DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

23 de julio de 2018 — Métodos numéricos

Integrante	LU	Correo electrónico
Luis Arroyo	913/13	luis.arroyo.90@gmail.com
Carlos Humpiri	942/14	jhumpiri1599@gmail.com
Matías Millassón	131/13	matiasmillasson@gmail.com
Eric Peker	548/13	epeker.135@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

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

1.1. Motivación del problema

Lo que nos motiva a realizar este experimento es el de encontrar una forma de observar que es lo que hay en medio de un cuerpo, y como no podemos cortarlo para analizarlo, lo tenemos que hacer por medio de un método alternativo. Por lo cual vamos a usar el método que es usado en los tomógrafos que traza distintos rayos a través de un cuerpo blando el cual calcula una cierta información y en base a generar muchos rayos de dentro de una misma sección de cuerpo, poder reconstruir la imagen. Pero como no tenemos la posibilidad de acceder a un tomógrafo real, lo que vamos a hacer es simularlo tomando los rayos de una imagen y reconstruirla en función de los rayos creados.

1.2. Objetivos

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

1.3. Modelado

El análisis tomográfico de una sección (que suponemos bidimensional y cuadrada) de un cuerpo consistente en emitir señales de rayos X que lo atraviesan en diferentes direcciones, midiendo el tiempo que tarda cada una en atravesarlo. Por ejemplo, la **figura 1** muestra una posible distribución de estas señales.

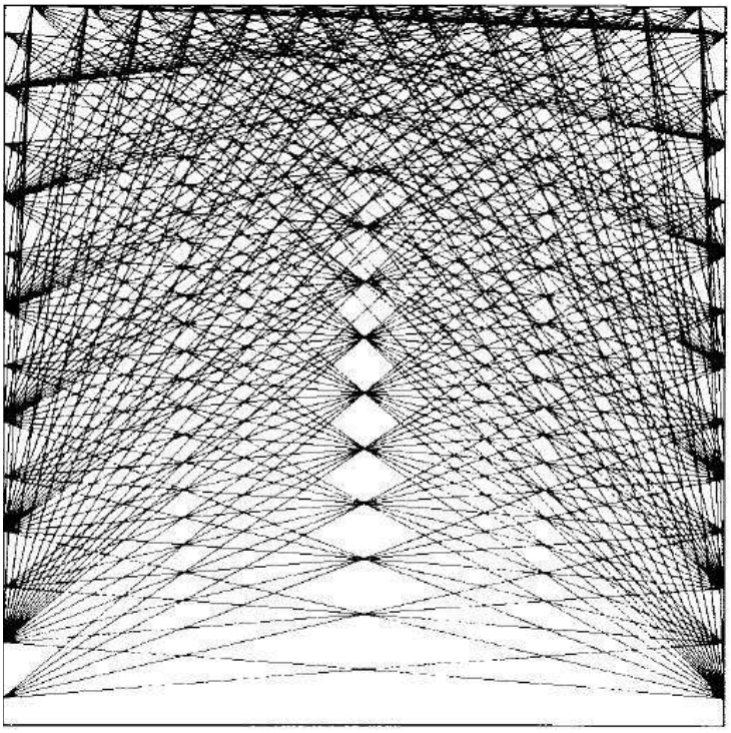


Figura 1: Ejemplo de configuración de señales

Suponemos que el cuerpo se discretiza en nxn celdas cuadradas, de modo tal que en cada celda las señales de rayos X tienen velocidad constante. Si d_{ij}^k es la distancia que recorre el k-ésimo rayo

X 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 de rayo X en esa celda, entonces el tiempo de recorrido de la señal completa es de:

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

Como resultado del análisis tomográfico, se tienen mediciones del tiempo que tardo cada señal en recorrer el cuerpo. Si se emitieron m señales, entonces el resultado es un vector $t \in \mathbb{R}^{\mathrm{m}}$, tal que t_k indica el tiempo de recorrida de la k-ésima señal. Sea $D \in \mathbb{R}^{\mathrm{mxn}^2}$ 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} se pueden reconstruir resolviendo el sistema de ecuaciones $Ds = t$. El vector solución $s \in \mathbb{R}^{\mathrm{n}^2}$ contiene los de las velocidades originales en cada celda.

1.4. Resolución de Cuadrados mínimos

Sea $D \in \mathbb{R}^{\mathrm{mxn}^2}$, donde el rango$(D) = r$ y $t \in \mathbb{R}^{\mathrm{m}}$, quiero resolver la siguiente sistema de ecuaciones lineales dada por: $Dv = t$, donde $v \in \mathbb{R}^{\mathrm{n}^2}$ es la solución del sistema, que es la imagen que vamos a querer reconstruir.

Lo que debemos considerar es la presencia de errores de medición en el vector t de tiempos , por lo tanto el sistema estará en general sobredeterminando 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 una solución aproximada al sistema original.

El sistema de ecuaciones lineales $Dv = t$, si la resolvemos por cuadrados mínimos lineales siempre tiene solución, ya sea única o teniendo infinitas.
Para la resolución del sistema vamos a utilizar las ecuaciones normales que son: $D^t D v^* = D^t t$, donde v^* es la solución que resuelve de forma aproximada el sistema original.

Sea la factorización SVD de la matriz D, $D = U\Sigma V^t$, donde U, V son matrices ortogonales y Σ una matriz diagonal, vamos a reemplazar a la matriz D por su factorización SVD en las ecuaciones normales e ir despejando una forma para poder calcular el vector v^*.

$$D^t D v^* = D^t t \tag{2}$$

Aplicando la factorización SVD tenemos:

$$V\Sigma^t U^t U \Sigma V^t v^* = V\Sigma^t U^t t \tag{3}$$

Como U es ortogonal tenemos que $U^t U = I$, entonces nos queda:

$$V\Sigma^t \Sigma V^t v^* = V\Sigma^t U^t t \tag{4}$$

Ahora si multiplico a izquierda por V^t en ambos lados vamos a tener que $V^t V = I$ por lo tanto nos va quedar:

$$\Sigma^t \Sigma V^t v^* = \Sigma^t U^t t \tag{5}$$

Veamos un poco como el la matriz $\Sigma^t \Sigma \in \mathbb{R}^{\mathrm{n}^2 \mathrm{xn}^2}$ y ver como podemos usarla:

$$\Sigma^t\Sigma = \begin{pmatrix} {\sigma_1}^2 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & {\sigma_2}^2 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & {\sigma_r}^2 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}$$

entonces podríamos definir como una pseudoinversa a la siguiente matriz $(\Sigma^t\Sigma)^{-1}$ $\mathbb{R}^{n^2 x n^2}$ de esta forma:

$$(\Sigma^t\Sigma)^{-1} = \begin{pmatrix} \frac{1}{{\sigma_1}^2} & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{{\sigma_2}^2} & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & \frac{1}{{\sigma_r}^2} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}$$

Usando esta pseudoinversa en la ecuación (5), podemos nos queda:

$$V^t v^* = (\Sigma^t\Sigma)^{-1}\Sigma^t U^t t \tag{6}$$

y multiplicando por la matriz V en ambos tenemos:

$$v^* = V(\Sigma^t\Sigma)^{-1}\Sigma^t U^t t \tag{7}$$

La formula (7) refleja como hallar la solución por cuadrados mínimos en forma matricial usando la descomposición SVD, pero también la podemos reescribir usando las columnas de la matriz V, las filas de la matriz U y los valores singulares. Para esto vamos a llamar a v_i el i-ésima columna de la matriz V, u_i^t la i-ésima fila de la matriz U^t, σ_i el i-esimo valor singular de la matriz Σ y además sabíamos que rango$(D) = r$, reescribiremos la formula (7) en forma vectorial:

$$v^* = \sum_{i=1}^{r} \frac{u_i^t t}{\sigma_i} v_i \tag{8}$$

De esta manera la formula (8) es la solución que vamos a implementar para poder resolver el sistema $Dv = t$ aplicando el método de cuadrados mínimos lineales usando la descomposición SVD.
Cabe aclarar que si resolvemos el problema por medio de la formula (8), siempre tenemos una única solución. Lo único que si vamos a tener que buscar el ϵ que nosotros interpretaremos como cero a la hora de implementar, para que nos sea mas conveniente a la hora de querer conseguir el rango de la matriz.

2. Desarrollo

2.1. Generación de rayos

La motivación detrás de la generación de rayos en este trabajo práctico es encontrar un patrón que cubra la mayor parte de la imagen. Para nosotros, un rayo está representado como un **punto de origen**, es decir, una posición de la matriz de píxeles que forma nuestra imagen y un valor decimal m que expresa la **pendiente** de la recta. Esta pendiente se obtiene tomando algún punto destino y realizando el cálculo:

$$m = \frac{destino.y - origen.y}{destino.x - origen.x} \tag{9}$$

El tiempo recorrido de la recta completa por sobre la imagen se calcula tomando la velocidad de la celda origen y luego sumándole m al valor de la columna para acceder al próximo punto y así obtener las diferentes velocidades. Esto se repite hasta que los índices se salgan por fuera del límite del borde de la imagen.

