2}$ es nuestra matriz de **distancias**, $x \in \mathbb{R}^{n^2}$ es nuestro vector de **velocidades** (o intensidad de cada píxel) y $b \in \mathbb{R}^m$ que son los **tiempos** que le toma a cada rayo traspasar la imágen. Luego para obtener la solución se utiliza la eliminación Gaussiana con pivoteo que ya hemos usado en el TP1 de la materia.

Uno quisiera que $Dx = b$ se cumpla. Y se supone que dos rayos que pasan por las mismas celdas deberían tardar lo mismo pero esto no necesariamente es así porque pasan por diferentes píxeles de la imagen. Lo cual hace que x no pueda satisfacer dos ecuaciones del sistema, entonces no se tendría una solución. Además al agregar el ruido al vector de tiempos se tendría menos probabilidad de que haya solución. Por estos motivos se usa cuadrados mínimos, es decir para poder aproximar una solución.
Para que tenga sentido el problema, el sistema a resolver por cuadrados mínimos debe estar sobredeterminado, es decir, tener mas filas que columnas. Es por esto que ciertos métodos para generar rayos, como ”rayos canonicos”, no son útiles (por ser en cantidad menos que n^2, el ancho de la matriz).

2.2. Rayos

A continuación se muestran como pasarían los rayos sobre una imagen según los tipos de rayos tratados en este trabajo.

Para el caso de los **rayos aleatorios**, este es el único de los 3 tipos de rayos en donde el usuario puede elegir la cantidad a lanzar.
Si se opta por agregar los transpuestos se generan los rayos transpuestos a los aleatorios originales y el resultado sera una disposición de rayos aleatoria:

Figura 2: Demostración de un lanzamiento de 25 rayos aleatorios en una imagen de 20x20 con n = 4

Los **rayos canónicos** están dados por el n que tomamos del input del usuario. En este caso, hay $2 * n$ rayos que atraviesan la discretización de la imagen. Si bien éstos rayos se plantearon como alternativa una alternativa de lanzamiento de rayos en un primer momento, al tener muy malos resultados y requerimientos muy exigentes con respecto a la matriz a testear, decidimos quitarlos de la experimentación.
Si se elige agregar los transpuestos se verán como se muestra a continuación (sino solo serán las lineas verticales).

Figura 3: Demostración de un lanzamiento rayos canónicos en una imagen de 20x20 con n = 4

El siguiente tipo de rayos(**catetos opuestos**) fue inspirado por el enunciado y trata de barrer la imagen tratando de establecer en los cuatro costados $min(l, a)$ disparadores (donde l es el largo de la imagen y a el ancho, luego de hacer la imagen cuadrada en caso de que no lo sea). Donde por cada disparador se lanzan $min(l, a)$ rayos. En caso de utilizar los traspuestos se ilustran así (en caso contrario serán solo los rayos del borde superior al inferior):

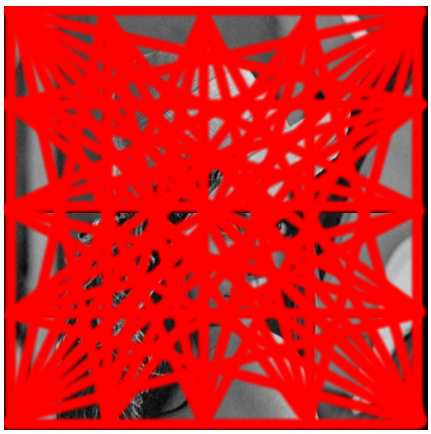


Figura 4: Demostración de un lanzamiento rayos mágicos en una imagen de 20x20 con n = 4

El siguiente tipo de rayos **todos** es similar al anterior pero no solo conecta cada punto con todos los puntos de sus catetos opuestos sino que también conecta con los bordes de los costados. Es decir conecta todos con todos (suponiendo que si incluyen los transpuestos). Este método no esta programado en algoritmo c++ como opción pero si lo simulamos desde python utilizando el método ”rayos custom” que permite leer rayos desde csv.

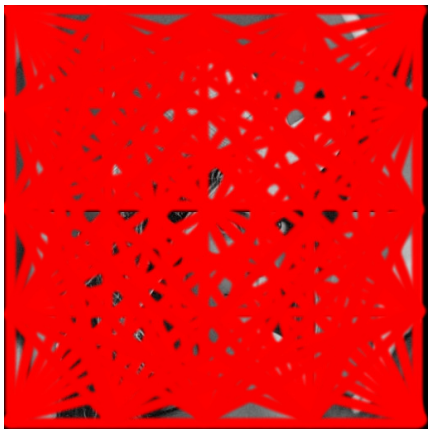


Figura 5: Demostración de un lanzamiento de los rayos ”todos” en una imagen de 20x20 con n = 4 con el flag de **traspuestos** en 1

2.3. Detalles de implementación

Para la implementación de éste trabajo práctico se utilizaron varios fuentes de los dos trabajos anteriores. En particular, la matriz que se usa para guardar las distancias de los rayos es la misma del trabajo práctico dos, como también el algoritmo de backward-substitution para conseguir el vector solución con las velocidades. La estructura de la matriz puede ser la típica, un vector de vector de doubles, ya que los tiempos con ruido pueden tener punto flotante. Con un flag pasado por consola además se puede cambiar esta configuración y usar una matriz esparsa, que es la misma que se uso para el primer TP de la materia (un vector de maps para las filas y un vector de maps para las columnas, donde cada map es una columna y una fila para el primer y segundo caso respectivamente).

