

Universidad de Santiago de Chile

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

TAREA 3: RESTAURACIÓN DE IMAGENES

TALLER DE PROGRAMACIÓN

Sección: 13315-0-A-1

Profesor: Pablo Roman Asenjo

Ayudante: Ricardo Hasbun Miranda

Autor: John Serrano Carrasco

Junio 2022

Contents

1	El problema	2
2	La solución	3
2.1	Pruebas y errores	6
3	Complejidad y eficiencia	7
4	Conclusiones	7
5	Bibliografía	7
6	ANEXOS	8

1 El problema

Se considera una imagen (matriz de valores float) de 256x256 de la cual se dispone la versión corrupta y la versión original (valores positivos). La corrupción de la imagen se simula mediante la convolución con un Kernel (Imagen pequeña) de tamaño $2M + 1$, valores típicos de $M = 1, 5, 12$. El programa retorna una imagen restaurada que minimiza la suma de dichos errores al cuadrado utilizando el método del gradiente conjugado. Esta imagen debe poder visualizarse desde python utilizando un archivo que contiene los valores de cada pixel.

Por ejemplo, se tiene la siguiente imagen:



Figura 1: Imagen original.

La restauración de la imagen anterior, es:



Figura 2: Imagen restaurada.

Cabe destacar de que la imagen nunca logrará ser restaurada completamente, siendo el resultado una aproximación de una restauración.

2 La solución

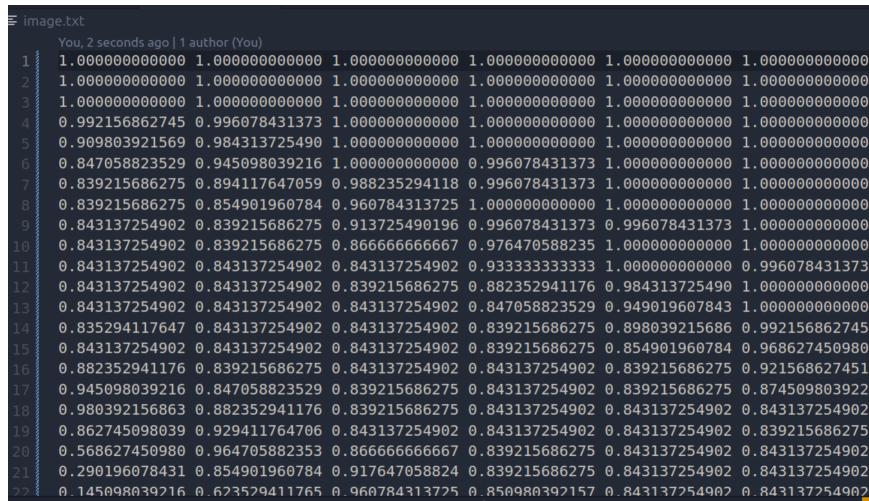
Para la solución de este problema, se tienen las siguientes clases:

- Clase Image
- Clase ImageProcessor
- Clase Optimizer
- Clase Objective

Ademas, se tienen dos archivos .py que ayudaran con el proceso:

- crearTxt.py
- cargaTxt.py

La solución es la siguiente: Primero, se debe generar el archivo.txt de la imagen. Para ello, se utiliza el archivo crearTxt.py. Se indica el nombre de la imagen a utilizar y se utiliza Numpy para crear un array correspondiente a la imagen. El resultado se guarda en un archivo de nombre "imagen.txt" y se muestra por pantalla gracias a Matplotlib.



```
image.txt
You, 2 seconds ago | author (You)
1 1.000000000000 1.000000000000 1.000000000000 1.000000000000 1.000000000000 1.000000000000
2 1.000000000000 1.000000000000 1.000000000000 1.000000000000 1.000000000000 1.000000000000
3 1.000000000000 1.000000000000 1.000000000000 1.000000000000 1.000000000000 1.000000000000
4 0.992156862745 0.996078431373 1.000000000000 1.000000000000 1.000000000000 1.000000000000
5 0.909803921569 0.984313725490 1.000000000000 1.000000000000 1.000000000000 1.000000000000
6 0.847058823529 0.945098039216 1.000000000000 0.996078431373 1.000000000000 1.000000000000
7 0.839215686275 0.894117647059 0.988235294118 0.996078431373 1.000000000000 1.000000000000
8 0.839215686275 0.854901960784 0.960784313725 1.000000000000 1.000000000000 1.000000000000
9 0.843137254902 0.839215686275 0.913725490196 0.996078431373 0.996078431373 1.000000000000
10 0.843137254902 0.839215686275 0.8666666666667 0.976470588235 1.000000000000 1.000000000000
11 0.843137254902 0.843137254902 0.843137254902 0.933333333333 1.000000000000 0.996078431373
12 0.843137254902 0.843137254902 0.839215686275 0.882352941176 0.984313725490 1.000000000000
13 0.843137254902 0.843137254902 0.843137254902 0.847058823529 0.949019607843 1.000000000000
14 0.835294117647 0.843137254902 0.843137254902 0.839215686275 0.898039215686 0.992156862745
15 0.843137254902 0.843137254902 0.843137254902 0.839215686275 0.854901960784 0.968627450980
16 0.882352941176 0.839215686275 0.843137254902 0.843137254902 0.839215686275 0.921568627451
17 0.945098039216 0.847058823529 0.839215686275 0.843137254902 0.839215686275 0.874509803922
18 0.980392156863 0.882352941176 0.839215686275 0.843137254902 0.843137254902 0.843137254902
19 0.862745098039 0.929411764706 0.843137254902 0.843137254902 0.843137254902 0.839215686275
20 0.568627450980 0.964705882353 0.8666666666667 0.839215686275 0.843137254902 0.843137254902
21 0.290196078431 0.854901960784 0.917647058824 0.839215686275 0.843137254902 0.843137254902
22 0.145098039216 0.623529411765 0.960784311775 0.856980392157 0.843137254902 0.843137254902
```

Figura 3: Archivo "imagen.txt"

La imagen es llevada a la corrupción; Se utiliza una instancia de la clase ImageProcessor y se realiza una copia del archivo .txt de la Imagen original, guardando lo anterior en un archivo de nombre "image corrupted.txt". Luego, se realiza la corrupción de la imagen, donde se modifican todos los pixeles de la imagen, se realiza una convolución con el kernel (Una imagen mas pequeña) y se le agrega ruido a la imagen. El resultado de esto se guarda en un archivo .txt de nombre "image corrupted.txt"

```

image_corrupted.txt
You, 1 second ago | 1 author (You)
1 0.0653823 You, now * Uncommitted changes
2 0.220802
3 0.267515
4 0.400987
5 0.28546
6 0.296412
7 0.448523
8 0.3288
9 0.3294
10 0.297131
11 0.314551
12 0.397723
13 0.321141
14 0.367706
15 0.412502
16 0.412742
17 0.310867
18 0.288361
19 0.220176
20 0.269268

```

Figura 4: Archivo "image corrupted.txt"



Figura 5: Imagen Corrompida

Luego, viene el metodo el metodo frprmn(), el cual es el metodo principal que utiliza todos los demas metodos, como linmin, mnbrak, brent, f1dim, f, df y otros mas asociados a las clases. A continuación, una breve descripción de los metodos mas importantes de los anteriormente mencionados:

- Frprmn: Método perteneciente a la clase Optimizer. Minimiza en N-dimensiones el gradiente conjugado, a traves de un número de iteraciones. A traves de este metodo, se utilizan otros metodos como truncate, linmin, brent, mnbrak y f1dim.
- Linmin: Método perteneciente a la clase Optimizer. Sirve de apoyo para realizar el proceso de minimizar en N-dimensiones el gradiente conjugado. En su desarrollo utiliza mnbrak y brent
- Mnbrak: Método perteneciente a la clase Optimizer. Encuentra un a, b y c tal que el minimo esta entre a y b
- F1dim: Método perteneciente a la clase Optimizer. Deja en una dimension y calcula la suma de pcom + x*xicom
- Brent: Método perteneciente a la clase Optimizer. Encuentra el minimo sin derivadas, utilizando el metodo de Brent, el cual es un metodo que mezcla distintos metodos de optimización para así encontrar de manera eficiente el minimo de una función.
- F: Método perteneciente a la clase Objective. Corresponde al valor de la función objetivo

- Df: Método perteneciente a la clase Objective. Calcula el gradiente de la función objetivo. (Gradiente = $K^*(K^*p - I_{corrupted})$). El código es bastante similar al de F, con dos líneas extras.