En cuanto al patrón, tomamos como origen a los puntos de la primer columna de la imagen. Para variar el espacio entre las rectas, decidimos especificar la distancia entre los puntos que se van a utilizar como orígenes, como puede verse en la figura 2. Por ejemplo, si especificamos un valor de *distancia entre puntos origen* de k, se tomarán como puntos origen los puntos $(1+i*k, 1)$, mientras el valor de $1 + i * k$ sea menor a la cantidad de filas de la matriz.

Por cada punto origen se toman varios puntos destino para calcular las rectas que pasan por los dos. En un principio, el patrón que se decidió tomar es que por cada origen se tome una recta que pase por este y por cada uno de los puntos de los bordes opuestos de la imagen, es decir, todos los puntos de la primera y última fila y los puntos de la última columna. Dado que generar rectas sobre todos estos puntos para realizar el cálculo de cuadrados mínimos puede llegar a ser muy intensivo, también se decidió elegir los puntos destino con una distancia fija que denominamos *densidad*. Por ejemplo, si tomamos una densidad de dos, se tomaran rectas que pasen por los puntos destino $(1, 1), (1, 3), (1, 5)...(1, 1 + 2 * k)$ para la primera fila y así también para los otros bordes.

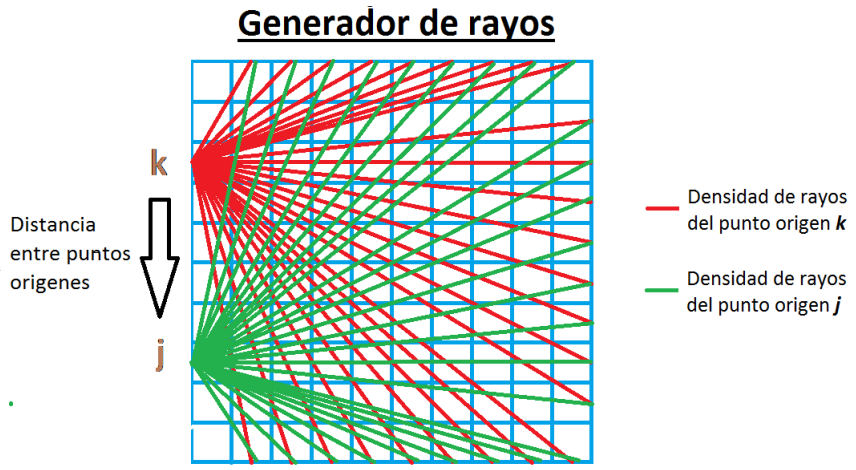


Figura 2: Generados de rayos

El próximo paso fue intentar obtener rectas que tuvieran su origen en puntos diferentes a los de la primer columna. Originalmente se decidió que se tomarían bases de puntos de origen equidis-

tantes entre si, es decir, que se elegiría una distancia fija d entre las bases y a partir de los puntos de cada una de ellas se tirarían rectas hacia los bordes opuestos. Esto quiere decir que nuestros puntos origen tendrían la forma $(1+i*k, 1+d)$, siendo k la distancia entre los puntos de la misma base y d la distancia entre cada columna de la cual se vaya tomar, es decir, la distancia entre bases.

Sin embargo, a la hora de realizar los experimentos e intentar recrear una imagen a partir de los rayos nos dimos cuenta de que esta forma de paneo sobre la imagen no resultaba satisfactoria: el hecho de que las rectas siempre fueran en la misma dirección (hacia .ªdelante", si interpretamos el crecimiento de los índices de las filas de izquierda a derecha, como en la figura 2) hacía que hubiera una gran densidad de rectas que impactaran sobre el lado derecho de la imagen, generando un desbalance entre la definición que se obtenía a la hora de la recreación.

Debido a este dato obtenido de forma empírica, se decidió variar la forma en que distanciamos las bases de puntos origen. Ya que se observó un desbalance en el lado derecho, compensamos creando más rectas desde el lado izquierdo distanciando a las bases de origen de forma cuadrática (como podemos ver en la figura 3). Esto quiere decir que si nuestra primera base es la columna 1, nuestra segunda será la 4, la tercera la 9 y así sucesivamente. Dado que la función cuadrática crece más rápidamente que la lineal, esto nos asegura que habrña más bases de puntos origen del lado izquierdo que el derecho a medida que el tamaño de la imagen se vuelva cada vez más grande.

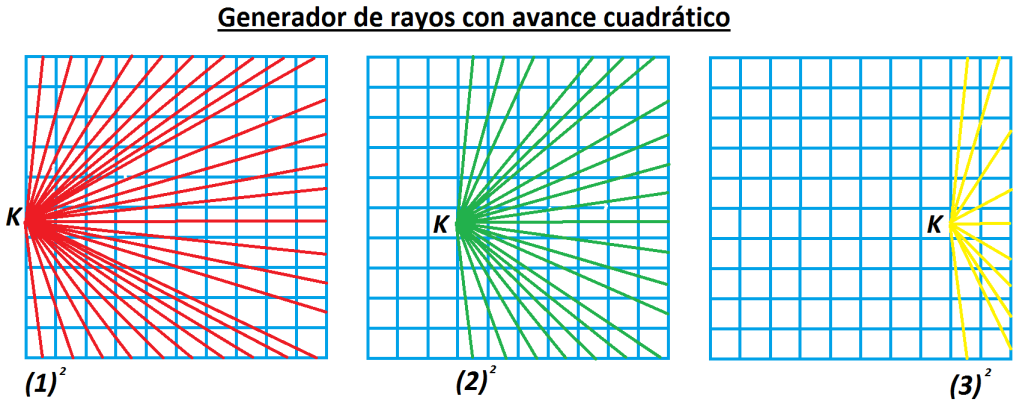


Figura 3: Generador de rayos con avance cuadrático

2.2. Armado de la matriz D y vector t

Ya definido como generamos nuestro rayos, lo que haremos es aplicar estos rayos a la imagen tomográfica, para poder crear la matriz D y el vector t.

Para simplificar la idea de como vamos a ir creando la matriz D y vector t, vamos a tirar un rayo sobre la diagonal principal de la imagen tomográfica, entonces t_i (es el tiempo i-ésimo que tarda en cruzar este rayo) va ser la suma de los píxeles (intensidades) de la diagonal principal de la imagen como se muestra en la Formula (9). Este t_i va a ir en la fila i-ésima del vector t.

$$t_i = \sum_{i=1}^{n} (imagen)_{ii} \tag{10}$$

Una vez que trazamos este rayo sobre la diagonal principal de la imagen, vamos a crear una matriz A con las mismas dimensiones que la imagen (cabe aclarar que la imagen es cuadrada y bidimensional como se menciono en el modelado). Sabemos que rayo a tocado los píxeles de la

diagonal, por lo tanto lo podemos expresar los píxeles tocados como $(imagen)_{ii}$ donde $i = 1...n$, entonces la matriz A se va llenar en este caso como se muestra en la Formula (10):

$$(A)_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Cabe destacar que la Formula (10) es para el rayo que cruzo la diagonal principal de la imagen tomográfica. Una ves que tenemos la matriz $A \in \mathbb{R}^{nxn}$ ya definida con sus respectivos valores, vamos a transformalo en un vector de $\mathbb{R}^{1xn^2}$, para poder colocarlo en la fila i-ésima de la matriz $D \in \mathbb{R}^{mxn^2}$. La **figura 4** se muestra como vamos armando las matriz D y el vector t aplicando rayos de distintos ángulos sobre la imagen tomográfica:

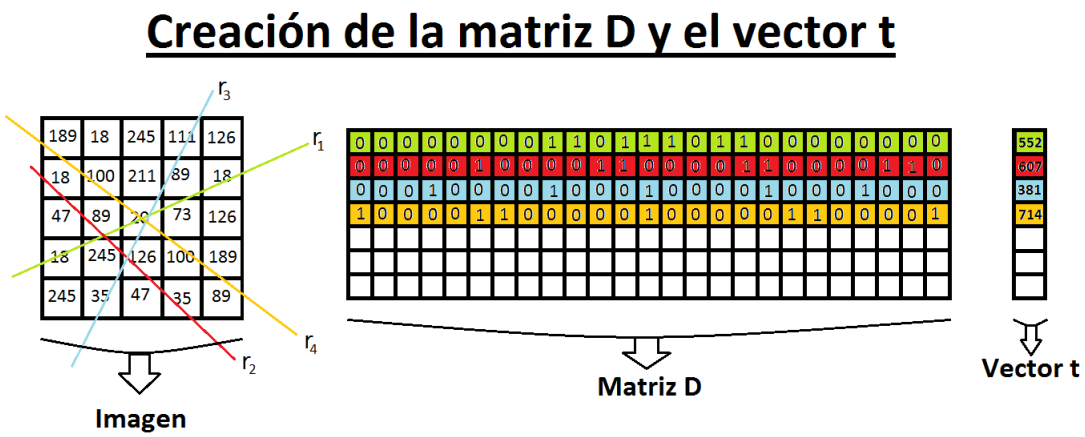


Figura 4: Creación de la matriz D y el vector t por medio de los rayos

Una vez terminado el armado de la matriz D y el vector t vamos a pasar a explicar la idea de como discretizar la matriz D.