A su vez se modifico el archivo de input y output del último trabajo. Las imágenes dicom originales se convirtieron a csv con un conversor en python proporcionado por la cátedra. El csv es un formato fácil de leer y escribir y mantiene los 16 bits que tienen las imágenes dicom. El archivo de salida de nuestro TP también es un archivo CSV.

Por último, el archivo simulacionRayos se encarga de simular la emisión de rayos. Gracias a la función damePuntos obtenemos los píxeles y por consiguiente las celdas por los cuales pasó cada rayo. Este archivo también saca las distancias que recorrió el rayo al pasar por una celda, velocidades del rayo al pasar por un pixel y tiempos que le toma a un rayo pasar por la imagen, para finalmente hacer cuadrados mínimos lineales.

Aparte de poder lanzar los rayos descriptos en la sección anterior, el programa también está preparado para recibir un archivo CSV con rayos customizados por el usuario.

A continuación se muestra un ejemplo de ejecución y se listan los flags utilizados en el programa:

formato de salida; ./nombre_ejecutable **-r** 0.001 **-i** prueba.csv **-s** salida.csv **-tipoRayo** 1 **-transpongoONO** 0 **-usoMatrizEsparsaONO** 0 **-n** 1

- -r es el máximo valor de ruido
- -i es la imagen de entrada
- -s es la imagen de salida
- -tipoRayo es la estrategia de lanzamiento
  - 1 es rayos random (donde se le dará la opción de elegir cuantos rayos quiere lanzar)
  - 2 es rayos canónicos

    - 3 es rayos mágicos
    - 4 es rayos desde csv

- -transpongoONO es si quiere o no trasponer los rayos. 1 es trasponer rayos y 0 es no trasponer rayos.
- -usoMatrizEsparsaONO es si quiere o no usar matriz esparsa. 1 es usar matriz esparsa y 0 es no usar matriz esparsa
- -n es valor de la discretización

3. Experimentación

En esta experimentación se busco investigar principalmente el comportamiento del algoritmo para diferentes valores de **ruido** y **dimensiones de la discretización**. Para darle una interpretación (en sentido de ”calidad”) a los resultados, fue preciso un previo análisis de la métrica propuesta **error cuadrático medio** y su relación con la calidad (aparente y subjetiva) de las reconstrucciones.
Se Analizaron también diferentes **métodos para lanzar rayos** y se estudio como estos métodos influyen en el error cuadrático medio.Para el método (que concluimos) mas exitoso se analizo la relación **cantidad de rayos - error cuadrático medio**. En todos los experimentos los rayos se generaron desde python y fueron enviados por csv al algoritmo implementado en c++ utilizando el método ”rayos custom”, lo que permitió un análisis mas profundo y brindo la posibilidad de experimentar con nuevos métodos no pre-programados.
Se estudio la relación **cantidad de rayos - tiempo** y la relación **dimensiones de discretizacion - tiempo**. Para esto se estudiaron en simultaneo los tiempos de ejecución del algoritmo (en su totalidad) y los tiempos de resolución de sistemas sobre-determinados con por cuadrados mínimos (lineales).
Por ultimo se investigo la posibilidad de reducir los tiempos de computo modificando la estructura de datos utilizada para representar matrices a una implementación para matrices ralas (ya trabajada en un anterior TP).

3.1. Experimento 1

Este primer experimento busca interpretar la métrica **Error Cuadrático Medio (ECM)**. Para poder atribuirle un cierto grado de calidad a cada valor de ECM, decidimos particionar (subjetivamente) todo el espectro de reconstrucciones posibles en 4 diferentes categorías y luego establecer rangos de ECM para cada categoría. Estas son:

- **Categoría A:**
  En esta categoría están las reconstrucciones que consideramos ”de alta calidad” por ser muy similares a la imagen original.Dicho de otro modo, la diferencia entre la reconstrucción y la imagen original, es muy poca.
  Esta categoría permite discernir detalles posiblemente necesarios en una tomografía.

- **Categoría B:**
  En esta categoría están las reconstrucciones que consideramos ”de aceptable calidad”, son similares a la imagen original. Dicho de otro modo, la diferencia entre la reconstrucción y la imagen original, es poca.
  Esta categoría permite discernir algunos detalles, pero es probable que pierda información útil para interpretar la tomografía.

- **Categoría C:**
  En esta categoría están las reconstrucciones que consideramos ”de baja calidad”, son poco similares a la imagen original. Dicho de otro modo, la diferencia entre la reconstrucción y la imagen original, es considerable.
  Esta categoría no permite discernir detalles con claridad, probablemente se pierda mucha información útil para interpretar la tomografía.

- **Categoría D:**
  En esta categoría están las reconstrucciones que consideramos ”de muy mala calidad”, la similitud con la imagen original es poca o nula. Dicho de otro modo, es mucha la diferencia entre la reconstrucción y la imagen original.
  Esta categoría no aporta información suficiente para interpretar la tomografía.

