Proyecto:
Algoritmo para la extracción de datos de archivos KMZ del radar meteorológico

Equipo 3

Morfín Chávez Guillermo
Pérez Cázares Evelyn Alejandra
Pineda Ibarra David Arodi
Preciado Manríquez Luis Alejandro

26 de junio de 2021

Licenciatura en Física, Departamento de Física, CUCEI, Universidad de Guadalajara.
Av. Revolución 1500, S.R. C.P. 44430, Guadalajara, Jalisco, México.

Resumen

Se escribió un algoritmo con la función de extraer los valores de reflectividad medidos por el radar meteorológico. Los datos están contenidos en archivos KMZ, de los que el algoritmo extrae la información necesaria para crear una matriz con los valores de reflectividad en dBZ. El algoritmo en cuestión funciona convirtiendo los valores codificados como un gradiente de color a la respectiva medida en dBZ extrapolando los valores correspondientes a un conjunto de puntos de referencia disponibles. Se realizaron pruebas con archivos de medidas arbitrarias, obteniendo resultados positivos. El algoritmo se realizó en Python, y el producto obtenido puede ser fácilmente exportado en el formato que sea necesario (e.g. CSV o TSV) para su uso en tareas posteriores de tratamiento de datos del radar.

1. Introducción

Un radar es un dispositivo capaz de emitir pulsos electromagnéticos o sonoros, y medir a continuación las características de la onda reflejada por algún objeto. De esta medida se pueden calcular distancias y características del objeto reflector a partir de la comparación de la intensidad de la onda emitida con aquella de la onda reflejada, así como el tiempo en el que se reflejó. En el caso de un Radar Doppler, se compara también la longitud de onda del pulso reflejado con aquella del emitido, lo que permite medir, mediante el efecto Doppler, la velocidad del objeto reflector.

Un radar meteorológico es un tipo de radar diseñado y utilizado específicamente para detectar y medir fenómenos atmosféricos, como tormentas. Las características de los objetos medidos permiten a los meteorólogos estudiar el comportamiento de los fenómenos en consideración. Por ejemplo, mediante la comparación de los niveles de reflectividad de diferentes objetos se pueden identificar las lluvias ligeras de las tormentas intensas y con granizo, así como las características específicas de la nube o tormenta en cuestión (tamaño, velocidad radial, evolución, etc.)

En este trabajo nos concentraremos en las medidas de reflectividad del radar, las obtenidas mediante la comparación de la intensidad de las ondas reflejadas y las ondas emitidas. Dichas medidas son cuantificadas utilizando una variable llamada *factor de reflectividad* (generalmente denotada con la letra Z). Dado que esta variable varía por varios órdenes de magnitud de una medida a la siguiente, se utilizan decibeles de esta cantidad, dBZ, para expresar las medidas del radar, ver ecuación (1).

El radar arroja los resultados de las mediciones realizadas en un formato de archivo llamado KMZ (ver sección 2), que permite visualizar y explorar la reflectividad en el mapa. A partir de esta representación

gráfica, un meteorólogo capacitado en la interpretación de medidas del radar es capaz de obtener conclusiones acerca de la naturaleza y comportamiento de los objetos observados; así como de hacer predicciones sobre su comportamiento futuro a corto plazo.

De cualquier modo, puede ser de interés obtener las medidas de reflectividad del radar en un formato tabular, no para el proceso de interpretación recién descrito, pero sí para realizar alguna tarea de tratamiento de datos. Este es el objetivo del proyecto, específicamente:

> Programar un algoritmo para extraer la información de reflectividad de los archivos KMZ arrojados por el radar meteorológico del Instituto de astronomía y meteorología de la Universidad de Guadalajara.

Dicha información se busca extraer en un formato matricial/tabular, como una matriz de tres columnas en la que la primera corresponda al valor de reflectividad medido en cierto punto, y la segunda y tercera a la longitud y latitud de dicho punto.

2. Investigación

2.1. Archivos KMZ y KML

Dentro del contexto de este proyecto, las siglas KML corresponden a la abreviación para "*Keyhole Markup Language*"[1], el cual es un formato de archivo utilizado para desplegar datos geográficos en los programas correspondientes para su lectura, como lo son *GoogleEarth*, *NASA WorldWind*, entre otras. Este formato fue desarrollado originalmente para su uso con Google Earth, por *Keyhole, Inc.*

Estos archivos poseen una estructura basada en etiquetas con elementos y atributos *anidados*, es decir, aquellos que se escriben dentro del *ambiente* de otros elementos[2]; creando una estructura *ramificada*, de manera similar a como se observa en los archivos TEX. Los datos en el documento se escriben en *ambientes* que siempre inician con una etiqueta de la forma "`<Tag>`" (e.g. `<Document>`, `<Placemark>`, `<coordinates>`) y cierran con la correspondiente etiqueta de la forma "`</Tag>`" (e.g. `</Document>`, `</Placemark>`, `</coordinates>`).

Las reglas de codificación de los archivos KML siguen el lenguaje XML, los cuales se encuentran organizados de tal manera que puedan ser leídos tanto por la computadora como por el humano. Esto se hace evidente si se mira el sistema de etiquetado mencionado anteriormente, el cual hace la codificación de los archivos lo suficientemente versátil para ser leída por programas creados en distintos lenguajes de programación.