2.3. Discretización

Una vez que tenemos la matriz D de distancia, una manera que podemos discretizar la matriz D es aplicando el método de los 4 vecinos mas cercano como se ilustra en la **Figura 5**:

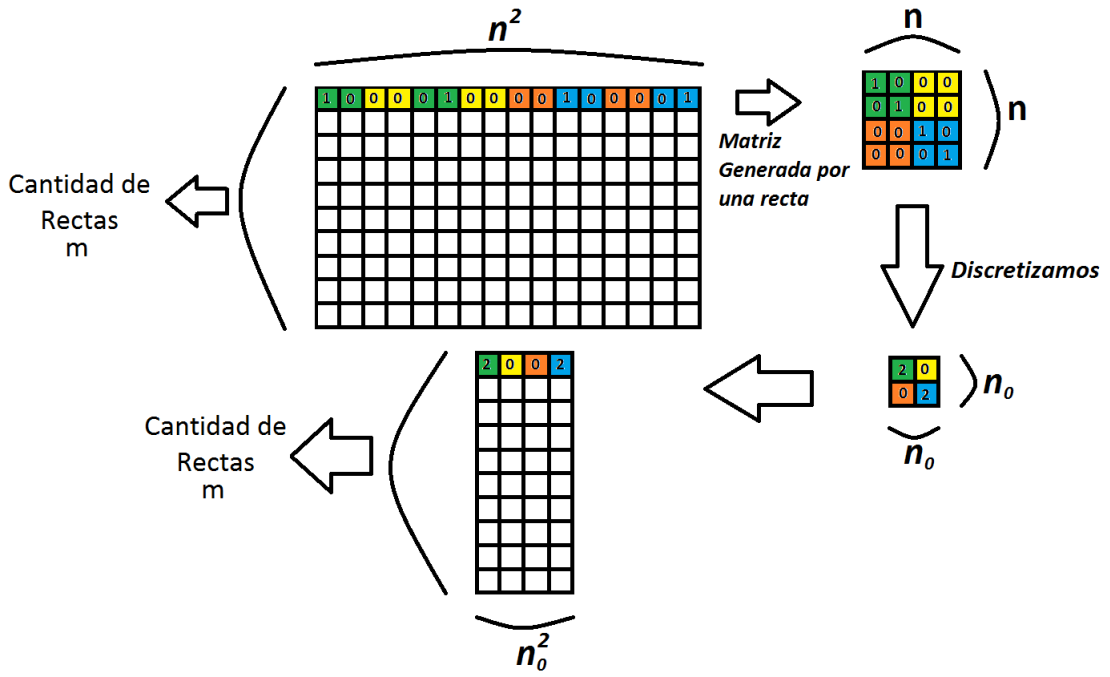


Figura 5: Discretización de la matriz D

Esta manera de discretizar lo que tiene de bueno, es que una vez aplicado una cantidad de rayos a la imagen original y obtuvimos la matriz D y el vector t, podemos aprovechar al máximo la información obtenida (que estará guardada en la matriz D), sin tener que aplicar mas rayos sobre la imagen.

2.4. Manejo de imágenes y Normalización

2.4.1. Manejo de imágenes

El formato de imagen con el cual trabajamos es PGM de 8 bits, osea que la imagen de 100x100 de formato PNG provisto por la cátedra (**Figura 6**), lo que hicimos fue convertirla a PGM para poder testear nuestro código implementado en $C++$.

Figura 6: Imagen provista por la catedra

A la hora de querer experimentar con una imagen, para poder hacer un análisis cualitativo y cuantitativo, se toma una imagen en formato PGM de 8 bits. Después para poder reconstruir la imagen resultante se opto por el formato PGM de 8 bits.

Otra forma que trabajamos fue usar los $CSVs$ de las imágenes en formato $DICOM$, la forma de como convertimos las imágenes $DICOM$ a formato CSV, fue usando los archivos de python

provistos por la cátedra.

En cada experimento se va a especificar que manejo de imagen se uso en ese respectivo caso.

2.4.2. Normalización de las imágenes

Una ves que obtuvimos $v^* \in \mathbb{R}^{n^2}$ que es la solución del sistema $Dv = t$ aplicando el método de cuadrados mínimos lineales, vamos a usar v^* para reconstruir la imagen resultante. Para hacer esta reconstrucción aplicamos dos métodos diferentes de normalización de imagen. Estas dos normalización se detallan a continuación:

a) **Normalización 1:** Todo numero negativo dentro del vector v^* lo vamos a reemplazar por el numero 0, si hay números mayores a 255 lo vamos a reemplazar directamente por 255 y todos los demás números que estén entre 0 y 255 se les hará un redondeo del numero como por ejemplo: 64.3698 = 64 o 64.7536 = 65.

b) **Normalización 2:** Para este normalización lo que vamos hacer es buscar el máximo y mínimo elemento del vector v^* para poder aplicar la siguiente formula: sea v_{max} el maximo valor y v_{min} el mínimo valor de v^* entonces:

$$(\frac{(v^*)_i - v_{min}}{v_{max} - v_{min}}) * 255 \tag{12}$$

donde $(v^*)_i$, donde $i = 1...n^2$.

Lo que esta haciendo la formula (9), es no despreciar las intensidades de cada componente del vector v^*, si no que expresar mejor los datos en imágenes de 8 bits.

A la hora de reconstruir la imagen, la Normalización 2 fue mejor que la Normalización 1, por lo tanto en toda la experimentación cuando queramos reconstruir una imagen siempre usaremos la Normalización 2.

2.5. Métricas

Para medir el error de la imagen resultante utilizaremos el error cuadrático medio, definido como:

$$ECM(v, v^*) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(v_{ij} - v^*_{ij})^2}{n^2} \tag{13}$$

Donde v_{ij} es la velocidad de la celda ij y v^*_{ij} la velocidad en la misma celda del cuerpo a reconstruir.

2.6. Formato de entrada y salida

El formato de entrada esperado es el siguiente:

1. un natural que indique la cantidad de reducciones
2. un natural que indique la cantidad de discretizaciones
3. un string que represente la ruta al archivo con la imagen PGM. Puede ser relativo o absoluto
4. un natural que indique la densidad de la generación de rectas
5. un natural que indique el avance para la generación de rectas

Todos los parámetros se reciben por la stdin en ese orden.
Ejemplo de ejecución:

```
echo "2 1 /home/rfavaloro/radiografiacerebral.pgm 1 1" | ./tp3
```

Con respecto al formato de salida el programa va a colocar la imagen reconstruida en el mismo directorio donde se encontraba la imagen PGM.

2.7. Resolución del problema

2.7.1. Funciones implementadas

2.7.1.1 Proceso Tomográfico

Esta función ejecuta el algoritmo de reconstrucción de imágenes tomográficas. El mismo calcula vector velocidades v por el método de cuadrados mínimos. Aplica el trazado de rayos a la imagen tipográfica, para así poder obtener la matriz D y el vector t, así obtener la imagen a reconstruir. También da opción por si queremos reducir la imagen tomográfica a dimensiones mas pequeñas para que la ejecución a la hora de tiempos sea rápido. Otra opción que tiene es poder discretizar la matriz D la veces que uno necesite. Se detalla el pseudocódigo en **Algorithm 1**:

Algorithm 1: proceso_tomográfico(Matriz original, int densidad, int avance, int cantidad_discretizaciones)→ vector velocidad

```
   /* Creación de las rectas */
1  conjuntoDeRayos ← dame_rectas(densidad, avance, original.alto, original.ancho)
   /* Creación de la matriz D y el vector t */
2  Matriz D, vector t ← aplicar_rectas(original, conjuntoDeRayos)
   /* Discretización */
3  for ( j ∈ [1...cantidad_discretizaciones] ) {
4  |  D ← discretizacion(D)
5  |  original ← reducir_tamanio_a_la_mitad(original)
6  }
   /* Resuelvo por cuadrados mínimos */
7  vector velocidad ← cuadrados_minimos(D,t)
   Data: Calcula el vector de velocidades
```

2.7.1.2 Cuadrados Mínimos

Este algoritmo ejecuta el método de cuadrados mínimos lineales, para el sistema $Dv = t$. Para esto utiliza las ecuaciones normales, por medio de la factorización SVD de la matriz de distancias D. Se detalla el pseudocódigo en **Algorithm 2**:

Algorithm 2: cuadrados_minimos(Matriz D, vector t)→ vector w

```
1  Matriz A ← D^t * D
2  Matriz V, Matriz C ← generacion_U_D(A,Columnas(A))
3  Matriz S ← matriz_nula(Filas(A),Columnas(A))
4  (S)_ii ← sqrt((C_ii))
5  Matriz U ← conversion_U_V(V,S)
6  for ( i ∈ [0..rango(U^t)] ) {
7  |  w ← ((fil(U)^t)_i * t / (S)_ii) col(v)_i
8  }
   Data: Resuelve Dw = t, donde w es el vector velocidad
```

2.7.1.3 Creación del generador de rectas

Este algoritmo va generar todas rectas que utilizaremos al momento de querer aplicarlo a la imagen tomografica. Para esto creamos opciones como la densidad y la distancia entre puntos (Ésto se explico en la parte de desarrollo) para poder así utilizarlo .Se detalla el pseudocódigo en **Algorithm 3**:

Algorithm 3: dame_rectas_sobre_base_cuadratica(int densidad, int avance, int alto, int ancho)→ conjunto conjRayos

```
1 vector< Punto > Puntos
2 vector< Recta > Rectas
3 for ( j ∈ [0..alto]; j = (j + 1) ∗ (j + 1 ) {
4 |   for ( i ∈ [0..ancho]; i = i + avance ) {
5 |   |   Punto P ← (i,j)
6 |   |   Puntos ← P
7 |   }
8 }
9 conjRayos ← dame_rectas(Puntos, densidad, ancho, alto)
```

Data: Arma el conjunto de rayos a utilizar en función de la densidad, el avance y el tamaño de la imagen

El siguiente algoritmo 4 muestra como se generan las rectas desde todos los puntos generados en el algoritmo 3. Notar que para representar una recta usamos dos puntos. Se itera por todos los puntos y se calculan los puntos por los cuales tiene que pasar cada recta.