Para establecer los rangos de valores de ECM que corresponden a cada categoría utilizamos el código del experimento 2 (experimento2.py). Este código itera ejecutando el algoritmo variando el ruido y por consiguiente el ECM y guarda las reconstrucciones obtenidas en el directorio /src/Exp1Resultados/ indicando en el nombre de la reconstrucción el ECM obtenido. El mismo código también guarda en el mismo

directorio las diferencias entre las reconstrucciones obtenidas y la imagen original (la diferencia sera una imagen negra si la reconstrucción es idéntica a la imagen original y sino tendrá puntos grises marcando las diferencias).
Lo siguiente que hicimos fue clasificar las reconstrucciones en las 4 categorías basados en una comparación (subjetiva) con la imagen original y basados también en la observación de los archivos de diferencia. El resultado de esta clasificación:

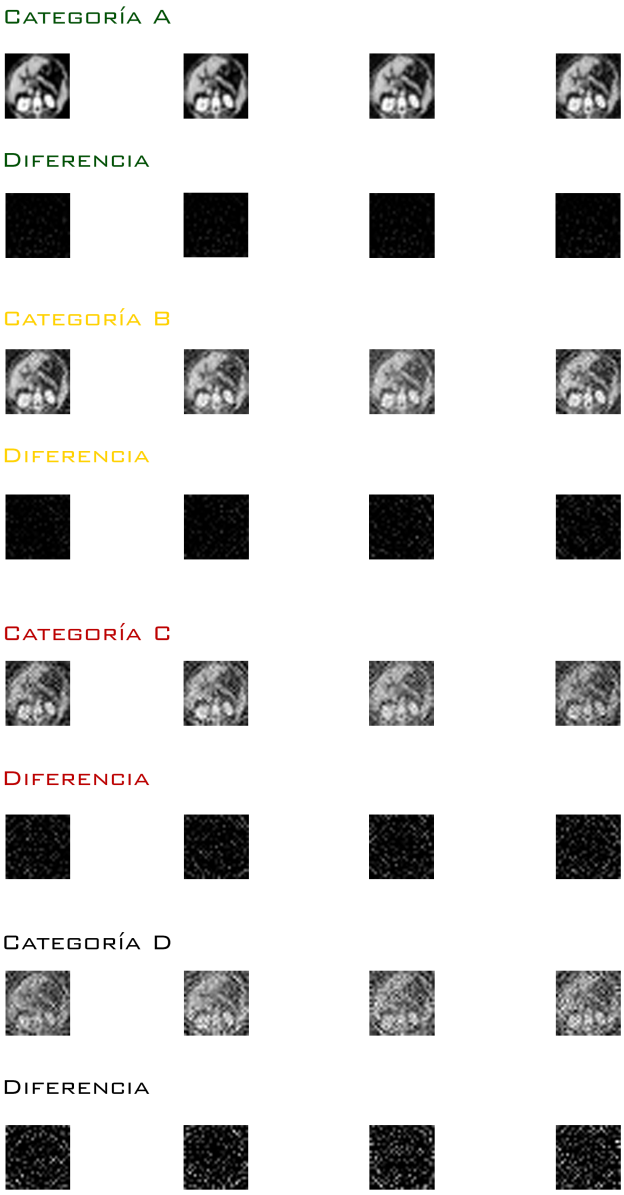


Figura 6: 25x25 a 25

Observaciones:
Observamos en los nombres de archivo de la reconstrucciones (que indican el ECM) en esta clasificación, que aquellas que clasificamos como **Categoría A** tienen $ECM < 200$,aquellas que clasificamos como **Categoría B** tienen $200 < ECM < 1000$, aquellas que clasificamos como **Categoría C** tienen $1000 < ECM < 2000$ y aquellas que clasificamos como **Categoría D** tienen $ECM > 2000$. Observamos como en las imágenes de diferencia, en cada categoría se marcan mas los grises y blancos, indicando la diferencia creciente y por lo tanto la perdida de información en cada categoría. En particular se observa como las diferencias de las categorías A y B son considerablemente mas oscuras que las diferencias en las categorías C y D.

3.1.1. Conclusiones Experimento 1

Concluimos de lo observado que, si una reconstrucción tiene $ECM < 200$ podemos esperar una buena cantidad de detalles y una gran fidelidad con la realidad(del cuerpo humano en estudio). Si el $200 < ECM < 1000$, podrían perderse detalles pero probablemente, se podrá trabajar con esa imagen para diagnostico (en algunos casos). Un ECM mayor a eso, ya sera no-recomendable para su uso en diagnostico.

3.2. Experimento 2

En el experimento 2 se busca analizar la influencia del ruido en la calidad de la reconstrucción (calidad en sentido dado por la métrica ECM).
Para esto se iterara ejecutando el algoritmo en cada iteracion incrementando progresivamente el nivel de ruido. Los rayos se generaran con el método ramdom+transpuestos que tras un breve análisis mostró ser un buen método para generar rayos (esto se analizara en detalle en el experimento 4,en esencia son rayos lanzados al azar desde todos los lados y hacia todas las direcciones).
El nivel de ruido determina el umbral máximo (en modulo) de ruido que se le sumara (o restara) a los tiempos de recorrida de cada rayo (los t_k). El ruido que se sume a cada t_k sera un numero aleatorio en el intervalo $[0, nivelDeRuido]$.
Abusando de la notación utilizamos indistintamente en adelante ”ruido” y ”nivel de ruido”.
Graficaremos entonces el ECM en función del ruido. Graficaremos también para tener de referencia, las constantes que dividen las 4 diferentes categorías (enunciadas en el experimento 1), en verde el limite de la **Categoría A** ($ECM < 200$), en amarillo el limite de la **Categoría B** ($200 < ECM$1000), en rojo el limite de la **Categoría C** ($1000 < ECM < 2000$).