Además, frprmn tiene los siguientes atributos:

- p: Son los parámetros del modelo de la imagen aplanada como un vector de 256 * 256. Un arreglo de flotantes específicamente
- n: Tamaño del vector
- ftol: Tolerancia con que termina el algoritmo
- iter: Retorna cuantas iteraciones realiza el algoritmo
- fret: Retorna el valor de la función
- func: Función que se va a minimizar
- dfunc: Gradiente de la función
- max it: Números máximos de iteraciones

Volviendo a la solución, de forma resumida; se declaran variables y arreglos a utilizar. Se calcula la función a minimizar y se calcula el gradiente. Iterando, se asigna a cada posición de x_i el valor de -gradiente. Se entra a un for principal que tiene un máximo de iteraciones, y se llama a linmin, donde se modifica p para obtener un mínimo en la dirección de x_i . Luego se utiliza el método truncate, donde elimina todo valor negativo de p y los reemplaza por 0. Continuando, en fp se guarda el resultado de la función que asigna en x_i el valor del gradiente. Se construye la dirección en la que se va a minimizar y si el gradiente es 0, entonces se implica que se ha encontrado un mínimo. Caso contrario, se continua el proceso.

En F y en df se realiza el proceso de convolución, donde de a poco se va restaurando la imagen a través de la modificación de los pixeles de esta. Además, se utiliza un kernel, el cual es una imagen más pequeña, para así restar pixeles y obtener pixeles similares a los de la imagen original.

Una vez se ha realizado todo el proceso anterior, se crea una nueva imagen, la cual recibe el valor de f y el valor de N, y esto se guarda en un archivo .txt llamado "restored.txt". Se utiliza el archivo .py "cargaTxt.py" para así cargar la imagen y mostrarla a través de matplotlib junto con la imagen original, terminando así el proceso de restauración de la imagen.

```

restored.txt U
restored.txt
1 0.399823
2 0.854137
3 0.692848
4 0.504105
5 1.08953
6 0.725714
7 1.02957
8 1.09962
9 0.323526
10 0.638829
11 0.346523
12 0.489497
13 0.975573
14 0.669407
15 0.738094
16 0.772372
17 0.243001
18 0.381066
19 0.819664
20 0.648756
21 0.857453
22

```

Figura 6: Archivo "restored.txt"

En el main del programa, se hace un ciclo for con N repeticiones del programa, para poder calcular el tiempo de resolución del proceso de restauración.

Como el código no fue optimizado lo suficiente, pueden haber problemas de memoria ("terminated (killed)") si es que se prueba sobre 3 iteraciones y con una tolerancia entre -7 y -12. Aún así, se recomienda probar el programa con una iteración y tolerancia -12 para así ver el tiempo de ejecución del programa.

El proceso de creación de la solución fue basado

ALTAMENTE en lo enseñado en clases, como también en el libro "Numerical Recipes in C" [1]. Aún así,

se intentó modificar lo que mas se pudo para que el código no fuera exactamente igual a otras entregas.

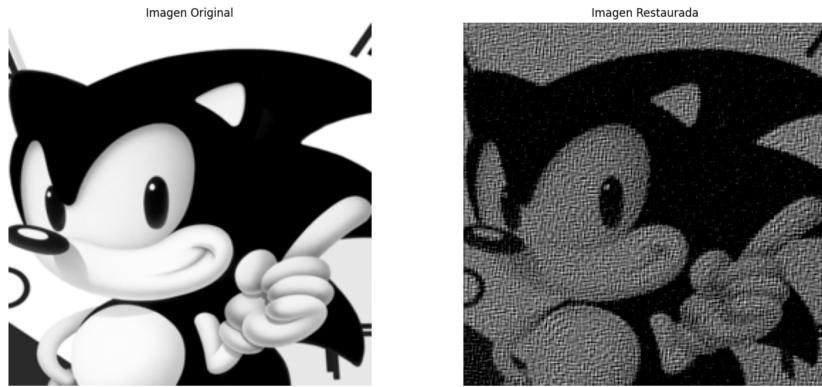


Figura 7: Resultado final

2.1 Pruebas y errores

Hubieron multiples pruebas y multiples errores durante el proceso de creación. A pesar de que el código había sido dado por el profesor, al pasar algunos códigos de .c a .cpp, hubieron problemas de sintaxis como también de funcionamiento. El test de Optimizer costó un poco, ya que se debía involucrar herencia, lo cual no se tenía idea de como hacer. Pero la guía del profesor durante la clase fue de gran ayuda.