Algorithm 4: dame_rectas(conjunto puntos,int densidad, int alto, int ancho)→ conjunto conjRayos

```
1  for ( k ∈ [0..puntos.size] ) {
2  |   for ( i ∈ [(k.first + 1)..alto − 2]; i = i + densidad ) {
3  |   |   Punto nuevo_izq ← (0, i)
4  |   |   Punto nuevo_der ← (ancho − 1, i)
5  |   |   if k.first ≠ 0 then
6  |   |   |   conjRayos.agregar(Recta(nuevo_izq,k))
7  |   |   end
8  |   |   if k.first ≠ ancho − 1 then
9  |   |   |   conjRayos.agregar(Recta(nuevo_der,k))
10 |   |   end
11 |   }
12 |   for ( i ∈ [ancho − 1..,0]; j = j − densidad ) {
13 |   |   if k.first ≠ i then
14 |   |   |   conjRayos.agregar((i,alto-1),k)
15 |   |   end
16 |   }
17 }
```

Data: Arma el conjunto de rayos a utilizar en función de un conjunto de puntos, la densidad, ancho y alto de la imagen

2.7.1.4 Aplicación de las rectas a la imagen tomográfica

Esta función ejecuta el algoritmo que se encargara de aplicar todas las rectas creadas anteriormente sobre la imagen tomográfica, para así poder obtener la matriz de distancias D y el vector de tiempos t. Se detalla el pseudocódigo en **Algorithm 5**:

Algorithm 5: aplicar_rectas(Matriz original, conjunto rectas)→ Matriz D, vector t

```
1  for ( k ∈ [0..rectas.size]] ) {
2  |  tiempo ← 0
3  |  Punto origen ← rectas[k].second
4  |  i ← origen.first
5  |  j ← origen.second
6  |  while (i < Filas(imagen) && 0 ≤ i && 0 ≤ j && j < Columnas(imagen)) do
7  |  |  i_destino ← k
8  |  |  j_destino ← (Columnas(imagen)*i) + j
9  |  |  tiempo ← tiempo + imagen[i][j]
10 |  |  (D)_ij ← 1
11 |  |  i,j ← dame_proximo_punto(rectas[k])
12 |  end
13 |  t[i] ← tiempo
14 }
```

Data: Aplica los rayos a la imagen y devuelve la matriz D y el vector t

2.7.1.5 Reducción de la imagen

Esta función ejecuta el algoritmo con el cual vamos a poder reducir las dimensiones de la imagen por la mitad (que en nuestro caso seria la imagen tomografica). Se detalla el pseudocódigo en **Algorithm 6**:

Algorithm 6: reducir_tamanio_a_la_mitad(Matriz original)→ Matriz res

```
1  for ( i ∈ [0..Filas(original)]; i+ = 2 ) {
2  |  for ( j ∈ [0..Columnas(original)], j+ = 2 ) {
3  |  |  res_{i/2 j/2} ← original_ij + original_(i+1)j + original_i(j+1) + original_(i+1)(j+1)
4  |  }
5  }
```

Data: Reduce el tamaño de la matriz a la mitad

2.7.1.6 Discretización de la matriz D

Este algoritmo ejecuta la discretización de la matriz D explicada anteriormente en la parte de desarrollo. Se detalla el pseudocódigo en **Algorithm 7**:

Algorithm 7: discretizacion(Matriz D)→ Matriz res

```
   /* D ∈ IR^{mxn^2} */
1  n ← √(columnas(D))
   /* A ∈ IR^{mx n^2/2} */
2  matrix A
3  for ( i ∈ [1...Filas(D)] ) {
4  |  for ( j ∈ [1...Columnas(D)] ) {
5  |  |  A[i][j/2] ← (D[i][j] + D[i][j+1])
6  |  }
7  }
   /* B ∈ IR^{mx n^2/4} */
8  matrix B
9  for ( i ∈ [1...Filas(A)] ) {
10 |  for ( j ∈ [1...n/2] ) {
11 |  |  for ( t ∈ [1...n/2] ) {
12 |  |  |  B[i][(n/2)*j +k] = A[i][(n*j) + k + (n/2)]
13 |  |  }
14 |  }
15 }
16 res ← B
   Data: Discretiza la matriz D
```

2.7.1.7 Funciones para calcular la matrices V y D

Aquí vamos a mostrar como implementamos las funciones necesarias para calcular las matrices V y D donde D es la matriz diagonal que tiene los autovalores de y V es la matriz donde están los autovectores asociados.

Para calcular el autovalor dominante programamos el método de la potencia. Este método consiste en multiplicar la matriz pasada por parámetro por el vector pasado como parámetro y luego verificar que se cumpla la definición de autovalor ($Av = \lambda v$). Si no vale la igualdad se sigue iterando hasta la máxima cantidad de repeticiones. El algoritmo finaliza si se encuentra el autovalor o si se iteró lo suficiente. Se detalla el pseudocódigo en **Algorithm 8**:

Algorithm 8: método_potencia(Matriz A, vector x, int repeticiones) → (int λ,vector v)

```
   /* los vectores son matrices de dimensiones v ∈ IR^{mx1} */
1  vector v ← x
2  int i ← 0
3  do
4  |  Matriz y ← A * v
5  |  int k ← ||y||_2
6  |  v ← 1/k * y
7  |  k ← ||v||_2^2
8  |  λ ← v^T * A * v
9  |  λ ← λ / k
10 |  i ← i +1
11 while (i < repeticiones && Av ≠ λv);
   Data: Calcula el autovalor dominante A con su autovector asociado
```

En este pseudocódigo vamos a calcular los autovalores y autovectores correspondiente de la matriz X, donde la matriz D que es diagonal, contiene los autovalores y la matriz U la cual tendrá

como columnas los autovectores correspondiente. Para realizar esta operación se usara el método de la potencia y además el método de deflación, el cual consiste en una vez calculado un autovalor α y su autovector v correspondiente hacemos por medio de la siguiente formula $X = X$ - $\alpha\ vv^t$, que el autovalor α que es dominante en la matriz X, pase a no ser lo y se convierta para la matriz resultante autovalor 0 y el autovector v siga siendo asociado a él. Se detalla el pseudocódigo en **Algorithm 9**:

Algorithm 9: generación_U_D(Matriz A, Matriz U, Matriz D, int alpha)

```
   /* A, U, D tienen las mismas dimensiones. */
1  autovector ← matriz(A.filas(), 1);
2  for ( 1 ∈ [1, ..., alpha] ) {
       /* Se crea un vector x_0 con valores random diferentes de 0 y se lo
          normaliza. */
3      x_0 ← matrizRandom(A.cantFilas(), 1)
4      x_0.normalizar()
       /* Se utiliza el método de la potencia con 500 repeticiones para obtener
          el autovector y autovalor asociado de A. */
5      autovalor, autovector ← metodo_potencia(A, x_0, 500)
       /* Se pone al autovector obtenido como la columna i de U, y al autovalor
          como el elemento (i, i) de D. */
6      U_i ← autovector
7      D_ii ← autovalor
       /* Deflación */
8      autovector.normalizar()
9      B ← autovector * autovector^T
10     B ← B * autovalor
11 }
   Data: Generación de matrices U y D.
```

3. Experimentación

3.1. Robustez del método

En este experimento, buscamos analizar que tan sensible al ruido es nuestro programa, para eso vamos a utilizar dos tipos de ruido uno gaussiano y otro uniforme, el ruido se va a aplicar a la matriz tiempo que es la que tiene la suma de todos los elementos por donde pasa el rayo, de esta forma estaríamos simulando que cuando se traza un rayo este no lo hace de forma exacta sino que tiene una determinada variación en función del sobrecalentamiento del sistema o de las ondas estáticas del tomógrafo.

En el experimento de ruido gaussiano, vamos a sumarle o restarle un valor a cada píxel en función de de una distribución normal con normal constante y desvió estandar variable, Para la recreación de este ruido vamos a usar la función que tenemos en C++ llamada normal distribution $< double >$ la media para esta función va a ser siempre cero porque no nos interesa agregarle un offset a la matriz tiempo, y luego vamos a iterar el desvió estándar de 0 a 20.000 estos números son relativamente arbitrarios, tomados a partir del valor en modulo máximo que normalmente toman los valores de la matriz tiempo.