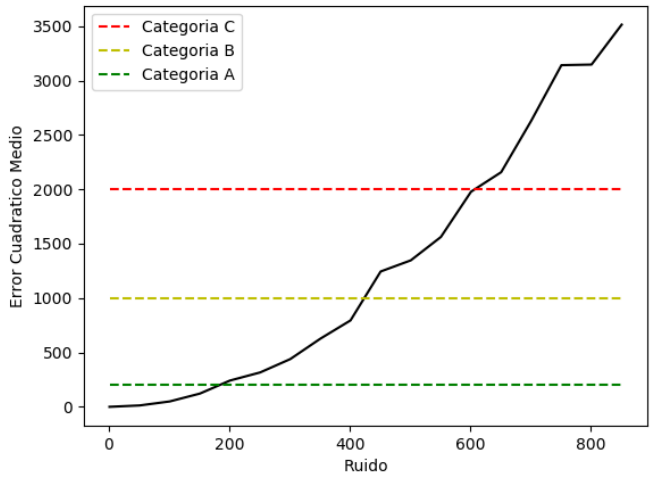


Figura 7: ECM en función del nivel de ruido de la imagen de 25x25 ”tomo25x25.csv” con n-discretizacion=25

Observamos También que cuando el n-discretizacion (la dimensión de la discretizacion, ancho o alto) es igual a dimensión de la imagen, el comportamiento del ECM es ligeramente distinto a cuando el n-discretizacion es menor al ancho (o alto) de la imagen.

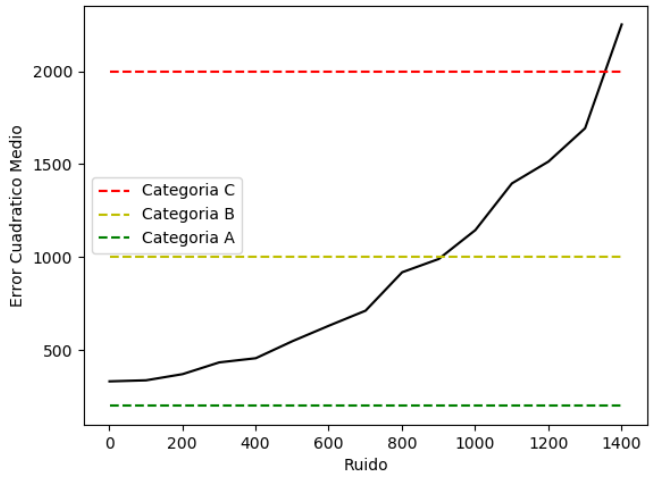


Figura 8: ECM en función del nivel de ruido de la imagen de 50x50 ”tomo50x50.csv” con n-discretizacion=25

Observaciones: se Observa en ambos graficos como el a medida que crece el nivel de ruido, crece el error (ECM).
Se observa como el error es prácticamente 0 para ruidos bajos cuando el n-discretizacion es igual al ancho/alto de la imagen original y Se observa como se mantiene en un rango de categoría A para ruidos relativamente altos (hasta ruido=200 aprox.) y luego en categoría B hasta ruidos mas alto aun (ruido = 400 aprox.) .
Se observa en el caso en que el n-discretizacion es menor que el ancho/alto de la imagen, incluso con ruidos bajos la reconstrucción tiene ECM cercano a 300 (Categoría B). Notamos que este error sin duda esta ligado a el método de reducción de la imagen que se usa para comparar el ECM.
Se observa como en este (segundo) caso es mas estable, en el sentido en se necesita mas ruido que antes para empeorar la calidad de la reconstrucción. Esto ultimo se lo atribuimos a que la imagen original es mas grande y por tanto se generan mas rayos.

3.2.1. Conclusiones Experimento 2:

Concluimos a partir de lo observado en este experimento que el método de reconstrucción es muy efectivo para ruidos chicos y continua siendo efectivo para ruidos considerablemente grandes (hasta 200,se mantiene en categoria A). Concluimos adicionalmente, que cuando la reconstrucción es de tamaño menor a la imagen original, para ruidos chico se nota una diferencia sustancial con el caso anterior. Para explicar esto se tendrá que profundizar la investigación de modo que un foco central sea el método de reducción de la imagen original al nuevo tamaño para realizar la comparación de ECM.
Concluimos por ultimo que incrementar el tamaño de la imagen original aumenta la tolerancia a ruido.
Concluimos por tanto que en condiciones normales, si el ruido no es excesivamente alto, el algoritmo podrá reconstruir las imágenes con un grado de calidad satisfactorio.

3.3. Experimento 3

Este experimento busca analizar los tiempos de computo asociados a la ejecución del algoritmo con las diferentes configuraciones de parámetros posibles. Luego de experimentar superficialmente con distintas imágenes y matrices planteamos una hipótesis que buscaremos confirmar o rechazar mediante la experimentación.
Nuestra **hipótesis** :los tiempos de ejecución del algoritmo dependen esencialmente de la cantidad de rayos utilizados y del n-discretizacion (el tamaño de la discretizacion), ya que el n-discretizacion y la cantidad de rayos determinan las dimensiones de la matriz para la cual resolver cuadrados mínimos lineales hipotetisamos que aporta la mayor parte del costo computacional.