Al intentar arreglar un fallo, se terminó rompiendo una parte del código, lo que provocaba que no fuera posible hacer restauraciones con una tolerancia menor a -5. Esto fue un grave problema, ya que eso implicaba que las imágenes restauradas se veían muy borrosas, por lo que la única solución fue restaurar un commit antiguo e ir probando si funcionaba o no. Lo anterior también trajo multiples problemas al momento de ejecutar el código, ya que las iteraciones de brent fallaban o se iban hacia valores que no debían, también hubieron problemas de memoria que entorpecían con las pruebas del código, probablemente debido a que el código no fue optimizado completamente.

Realizando pruebas con tres distintas imágenes, se llegó a distintos resultados:

- Imagen Mapache: De 9-30 iteraciones de brent
- Imagen "Sonic": De 100 - 120 iteraciones de brent. (A pesar de esto, la imagen restaurada es obtenida sin alteraciones)
- Imagen "Touhou": De 15 - 30 iteraciones de brent

Se descubrió que con ciertas imágenes, la restauración puede quedar un poco borroza, oscura, o no tan restaurada como lo esperado. Sin embargo, debido a que esto es una aproximación, es esperable que el resultado final no sea completamente igual a la imagen original. Sin embargo, que tan exacto sea el resultado final también va a depender de la imagen, como se puede apreciar a continuación:

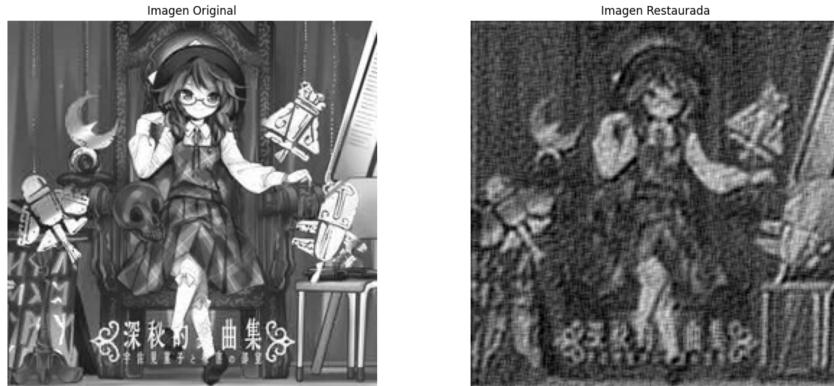


Figura 8: Un resultado final donde la imagen restaurada es mas borrosa y oscura que la original

3 Complejidad y eficiencia

A pesar de que hubieron muchos problemas, pruebas y errores con el proceso de creación, resulta que si se tiene un Algoritmo eficiente, no tanto como lo esperado, ya que no se optimizó lo suficiente, pero es posible restaurar la imagen entre 9 a 25 segundos, lo cual en si es bastante rápido.

Si el programa logra realizar 10 iteraciones, entonces tiende a demorarse entre 2 a 4 minutos. Sin embargo, lo anterior se puede ver entorpecido por la falta de optimización, ya que eso conlleva problemas de memoria y Linux puede decidir terminar o matar el proceso prematuramente.

Respecto a la complejidad, considerando que no se tiene un código bastante eficiente, se puede decir con certeza que se tiene una complejidad muy grande, superando con creces a n^*n . Sin embargo, debido al origen de la gran mayoría del código y al largo de este, es difícil saber con exactitud que complejidad tiene.

Aún así, el código logra realizar el proceso de restauración, por lo que se tiene un código no muy eficiente, pero funcional.

Debido a que el código requiere de archivos .py y .cpp, se recomienda ejecutar la instrucción "make run" para así ver como funciona el algoritmo en su totalidad, ejecutando los archivos correspondientes para realizar todo el proceso de corrupción y restauración de imágenes.