La imagen que va a ser evaluada, es de un tamaño original de $512x512$ pero la vamos a reducir 4 veces a un tamaño de $32x32$ ya que la idea principal de este experimento es evaluar como varia el ECM a medida que le agregamos distintos niveles de ruido, por lo cual no es demasiado necesario tener una imagen muy grande.

La imagen que vamos a utilizar para este experimento es una imagen que extrajimos de la pagina que nos recomienda la catedra, originalmente la obtuvimos en formato dicom, pero lo convertimos de dicom a un archivo .csv usando el programa provisto por la catedra porque el formato dicom es un formato especial que contiene muchos mas datos, no solo la imagen, lo convertimos a .csv porque no queriamos perder la calidad, ya que las imagenes en el dicom estan guardados en 16bits y no conseguimos una forma mejor de guardar la imagen y que sea facil de leer, si hubieramos optado por png este se guardaría en 8 bits por lo que perdemos muchisima información de la imagen.

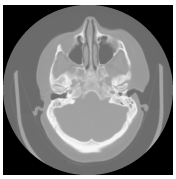


Figura 7: Imagen para la experimentación

Para el calculo del ECM usamos nuestro código, y la comparación la realizamos entre archivos .csv de esta forma nos garantizamos de que no haya ningún tipo de perdida de calidad entre la imagen original y la imagen reconstruida.

El experimento de ruido uniforme, vamos a variar el rango de ruido que vamos a sumar un valor entre $\alpha y - \alpha$ que va a ser la variable de testeo el cual lo vamos a establecer entre 0 y 20.000 por la misma razón que sucede en el experimento de ruido gaussiano, para la representación del ruido vamos a usar la función uniform real distribution $< double >$.

Luego de tener ya una matriz tiempo con información ruidosa, vamos a realizar la reconstrucción de la matriz para ultimo compararla con la original sin ruido y estudiar que es lo que sucede con su error cuadrático medio.

La forma en la que vamos a tomar las rectas es la que toma la mayor cantidad de rectas que se puedan hacer, de esta forma nos vamos a garantizar que, en caso de que haya alguna diferencia esta sea por culpa del ruido aplicado y no por la falta de información a la hora de trazar rectas sobre la imagen.

Por ultimo vamos a graficar como evoluciona el error cuadrático medio en función de la cantidad de ruido que le vamos a ir agregando a la imagen.

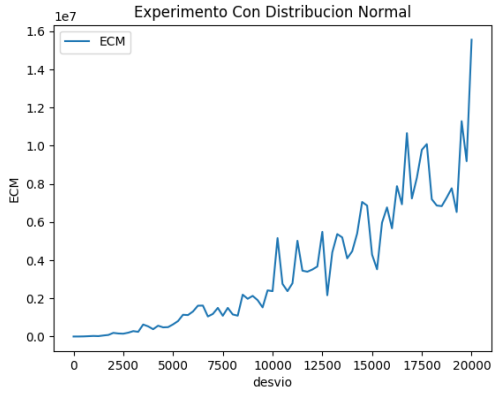


Figura 8: Imágenes con ruido gaussiano.

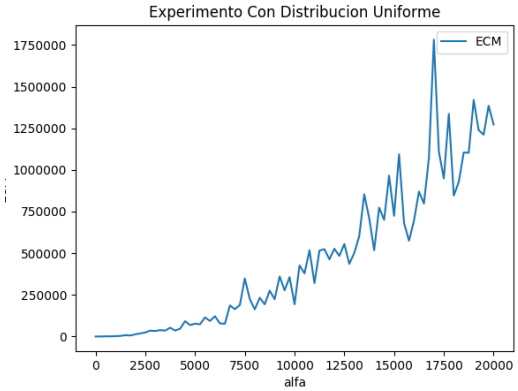
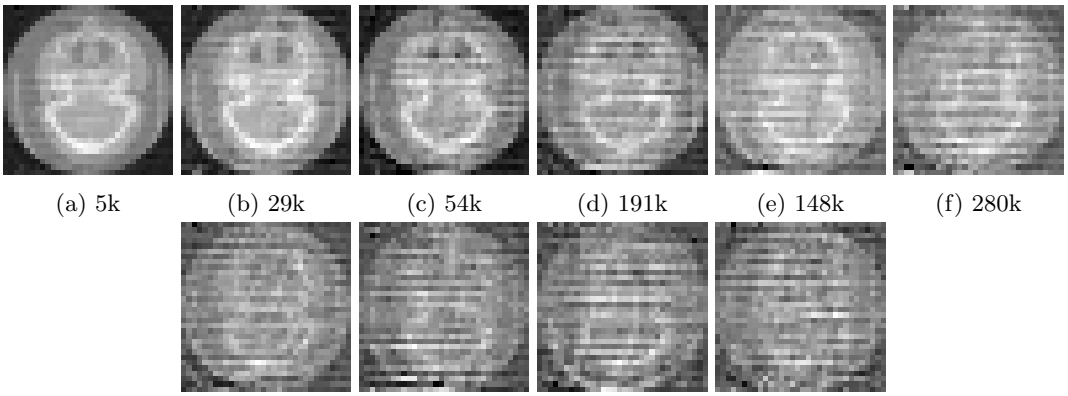


Figura 9: Imágenes con ruido Uniforme.

Se puede apreciar tanto en el experimento con ruido gaussiano, como con el de la distribución uniforme, que tienen un crecimiento que a grandes rasgos parece a una curva polinomial, pero, a partir de los 6000 puntos de niveles de ruido aproximadamente la gráfica empieza a tener picos muy grandes tanto positivos como negativos respecto a una curva polinomial, por lo cual esto nos parece interesante para analizar.

Otra cosa que no podemos apreciar es que relación hay con la imagen y con el ECM porque, quizás a partir de 1000 de ECM ya la imagen esta completamente perdida y el resto del gráfico estamos comparando la imagen original con una imagen completamente destruida y lo que observamos es la evolución del ECM de una imagen con otra de ruido aleatorio. Por esta razón vamos a ver como son las imagen y cual es el nivel de ruido que tienen

Vamos a mostrar de arriba a abajo y de izquierda a derecha la imagen reconstruida con ruido generado por una distribución normal de 500 a 7000 aumentando de manera constante en 500 por cada imagen y de pie de foto vamos a figurar cual es el ECM que tiene cada una de las imágenes



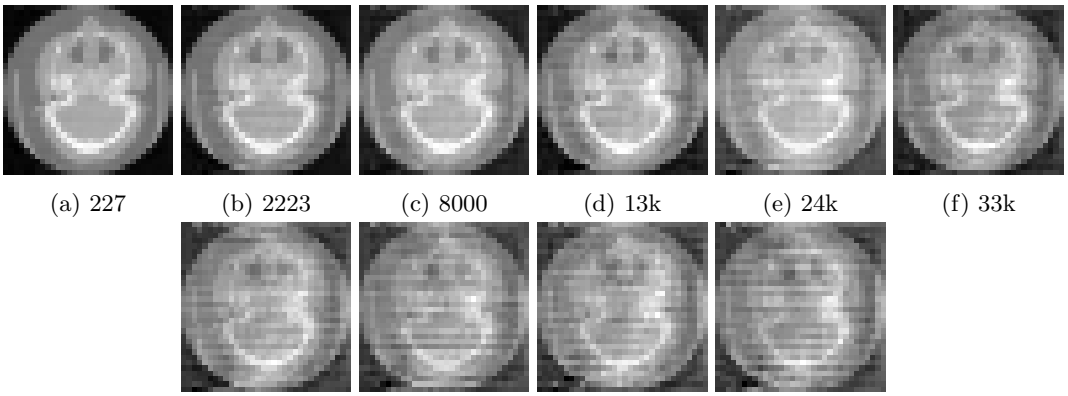
(a) 5k (b) 29k (c) 54k (d) 191k (e) 148k (f) 280k

(g) 629k (h) 383k (i) 480k (j) 630k

Figura 10: Imágenes con ruido gaussiano

podemos ver que a partir de los 3000 de desvió estándar, la imagen ya pasa a estar con un ruido tan grande que los detalles de la imagen se pierden, por lo que podríamos decir que ya a partir de esta intensidad de ruido la imagen pasa a ser inconstruibles y a pesar de que intentemos aplicar algún procedimiento para la reducción de ruido ya perderíamos la imagen. Esto tiene un poco de sentido si lo relacionamos con su error cuadrático medio, porque a partir de los 2500 de intensidad de ruido, la imagen empieza a tener picos y saltos y esto se debe a que la imagen pasa a estar completamente distorsionada.

Luego si miramos las imágenes con ruido uniforme tenemos un panorama muy similar.



(a) 227 (b) 2223 (c) 8000 (d) 13k (e) 24k (f) 33k

(g) 35k (h) 35k (i) 92k (j) 77k

Figura 11: Imagenes con ruido uniforme

Respecto al ruido uniforme, las imágenes mantienen bastante mas la forma en terminos generales podríamos decir que el método es mas susceptible al ruido gaussiano que al ruido uniforme. Pero esto tampoco es una afirmación porque para esto deberíamos asegurarnos que la deformación de una imagen es igual a la otra si le aplicamos distintos tipos de ruido, pero no es la idea de probar eso sino, experimentar con distintos tipos de ruidos sobre nuestra reconstrucción.