Para poner a prueba esta hipótesis graficaremos el tiempo que tarda la resolución por cuadrados mínimos lineales en funcion de n-discretizacion (experimento3.py). Este codigo utiliza para generar rayos el método ”todos”, este es el que genera todos los posibles rayos (para una imagen determinada). Como la cantidad de rayos sera constante los incrementos en tiempo se darán por incrementos en n-discretizacion únicamente (recuerdo que estamos hablando de los tiempos de resolución de cuadrados mínimos lineal, no los tiempos de todo el algoritmo).
Compararemos entonces el tiempo de resolución por CML , que solo depende de n-discret y cantidad de rayos(cantidad de rayos esta fijo en este experimento), con el tiempo total del algoritmo incluyendo los pasos previos (de deconstruccion). Si resultan muy similares los tiempos, se verificara la hipótesis de que los tiempos dependen esencialmente de n-discret (y quedara comprobar en otro experimento que también de cantidad de rayos). Caso contrario, es decir si no son muy similares los tiempos de computo del algoritmo completo con los de CML, no se podrá verificar ni rechazar la hipótesis y se necesitara un análisis mas profundo, segmentando el proceso de deconstruccion.

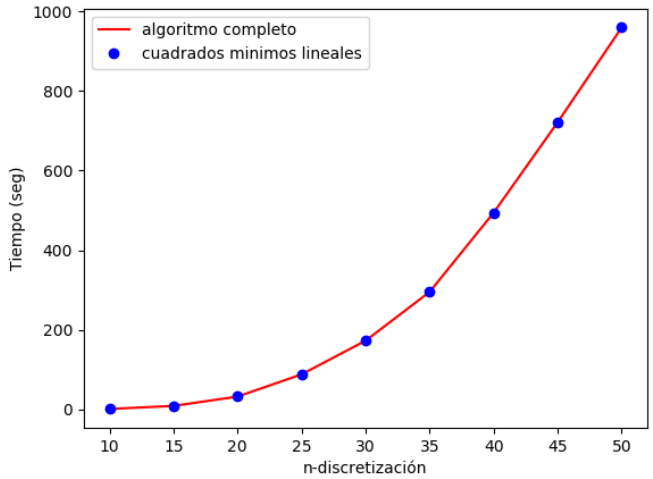


Figura 9: Tiempos en función de n-discret para la imagen ’tomo.csv’ de 100x100

Observaciones:
Se observa como el costo de resolver CML (cuadrados minimos lineales), es escencialmente el costo total del algoritmo.
Se observa el crecimiento agigantado de los tiempos de computo a medida que crece el n-discret y por consiguiente la matriz D (con $(n\text{-}discret)^2$ columnas).

Por lo recién observado, para terminar de comprobar la hipótesis solo faltaría verificar que la cantidad de rayos (independientemente del n-discret) influye significativamente en el tiempo de computo. Para esto iteraremos utilizando el metodo ”rayos random” e iremos variando la cantidad de rayos que se utilizaran, mantendremos fijo el n-discret para que se si se observan cambios significativos, sean por lo observado en el inciso anterior debidos a la resolución de CML y a su vez como el n-discret esta fijo, se le atribuya el cambio significativo a la cantidad de rayos (experimento6.py- parte 1).

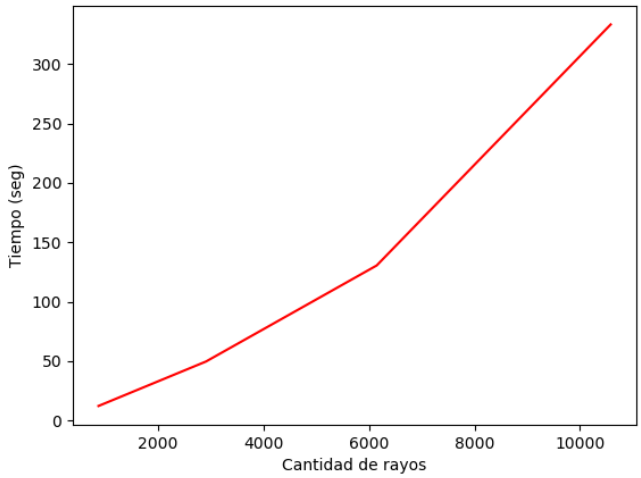


Figura 10: Tiempos en función de cantidad de rayos, imagen 'tomo50x50.csv', n-discret =25

Observaciones:
Se observa en el gráfico como influye de manera significativa la cantidad de rayos.
Se observa que los tiempos son largos incluso para pocos rayos y esto es peor aun cuando se tiene en cuenta que el experimento se realizo con una discretizacion chica(25x25).

3.3.1. conclusiones Experimento 3:

Concluimos por lo observado que se verifica la hipótesis, el tiempo de computo es en su inmensa mayoría costo de la resolución por cuadrados mínimos. es por esto y por lo observado en las figuras 9 y 10 que el esencialmente el costo dependerá de n-discret y m (cantidad de rayos).
concluimos por lo observado y experimentado que, como la cantidad de rayos esta ligada al tamaño de la discretizacion (debe ser mayor para sobredeterminar el sistema) y el tamaño de la discretizacion viene dado por el tamaño mínimo para ver con claridad la tomografía no hay una solución ideal al problema del alto tiempo de computo. Proponemos para futura investigación, investigar y experimentar con nuevos métodos para resolver sistemas de aproximación por cuadrados mínimos y/o experimentar con otros algoritmos para multiplicaciones de matrices.