4 Conclusiones

Finalmente, se logró completar la tarea cumpliendo los objetivos propuestos, donde se tiene un algoritmo que es capaz de resolver el proceso de corrupción y restauración de una imagen en un tiempo bastante decente.

A pesar de las complicaciones, se logró programar la tarea en C++ aprendiendo nuevas cosas en el proceso y se espera que este conocimiento sirva de experiencia para las futuras entregas de la Asignatura. Cabe destacar de que esta fue una tarea un poco difícil y confusa, especialmente ya que se tuvo que traducir código de C a C++, pero finalmente, se logró tener un código que logra realizar lo solicitado sin complicaciones mayores.

5 Bibliografía

1. William H. Press ... [y otros]. (1992). "Numerical recipes in C : the art of scientific computing". Cambridge [Cambridgeshire] ; Nueva York :Cambridge University Press.

6 ANEXOS

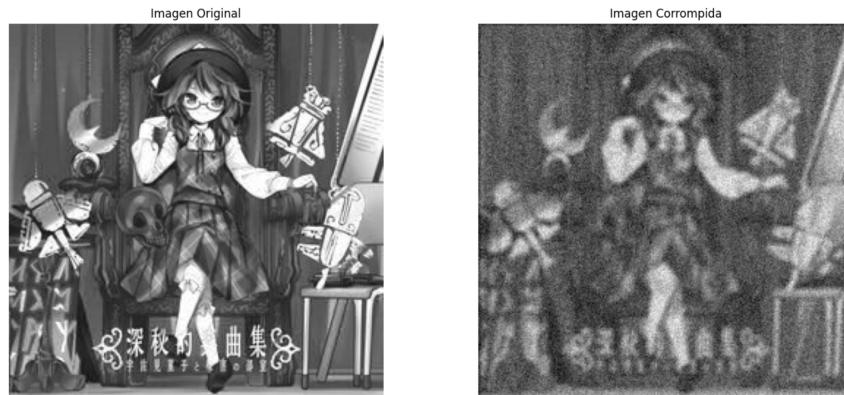


Figura 9: Comparación entre una imagen original y una imagen corrompida

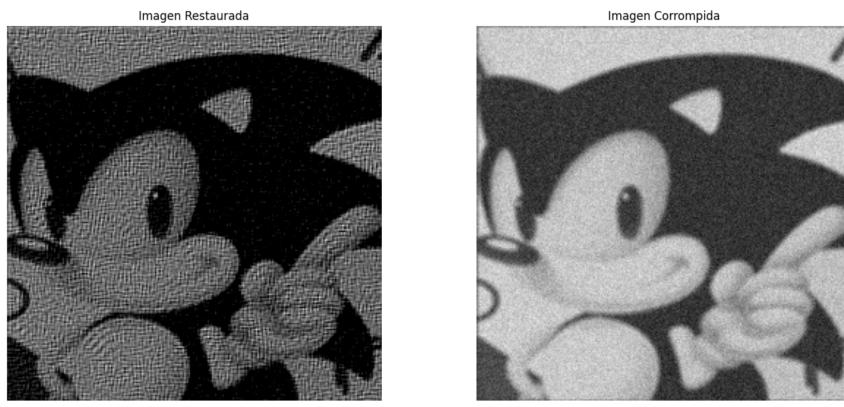


Figura 10: Comparación entre una imagen restaurada y una imagen corrompida

```
brent xmin: 0.0170293581
iteracion: 92
brent xmin: 0.0119930059
iteracion: 93
brent xmin: 0.0648587719
iteracion: 94
brent xmin: 0.0298205502
iteracion: 95
brent xmin: 0.0302683916
iteracion: 96
brent xmin: 0.0299840923
iteracion: 97
brent xmin: 0.0189430993
iteracion: 98
brent xmin: 0.0184516925
iteracion: 99
brent xmin: 0.0441842079
iteracion: 100
brent xmin: 0.0091214264
Too many iterations in fprmn%
Tiempo de ejecucion: 14.2182[s]
Tiempo promedio: 14.2182[s]
```

Figura 11: Iteraciones de Brent. Notar que con la imagen "Sonic1.png", las iteraciones superan las 100.