3.1.1. Conclusión

Podríamos decir que tiene una resistencia aceptable al ruido, porque los valores por los que hemos experimentado van desde el 0 hasta la norma 1 del vector tiempo y esto implica que podría hasta mas de duplicar el valor que existe en el vector. Y por lo que hemos visto, en las figuras **Figura 10** y **Figura 11** la imagen empieza a volverse irreversible a partir de los 3500 o 4000

unidades de intensidad cosa que tiene cierto sentido si lo analizamos con las figuras **Figura 8** y **Figura 9** que a partir de estos valores, el ECM empieza a tener grandes picos lo cual es debido a que la imagen esta tan rota que pierde sentido el hecho de comparar la imagen real con la reconstruida porque el ruido es tan grande que la reconstrucción es totalmente inválida, luego también podemos decir que nuestro programa soporta hasta 50.000 unidades de ECM, cosa que para las imagenes es tanto ruido que a nuestro criterio seria dificil de replicar a menos que los sistemas en los que está construido el tomografo sea muy malo.

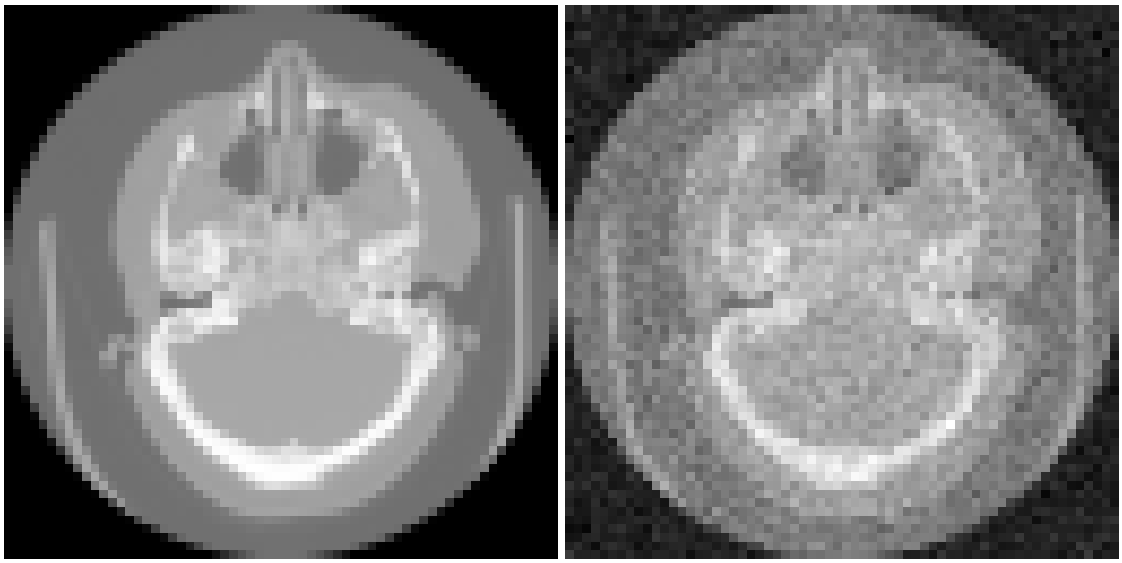
3.2. Estudio de la discretización sobre matriz D

En este experimento, vamos a aplicar una cantidad constante de rectas a la imagen tomográfica, la cual va tener ruido gaussiano. Una vez que tengamos la matriz de distancia D, intentaremos aprovechar al máximo la información que contiene, discretizando la matriz D, para comprobar si podemos reducir el ECM y tener una buena reconstrucción de la imagen resultante en dimensiones mas pequeñas respecto a quien le aplicamos los rayos.

Para hacer esta experimentación vamos a usar una imagen con formato PGM de 8 bits, el cual se creo, trasformado un CSV de una imagen de formato $DICOM$ de 16 bits a formato PGM de 8 bits.
Esta imagen tomográfica tiene un tamaño de 512x512, entonces lo que hicimos fue reducir la imagen a un tamaño de 64x64 (como se ve en la **Figura 12(a)**), para poder hacer la experimentación. El motivo de la reducción de la imagen fue el tiempo de ejecución, pues tomaba demasiado tiempo a la hora de realizar la experimentación.

A la imagen tomográfica de la **Figura 12(a)**, le hemos agregado ruido gaussiano como se muestra en la **Figura 12(b)**. La imagen de la **Figura 12(b)**, le vamos aplicar el generador de rayos de avance cuadrático, para poder reconstruir la imagen resultante, y asi analizar entre la imagen de la **Figura 12(a)** y la imagen resultante el error cometido por medio de la métrica del ECM.

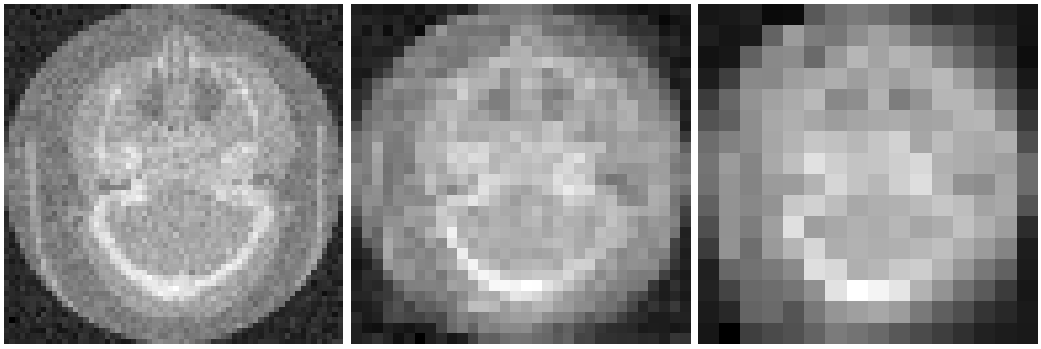
Al realizar cada experimentación, una ves aplicado una cantidad de rayos, esa cantidad de rayos se mantiene constante hasta terminar dicha experimentación.



(a) Imagen tomográfica. (b) Imagen tomográfica con ruido gaussiano.

Figura 12: Imágenes tomográfica de tamaño 64x64

Para el **experimento 1**, hemos aplicado 31504 rayos a la imagen de la **Figura 12(b)**, entonces una ves que obtenemos la matriz de distancia D y el vector de tiempo t, vamos a aplicar una cierta cantidad de veces (0, 1 y 2) la discretización explicada anteriormente.



(a) ECM:161.49 (b) ECM:.265.862 (c) ECM:290.384

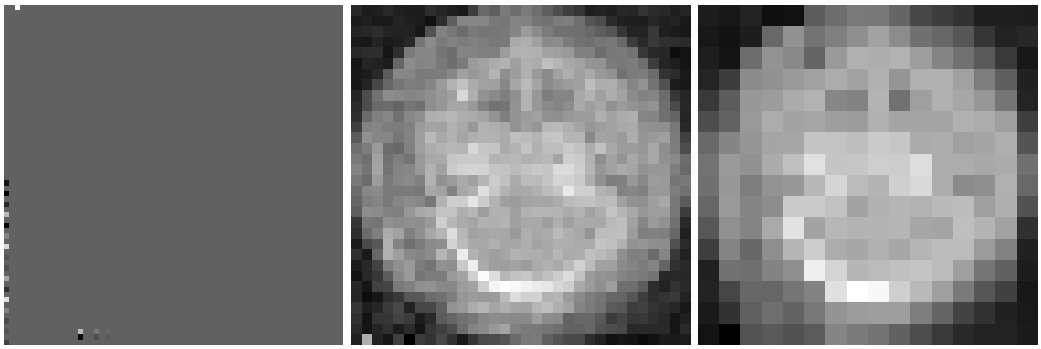
Figura 13: Experimento 1

Aquí podemos observar que la imagen de la **Figura 13(a)**, donde no se uso la discretización, por lo tanto la imagen resultante tiene un tamaño de 64x64, notamos que al aplicar 31504 rayos la reconstrucción es bastante buena y su ECM se podría considerar despreciable.

Ahora mirando la imagen de la **Figura 13(b)** de tamaño 32x32 y la imagen de la **Figura 13(c)** de tamaño 16x16, podemos notar que el ECM va aumentando a la hora de discretizar la matriz D (cantidad de discretización 1 y 2 respectivamente).

Cuando se aplica una cantidad excesiva de rayos, la reconstrucción de la imagen resultante es buena aplicando discretización 0 veces, pero si aplicamos discretización 1 o 2 veces, lo que refleja es el aumento del ECM.

Para el **experimento 2**, vamos hacer lo mismo que hicimos en el **experimento 1** pero ahora aplicando 15752 rayos.



(a) ECM:4.11802e+18 (b) ECM:309.14 (c) ECM:335.431

Figura 14: Experimento 2

Aquí podemos ver que la imagen de la **Figura 14(a)**, donde no se uso la discretización, por lo tanto la imagen resultante tiene un tamaño de 64x64, podemos observar que al aplicar 15752 rayos, la reconstrucción de la imagen no refleja nada bueno para nuestro estudio y además su ECM es demasiado alto.

Ahora mirando la imagen de la **Figura 14(b)** de tamaño 32x32, notamos que su ECM comparado con la imagen **Figura 14(a)** habido una gran disminución, solo aplicando una vez la discretización, además la reconstrucción fue muy satisfactoria para el estudio.