3.4. Experimento 4

En este experimento se busca comparar diferentes métodos para generar rayos, en particular se busca identificar cual es/son los métodos que minimizan el error cuadrático medio.
Para generar estos rayos se utilizo el método "rayos custom" que utiliza los pares de puntos recibidos por csv para generar rayos y desde python se genero los pares de puntos correspondientes a cada método (cada configuración de rayos).
Se decidió evaluar 3 métodos diferentes para generar rayos:

- **Random**: Desde el borde superior genera $n\text{-}discret^2$ rayos dirigidos al borde inferior, luego desde el borde superior genera $n\text{-}discret^2$ a cualquier lado y por ultimo desde el borde inferior genera otros $n\text{-}discret^2$ a cualquier lado.Siempre generando rayos validos.
- **Catetos opuestos:** Desde el borde superior se generan rayos a todos los elementos del borde inferior. Es posible regular la densidad de puntos generadores de rayos, de modo que solo se lancen rayos de un borde al contrario en un 33 % de los puntos (por ejemplo). En Para este estudio sin embargo se utiliza la máxima cantidad de rayos posibles, es decir que de todos los puntos de borde superior salen rayos a todos los del inferior.

- **todos:** Este método busca generar todos los rayos posibles (validos), es decir de cada borde a todos los otros. En la practica solo se generan la mitad de rayos y luego, cuando se traspone se logran los demás. Hay una cantidad muy pequeña de rayos que si bien son validos no los generáramos por ciertas precondiciones en el algoritmo que usamos para generar rayos, pero es proporcionalmente chica (ejemplo una imagen que debería generar 5000 rayos genera 4876).

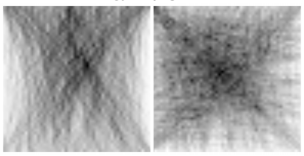
Cada uno de estos métodos sera evaluado en sus 2 versiones, estas son: utilizando y sin utilizar sus rayos traspuestos.
los rayos traspuestos son rayos que agregamos después de generar los rayos desde los pares de puntos.Para generarlos, trasponemos la matriz asociada al rayo (matriz de 1 y 0, 1 por donde pasa el rayo).
Decidimos no utilizar el método de ”rayos canonicos” pues, al ser tan solo $2x$n-discret rayos (si incluimos los traspuestos), no se sobredeterminaria el sistema en la gran mayoría de los casos y por lo tanto, no seria un buen contendiente a ser ”el mejor metodo”.

3.4.1. Experimento 4-A

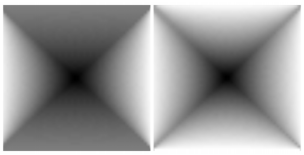
La primer parte del experimento consta de dar una orientación visual a los rayos generados por cada uno de los 3 métodos.
Para estos mostraremos las imágenes que denominamos ”disposiciones de rayos”. Estas imágenes las conseguimos a partir de sumar todos los rayos generados por el método. Notar que se ajusta la imagen de modo que el punto mas oscuro lo pinta de negro y el resto los pinta proporcionalmente.

Disposiciones de rayos (derecha con rayos traspuestos, izquierda sin):

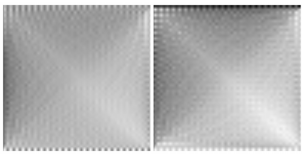
Random



Catetos opuestos



Todos



Observaciones:
Se observa como la disposición de rayos del método random es desorganizada y varia irregularmente. También puede verse del mismo método que a pesar de las irregularidades, parece cubrir de manera relativamente uniforme todos los pixeles, esto se ve mejor particularmente en la versión que incluye los transpuestos,ya que la que no los incluye parece concentrar rayos entre las diagonales (del mismo modo que lo hace el método ”catetos opuestos” aunque este es mas evidente).
Se observa de la disposición de rayos del método catetos opuestos sin traspuestos, como la intensidad se concentra en el centro y entre las diagonales y el eje vertical y queda poco recorrido el resto de la imagen.Se observa también como con los traspuestos se corrige esto pero sigue siendo desproporcionada la intensidad en el centro con respecto al resto de lugares lo que da lugar a creer que esto traerá problemas en la reconstrucción donde todos los pixeles aportan la misma cantidad de información.

Por ultimo se observa en la disposición de rayos del método ”todos”, como estos rayos están uniformemente distribuidos.Esto daría lugar a creer que la reconstrucción sera buena.

3.4.2. Experimento 4-B

La segunda parte del experimento es comparar los errores cuadráticos medios (ECM) de los diferentes métodos, para esto con una imagen y n-discretizacion fijas se evaluara el desempeño de cada uno de los métodos. Adiccionalmente se mostraran la cantidad de rayos utilizados en cada método, de modo que uno pueda elegir a conveniencia, intercambiando calidad por tiempo (utilizando los resultados de tiempo por rayo del experimento 3).

