Documentation of project

Model Alpaca Emblem

Teacher: Alexandre Bergel
Date: 16 de agosto de 2019
Link de Trabajo

Author: Sebastián Sepúlveda A
User GitHub: sesepulveda17
ID: 19.640.031-1

Abstract

El presente informe tratará de explicar los detalles detrás de la elaboración del programa *reparador de imágenes 2D*, el cual busca reconstruir una imagen que tenga un sector desconocido en blanco **Ver Figura ??**. En el proceso de la construcción del algoritmo se ocuparon elementos de diferencias finitas, librerias de python para procesamiento de imágenes y resolución de matrices, en conjunto con el uso de matrices *sparse*. Gracias a esto último se logró obtener una imagen "reparada" con una optimización en el tiempo de resolución de la ecuación $A\vec{u} = \vec{b}$ y un óptimo uso en memoria.

Resolución problema de Laplace

Al inicio se comenzó obteniendo los vectores de colores de la imagen, lo que nos entrega los valores del dominio sobre el cual vamos a trabajar y las condiciones de *borde tipo Dirichlet* (colores conocidos del borde del dominio). La forma de obtener estos valores fue recorrer la matriz o paleta de colores hasta encontrar un vector blanco, en ese instante se comienza a consultar a los "vecinos" del vector blanco, si son colores distintos a blanco entonces pertenecen a un caso borde y lo guardamos, en caso contrario no hacemos nada. Dado a que en las condiciones del problema no se especifica claramente, se asumió que el sector desconocido podría estar en una esquina o en un borde de la imagen original, por lo que el algoritmo hace la consulta de los colores vecinos a cada color desconocido respecto a un sector que pertenezca a la imagen, sin superar los bordes de la matriz de la imagen. **Nota:** El programa acepta vectores de color de largo 3 como de largo 4.

Luego de obtener los indices de las incógnitas, podemos obtener la matriz A, primero creando una matriz de ceros de tamaño $N = largo([\texttt{vector de incognitas}])$, y luego recorriendo el `vector de incógnitas` colocando $-4's$ en la diagonal por cada indice del elemento del vector donde "estamos ubicados", pues son las incognitas de nuestro problema, y sumar $1's$ según cada color desconocido del dominio que sea vecino del color desconocido. Como se mencionó en el párrafo anterior, al considerar los bordes de la imagen como un caso posible donde se ubique el sector desconocido, es neceario agregar en la consulta a los vecinos *sólo* a elementos del dominio, por lo que en caso de que los indices que consultemos no estén en la matriz de la imagen (están fuera de los limites de la imagen) se debe sumar 1 en la posición de la diagonal donde se ubiquen estos indices, disminuyendo los puntos del *sténcil* de 5 puntos.

Finalmente, con el resultado de la matriz A y conociendo el vector de las condiciones de bordes b, podemos resolver el problema con la función de `numpy: np.linialg.solve`. Como resultado se entrega la imagen en formato `.png`

Matrices sparse

Dado a que la matriz A que almacena los coeficientes de las incógnitas posee muchos ceros, ocupa un gran espacio en memoria y en el tiempo de ejecución. La solución a este problema fue ocupar una matriz *sparce* que solo almacena los datos distintos a cero. La *sparce matrix* que se utiliza para el programa es de formato `CSR: Compressed Sparse Row format` [1], la cual guarda la información distinta de cero de la matriz en 3 listas: `data` (información distinta de 0c), `indices` (guarda los indices de la columna de la información de la matriz), y en `indpr` (guarda el indice "donde comienza" la fila).

[1] Revisar Scipy matrices documentation

Como test del método con matrices *sparse* se utilizó la **Figura ??** de tamaño 300px. La comparación entre la matriz con ceros y la matriz sparse de la misma imagen se resumen en la siguiente **Tabla 1** [2]:

	Con matriz sparse	Sin matriz sparse
N^o de bytes (Kb) en memoria	336.2	605000.00
Tiempo de ejecucion (s)	9.2	75.8

Cuadro 1: Comparación entre matriz *sparse* y matriz con ceros

Utilidad y limitaciones

Este tipo de algoritmos para reparar imágenes es relevante para variados usos, como reconstrucción de retratos de personas, reconstrucción de imagenes con mala resolución o deterioradas, o incluso en reparación de videos que tienen subtitulos "pegados". Sin embargo, en esta oportunidad, al ser un algoritmo que utiliza solamente diferencias finitas, y sólo realiza una sola revisión sobre el sector en blanco, no es posible obtener una resolución alta de la imagen, ni tampoco es posible determinar un color "claro" para incognitas que estén muy alejadas del borde del dominio, es decir, muy "centradas". Por ende, una primera forma simple de mejorar el programa es hacer "reparaciones" iterativas sobre el dominio incognito de la imagen, volviendo a hacer.

[2]Para obtener los bytes de los datos de la matriz se ocupó la función `nbytes` de numpy y scipy