A la imagen de la **Figura 14(c)** de tamaño 16x16, a la cual se le aplico 2 veces la discretización, observamos que su ECM subió un poco respecto a la **Figura 14(b)**, pero esa diferencia lo podríamos considerar despreciable. Además tanto la **Figura 14(b) y 14(c)** son muy buenas reconstrucciones para análisis de tomografía computada.

Para el **experimento 3** lo que vamos hacer a diferencia del **experimento 1 y 2**, es aplicar 3612 rayos, que es un numero menor a de las dimensiones (64*64 = 4096), en teoría no deberíamos poder reconstruir la imagen ya que son muy pocos rayos para las dimensiones de la imagen.

(a) ECM:7.90123e+20 (b) ECM:7.39993e+23 (c) ECM:476.626

Figura 15: Experimento 3

Notamos que en la **Figura 15(a)** de tamaño 64x64 y la **Figura 15(b)** de tamaño 32x32, las cuales recibieron discretizaciones 0 y 1 respectivamente, las dos reconstrucciones son muy malas y además los ECM son demasiados altos, pero la **Figura 15(c)** que tiene un tamaño de 16x16, a la cual discretizamos 2 veces, observamos que la reconstrucción es muy buena para el estudio de tomografía computada y además tiene un ECM muy bueno.

En resumen, podemos notar que a la hora de discretizar la matriz D que contiene la información de una cantidad de rayos constante, podemos hacer un alta reducción del ECM y reconstruir la imagen es un estado para nuestro estudio de tomografía computada.

Una buena aplicación de esto, es buscar la mínima cantidad de rayos, que contenga una buena información, para poder usarla bien a la hora de discretizar la matriz D y así hacer buenas reconstrucciones de imágenes pero de menor dimensiones.

3.3. Estudio de la cantidad de rayos

La aplicación de este trabajo está pensada para ser utilizada en tomógrafos. Como todos sabemos recibir mucha radiación es malo, por lo tanto en esta sección vamos a responder dos preguntas:

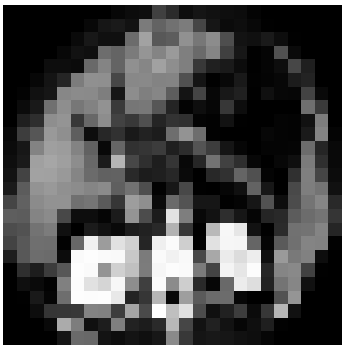
- ¿Cuál es la mínima cantidad de rayos que puedo lanzar para obtener una imagen con la que un médico pueda trabajar?
- ¿Cuánto tarda el programa en generar esa imagen?

Se usó como entrada la imagen 6 en formato PGM. La imagen original es cuadrada y tiene cien píxeles de lado, pero se la redujo a la cuarta parte.

3.3.1. Cantidad de rayos

En la **Figura 16** podemos ver un conjunto de reconstrucciones realizadas con nuestro algoritmo. Para una mejor visualización están ordenadas según el error cuadrático medio.

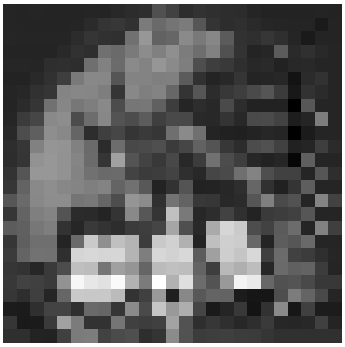
(a) Imagen original luego de la reducción



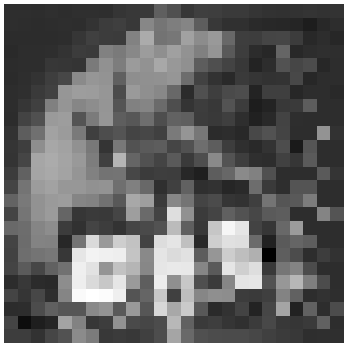
(b) #rectas = 3387, ECM = 0.55

(c) #rectas = 1764, ECM = 13.52

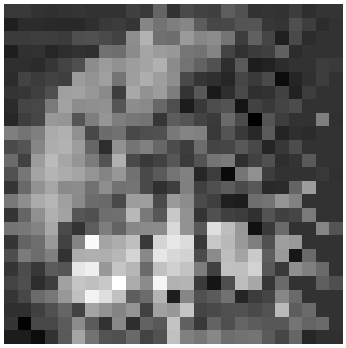
(d) #rectas = 1179, ECM = 16.15



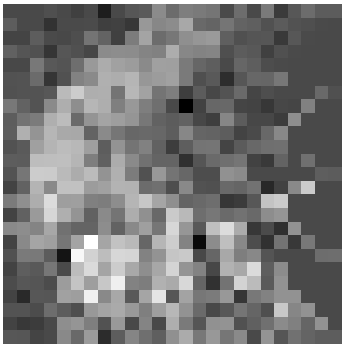
(e) #rectas = 900, ECM = 1007.04



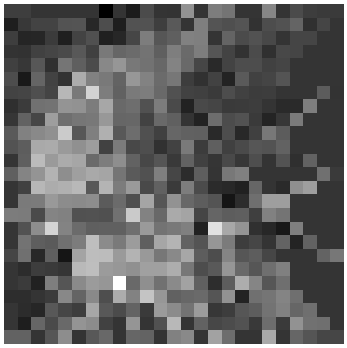
(f) #rectas = 624, ECM = 1468.19



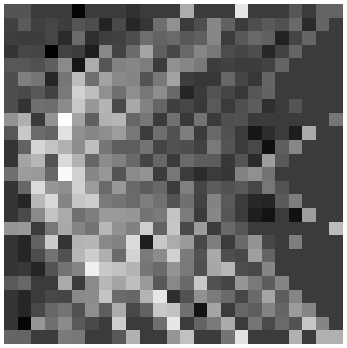
(g) #rectas = 486, ECM = 2429.75



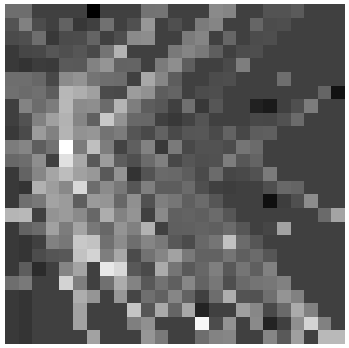
(h) #rectas = 348, ECM = 4563.6



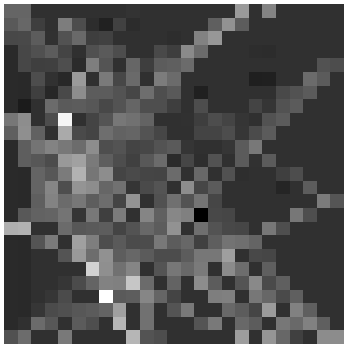
(i) #rectas = 228, ECM = 3729.08



(j) #rectas = 135, ECM = 5270.19



(k) #rectas = 87, ECM = 5236.45



(l) #rectas = 60, ECM = 4636.68

Figura 16: Comparación de las reconstrucciones

Como es esperado se ve que cuantas más rayos se emitan mejor es el resultado. Pero haciendo un análisis más fino podemos decir que en las cuatro imágenes con menos cantidad de rayos no se distingue ninguna parte del cerebro. En cambio en las siguientes cuatro imágenes ya se puede visualizar la forma de la cabeza y las tres secciones blancas de la parte inferior. Pero la buena reconstrucción ocurrió cuando se utilizaron más de mil rayos, en esos casos la imagen es realmente muy similar a la original. También es bueno aclarar que en este caso el error cuadrático medio parece ser una muy buena métrica para ver si una imagen es similar a otra porque en imágenes similares dio un numero pequeño y en las imágenes difusas un número muy grande.

Además de la comparación visual nos interesa ver como influye en el error cuadrático medio la cantidad de rayos lanzados. Para esto realizamos la reconstrucción cambiando más la cantidad de rayos lanzados y tomamos el error cuadrático medio.

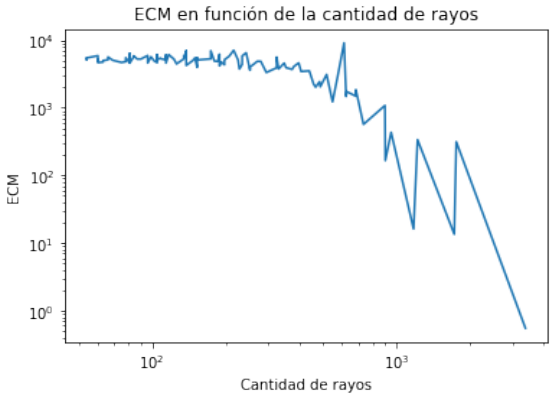


Figura 17: Calidad de las reconstrucciones

Se puede observar en 17 que si es usan menos de mil rayos el error es muy alto, pero si se usan más de mil rayos el error es del orden de 100 salvo que se usen más de dos mil rayos.