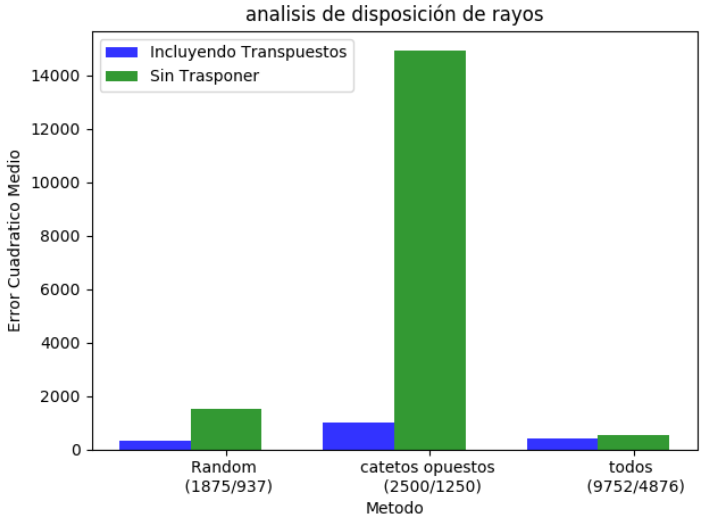


Figura 11: imagen usada: ’tomo50x50.csv; de 50x50; n-discret = 25; ruido = 150

Observaciones:
El método ”Random” incluyendo traSpuestos parece tener un ECM muy similar al método ”Todos” (con traspuestos) pero tiene tan solo un 20 % de la cantidad de rayos.
el método ”Todos” no mejora sustancialmente al trasponer a pesar de estar agregando una cantidad sustancial de rayos. Esto y la cercanía de este método con el método ”random” da lugar a interpretar que, agregar rayos solo sirve para mejorar el ECM hasta cierto punto, Hay un limite en el que mas rayos, no se traduce a mayor calidad. Proponemos ejecutar de nuevo el algoritmo aumentando el ruido para darle mas lugar de movimiento al ECM y experimentar la relación ECM-cantidad de rayos.
Observamos por ultimo el fracaso del método catetos opuestos, incluyendo traspuestos(muy inferior a random pero aun así usa mas rayos).El método catetos opuestos sin traspuestos fracaso aun mas.

Por lo pedido en esta ultima observación volveremos a ejecutar el experimento pero con un ruido mas alto para darle mas lugar de movimiento al ECM:

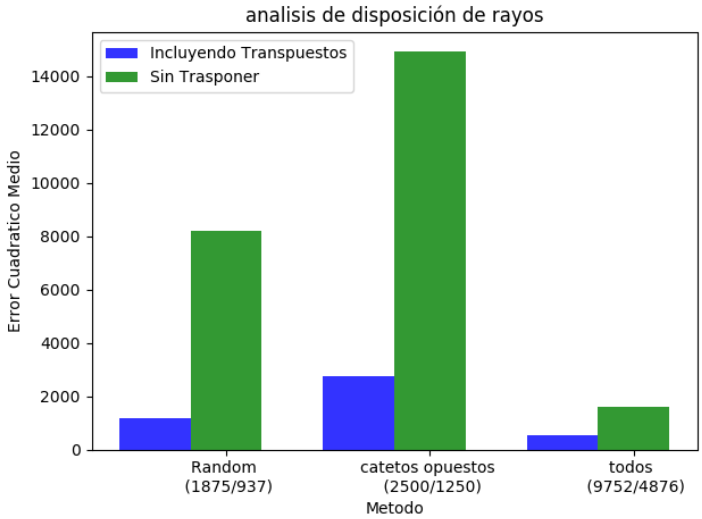


Figura 12: imagen usada: 'tomo50x50.csv; de 50x50; n-discret = 25; ruido = 1000

Observaciones:
Se observa en esta nueva instancia del mismo experimento pero con ruido = 1000, como se cumple la predicción. Cuando el ECM no esta en el limite, entonces puede mejorar usando una mejor disposición y/o mayor cantidad de rayos.En cambio si sucede como sucedía en el caso anterior, con el ruido = 150 el ECM estaba muy cerca de su lugar optimo(la mejor reconstrucción con cualquier método posible) y por lo tanto cualquier disposición de rayos lo suficientemente buena dará resultados parecidos (resultados cercanos al optimo). Si se lo aleja, como se hizo en este caso aumentando el ruido,se aprecia como los mejores métodos de reconstrucción (como el método "Todos") le sacan ventaja a los métodos inferiores como el método "Random".
Por otro lado, si bien le gana el algoritmo "todos" no hay que dejar de tener en cuenta que el método "Random" (incluyendo transpuestos) esta relativamente cerca, es competente y utiliza tan solo un 20 % de los rayos lo que puede ser muy útil en materia de tiempos de computo y también en cantidad de rayos emitido al tejido humano (que podría ser dañino en grandes cantidades).

3.4.3. Conclusiones Experimento 4:

Concluimos a partir de lo observado que, el método "Todos" es el método que mejor reconstruye, aunque para ruidos bajos, el "Random" reconstruye igual de bien con tan solo un 20 % de la cantidad de rayos, lo que dependiendo de si se tienen limitaciones temporales, puede ser una buena idea. De establecerse alguna métrica del daño a la salud por recibir rayos, podrían realizarse experimentación para evaluar cual de los 2 métodos tiene un mayor costo-beneficio.
Concluimos que para explorar otros posibles métodos, es una buena euristica observar que la disposición de rayos este distribuida de modo uniforme por todos los pixeles.