3.3.2. Tiempo de ejecución

El tiempo de ejecución de nuestro algoritmo no es un detalle menor, siempre se busca el mejor resultado posible pero dentro de un período razonable. Por esta razón medimos los tiempos de ejecución de nuestro algoritmo y obtuvimos lo siguiente:

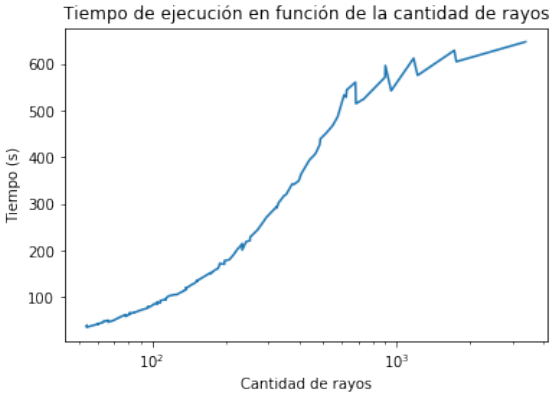


Figura 18: Tiempo de procesamiento de las reconstrucciones

En la figura 18 se observa que el tiempo de ejecución aumenta muy rápido hasta que se mantiene luego de los mil rayos lanzados.

3.3.3. Conclusiones

Si hay que reconstruir una sola imagen quizás no es tan grave esperar alrededor de diez minutos para tener la reconstrucción. En caso de que sean más imágenes se pueden utilizar más de 850 rayos para tener una reconstrucción no tan buena pero un tiempo de ejecución mucho menor. Estas recomendaciones sirven solamente para imágenes cuadradas que tengan 25 píxeles de lado.

4. Conclusiones

En este trabajo analizamos e implementamos ideas a la hora de ver como generamos los rayos (rectas) y diversas variantes, para poder establecer buenas reconstrucción de imágenes tomográficas a las cuales le agregamos diferentes ruidos. Otra análisis bastante interesante fue que a la hora de discretizar la matriz D, pudimos comprobar que se puede aprovechar al máximo la información extraída a la hora de aplicar pocos rayos a imágenes de grandes dimensiones y obtener buenos resultados cualitativos.

El error cuadrático medio es una gran métrica a la hora de hacer un análisis cuantitativo, por que determinar si dos imágenes son similares visualmente o no. En algunos casos cuando tenemos varias imágenes que debemos verificar si hubo una buena reconstrucción, es mas preciso usar ECM, que refleja cuanto error hay por píxel, en ves de dar un análisis visual imagen por imagen.

5. Anexo

5.1. Enunciado

Métodos Numéricos
1^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 el método de aproximación por cuadrados mínimos.

En muchos ámbitos de la medicina se trabaja con estudios que buscan obtener los mejores resultados posibles. Sin embargo, con el fin de evitar procesos invasivos en los pacientes o disminuir los efectos colaterales se hace uso de la tecnología. Ejemplo de esto son las tomografías computadas, donde se atraviesa el objeto de estudio con rayos X. Luego, según la proporción de la radiación inicial que llega al otro lado, se consigue estimar la densidad del objeto atravesado.

Dado que cada envío de un rayo atraviesa una sección se realizan diversos disparos con diferentes ángulos para tratar de captar diferentes combinaciones con las diferentes partes del objeto. Como además los instrumentos tienen errores en la medición es recomendable realizar repetidas veces el mismo envío para tratar de disminuir los efectos negativos asociados. Dada una cierta granularidad elegida para diagramar 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 no será determinado sino sobredeterminado, debido a que se toman múltiples medidas y a los errores numéricos en ellas. Para conocer los niveles de densidad en cada parte del objeto en la discretización basta entonces resolver este sistema, para lo que se usarán métodos númericos 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 la *simulación* sobre una imagen obtenida en una tomografía lo que nos permitirá además juzgar que tan buena es la aproximación obtenida.

El 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 una en atravesarlo. Por ejemplo, la Figura 1 muestra una posible distribución de estas señales.

Suponemos que el cuerpo se discretiza en $n \times n$ celdas cuadradas, de modo tal que en cada celda las señales de rayos X tienen velocidad constante. Si d_{ij}^k es la distancia que recorre el k-ésimo rayo X 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 de rayo X en esa celda, entonces el tiempo de recorrido de la señal completa es de

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

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

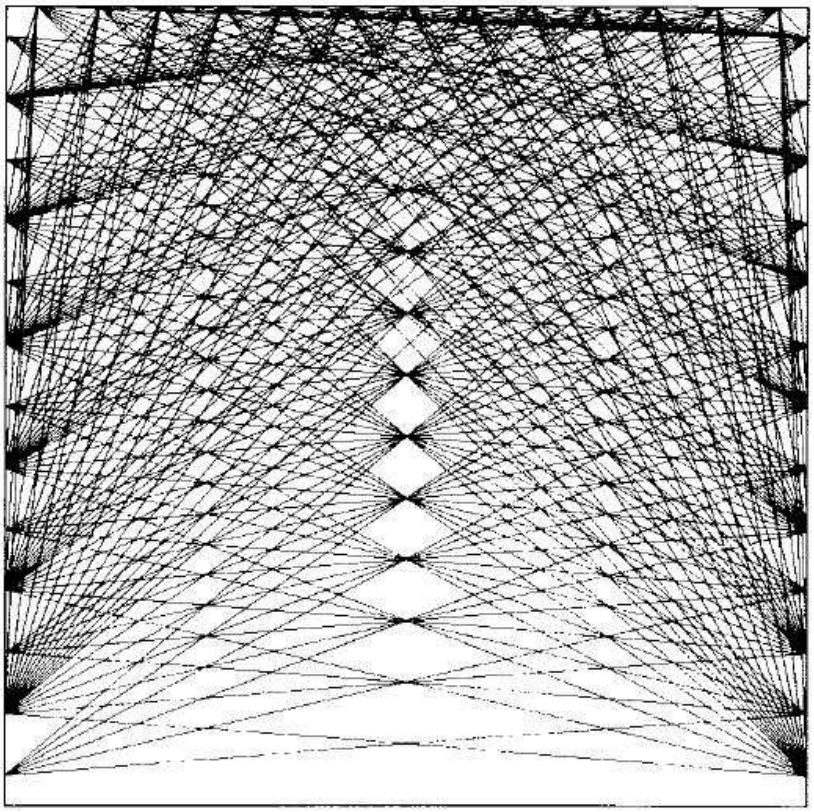


Figura 1: Ejemplo de configuración de las señales.

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 $t \in \mathbb{R}^m$, 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} 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.

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.

Enunciado

El trabajo práctico consiste en implementar un programa en `C++` que simule el proceso de tomografía y reconstrucción, realizando los siguientes pasos:

1. Leer desde un archivo una imagen con los datos reales (discretizados) de un cuerpo. Para los fines de este procedimiento, se considerará que el valor de cada pixel de la imagen corresponde a la velocidad de recorrida de las señales de rayos X al atravesar ese pixel.
2. Ejecutar el proceso tomográfico, generando un conjunto relevante de señales y sus tiempos de recorrida exactos.
3. Perturbar los tiempos de recorrida con ruido aleatorio.
4. 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.

El programa deberá tomar como parámetros los nombres de los archivos de entrada y de salida, junto con un parámetro que permita especificar el nivel de ruido a introducir en la imagen. El formato de los archivos de entrada y salida queda a elección del grupo. Para simplificar la implementación, es posible utilizar un formato propio que no sea ninguno de los formatos estándar para guardar imágenes[2].

Experimentación

Sobre la base de la implementación, se **pide** medir la calidad de la imagen reconstruida en función del nivel de ruido. Para medir el error de la imagen resultante se deberá utilizar el error cuadrático medio, definido como

$$ECM(v, \tilde{v}) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} (v_{ij} - \tilde{v}_{ij})^2}{n^2}, \tag{2}$$

siendo v_{ij} la velocidad de recorrido de la celda ij del cuerpo original, y $\tilde{v}_{ij}$ la velocidad en la misma celda del cuerpo reconstruido.

Además, se **deben** reportar los tiempos de procesamiento.

Se **deben** probar con distintas estrategias geométricas para generar las señales de rayos X (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, rango de ángulos de salida de las señales, etc.) para buscar la estrategia que minimice el error de la reconstrucción.

Por otra parte, puede ser interesante medir el número de condición de la matriz asociada al sistema de ecuaciones normales que resuelve el problema de cuadrados mínimos, para analizar la estabilidad de la resolución. 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.

[2]Se sugiere utilizar el formato pgm ya visto en el TP2.

Dataset

Para la experimentación se deberá utilizar, además de las imágenes provistas por la cátedra, alguno de los conjunto de datos provisto en el siguiente link: `http://www.via.cornell.edu/databases/`. [3] Las imágenes de estos dataset se encuentran en el formato DICOM, que es un formato de imagen en escalada de grises con una profundidad de pixel de 16 bit. Las mismas puede ser leídas mediante la función `dicomread` de Matlab/Octave. Se recomienda, convertir este formato a ppm de 16 bit.

Fecha de entrega

- Formato Electrónico: Martes 26 de Junio de 2018 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: Miércoles 27 de Junio de 2018 a las 18 hs. en la clase de laboratorio.

- Pautas de laboratorio:
  `https://campus.exactas.uba.ar/pluginfile.php/79576/course/section/12820/pautas.pdf`

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

[3] Tener en cuenta que se deberán registrar primero.

5.2. Código