3.5. Experimento 5

En Este experimento nos proponemos estudiar la influencia de la cantidad de rayos en la calidad de la reconstrucción.
Para evaluar esto iteraremos ejecutando el algoritmo variando la cantidad de rayos lanzados (con el método "Random") y evaluaremos en cada iteracion el ECM.(experimento6.py–parte 2)

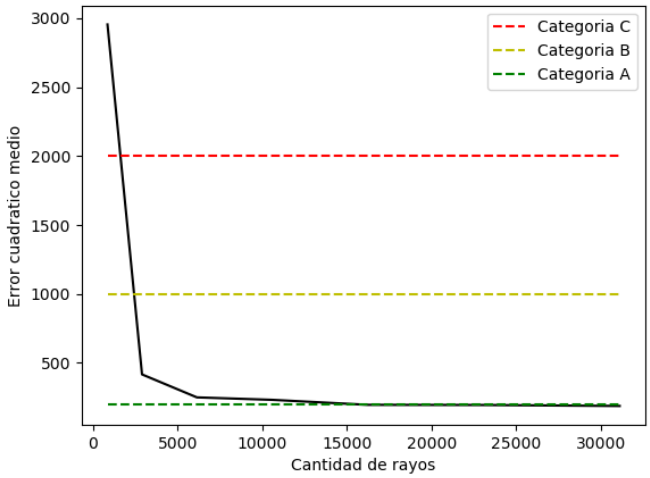


Figura 13: ECM en función de cantidad de rayos, imagen:'tomo50x50.csv', ruido = 150, n = 25

Observaciones: Se Observa del gráfico como la cantidad de rayos es muy influyente en un principio, cuando hay pocos, pero llega un punto a partir del cual, agregar mas rayos no mejora considerablemente la calidad.
Debe recordarse del experimento 3 que la cantidad de rayos es muy influyente en el tiempo de procesamiento.
Mas importante aun los rayos en grandes cantidades pueden ser dañinos para la salud, por lo que es muy importante buscar el equilibrio justo.

3.5.1. Conclusiones experimento 5:

A partir de los observado re-confirmamos lo concluido en el experimento 4, agregar rayos ayuda a mejorar el ECM siempre y cuando, el ECM este lejos de su optimo (lejos de la mejor reconstrucción posible) una vez cerca, no mejorara mucho.Proponemos para encontrar este punto optimo un algoritmo similar al usado en algunas implementaciones del método de la potencia para terminarlo. Fijar (en función de las restricciones de tiempo que se tengan) un epsilon de modo que cuando el ECM difiere en menos de epsilon se dejen de agregar rayos.
Por ultimo, igual que en en la conclusión del 4, seria necesario alguna métrica si se quiere evaluar el costo del daño producido en el cuerpo por rayo.

3.6. Experimento 6

En este experimento nos proponemos comparar los tiempos de ejecución del algoritmo utilizando la implementacion de matriz original ($vector < vector < double >>$), con una implementacion para matrices ralas (la que utilizamos en el TP1).
La intención es ver si puede sacarse provecho de la gran cantidad de 0 que pueden aparecer en una imagen tomografica y de este modo ahorrar tiempo de computo utilizando una implementacion de matriz de este tipo.
Graficamos los tiempos de ejecucion del algoritmo en ambos casos variando el n-discretizacion y utilizando el método random+transpuestos para generar los rayos en cada instancia.(experimento7.py)

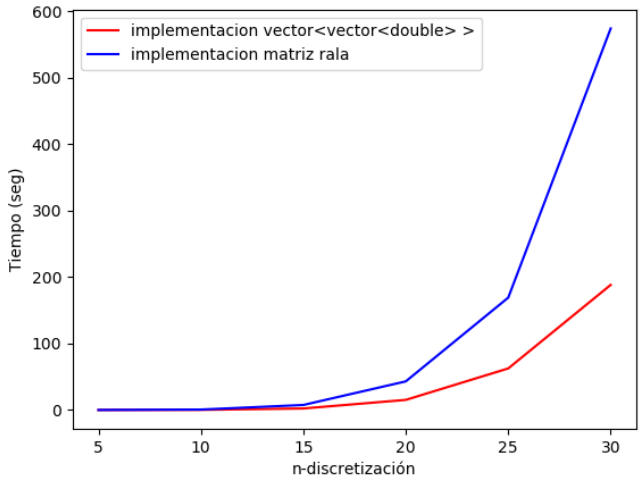


Figura 14: Tiempos de ejecución de ambas implementaciones con la imagen 'tomo.csv' de 100x100

Observaciones: Se observa del gráfico como rápidamente, la implementacion original le gana a la que utiliza una estructura de datos para matriz rala.

3.6.1. Conclusiones experimento 6:

Se concluye a partir de lo observado que es recomendable utilizar la implementacion original por sobre la que implementa una estructura de datos especifica para matrices ralas.

3.7. Conclusiones Generales

Por lo evaluado en los distintos experimentos concluimos que: El algoritmo reconstruye bien para ruidos que no sean excesivamente altos.
El algoritmo tiene problemas en cuanto al tiempo de computo, procesar imágenes chicas le toma mucho tiempo. Observamos que el cuello de botella esta en la etapa de resolver el sistema sobredeterminado de cuadrados mínimos, mediante la ecuaciones normales y luego aplicando Gauss. Este proceso (en particular la multiplicación de matrices) tarda mucho. Concluimos que como no podemos reducir el tamaño de la discretizacion (el doctor necesita ver la tomografía en cierto tamaño) la única alternativa para reducir el tamaño de la matriz es reduciendo la cantidad de rayos, para lo que propusimos utilizar el método random que genera resultados aceptables o muy buenos (si el ruido es poco) pero manteniendo acotada la cantidad de rayos (en comparacion con el otro método efectivo).
Sin embargo incluso con el método 'Random' el tiempo de ejecución excede lo deseado, por esto propusimos a futura investigación, investigar algoritmos de multiplicación de matrices y/o algoritmos alternativos para resolver sistemas de cuadrados mínimos.
Propusimos en el experimento 5 un métodos para elegir la cantidad de rayos a generar.
Concluimos que la matriz implementada sobre vector de vectores es mas eficiente que la implementacion pensada para matrices ralas.

4. Apéndice