Por otro lado, un archivo KMZ es un conjunto de archivos comprimidos en formato ZIP, entre los cuales siempre se incluye un archivo KML junto con cualquier otro archivo adicional (imágenes, sonidos, animaciones, etc.). En general un archivo KMZ consistirá meramente de un archivo KML comprimido sin archivos adicionales. Para leer este tipo de archivos es entonces necesario descomprimirlos primeramente para obtener el KML. El archivo KML comprimido en un archivo KMZ es por defecto llamado `doc.kml` independientemente del nombre original del archivo KMZ. Lectores como Google Earth soportan archivos KMZ sin necesidad de ser descomprimidos previamente; de la misma manera son capaces de generar archivos KMZ sin necesidad de realizar la compresión con software adicional.

2.2. Archivos KMZ arrojados por el radar

Obtenidos los archivos KMZ arrojados por el radar de nuestro interés, que nos fueron proporcionados por el Instituto de Astronomía y Meteorología de la Universidad de Guadalajara, se procedió a descomprimirlos y explorar el formato en el que se presenta la información recopilada por el radar, se observó lo siguiente:

Cada archivo KMZ que arroja el radar contiene un archivo KML, junto con una imagen PNG. La información de reflectividad está "coloreada" en la imagen PNG (figura 1); al ejecutarse en un programa como Google Earth, la función del archivo KML es ordenar al programa que sobreponga la imagen PNG a cierta escala en cierta sección del mapa (estando la escala y posición declaradas en el archivo KML).

En el archivo KML la posición de la imagen se declara mediante cuatro límites coordenados, que representan los límites norte, sur, este y oeste en los que la imagen se deberá sobreponer en el mapa. El siguiente es un ejemplo de archivo KML arrojado por el radar

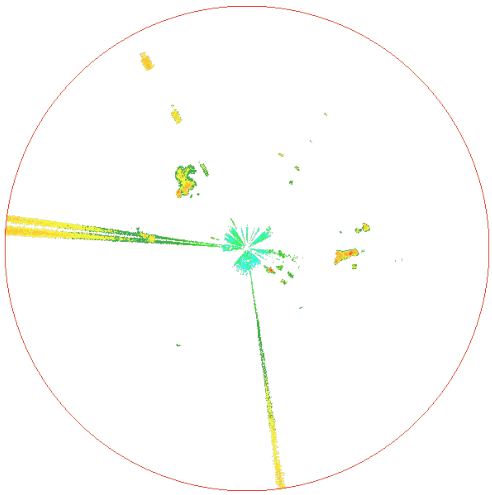


Figura 1: Ejemplo de imagen PNG comprimida en un archivo KMZ arrojado por el radar.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2"
xmlns:kml="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom">
<Folder>
        <name>MEX-Filtered Intensity(Horizontal)-Wed Oct 2 02:38:39 2019</name>
        <ExtendedData>
                <edge:Moment>Filtered Intensity(Horizontal)</edge:Moment>
                <edge:MomentAbbreviation>ZH</edge:MomentAbbreviation>
        </ExtendedData>
        <GroundOverlay>
                <Icon>
                        <href>files/MEX_ZH_1569983919_1569983989.png</href>
                </Icon>
                <LatLonBox>
                        <north>22.03030437021881</north>
                        <south>19.32059531316582</south>
                        <east>-101.9462411978663</east>
                        <west>-104.8254262826025</west>
                </LatLonBox>
        </GroundOverlay>
        <lookAt>
   <longitude>-103.3858337402344</longitude>
   <latitude>20.67555618286133</latitude>
  </lookAt>
</Folder>
</kml>
```

Los mencionados límites coordenados están especificados en la sección `<LatLonBox>`, a su vez la sección `<GroundOverLay>` (en la que está contenida la sección previamente mencionada) es en donde se declara la acción de pegar la imagen sobre el mapa en los límites descritos.

3. Estructura del Algoritmo

De la sección anterior resulta evidente que de la imagen PNG deberá extraerse la información de reflectividad, mientras que el archivo KML la referencia al sistema coordenado, a partir de la cual se asignará un

punto coordenado para cada valor de reflectividad. Dicho esto, proponemos que el algoritmo siga el proceso siguiente:

1. Descomprimir el archivo KMZ, obteniendo el archivo KML y la imagen PNG.
2. Extraer la matriz de colores RGBA de la imagen PNG.
3. Extraer la escala de coordenadas del archivo KML.
4. Convertir los valores de color contenidos en la matriz de valores RGBA a reflectividad (dBZ).
5. Escalar la matriz de reflectividad, asignando un punto coordenado a cada entrada de dicha matriz.

A continuación se describen a detalle cada uno de estos pasos. En el apéndice se adjunta el algoritmo completo; de modo que pueda ser consultado conforme se hace referencia a éste en el texto siguiente.

3.1. Descompresión del archivo KMZ

Como se mencionó en la sección anterior, la parte inicial del algoritmo de análisis se implementa para extraer los elementos dentro del archivo KMZ, en efecto los elementos complementarios (que en nuestro caso es solo la imagen PNG que arrojó el radar meteorológico) y el archivo KML. Sabemos de primera instancia que el archivo KMZ está empaquetado, originalmente, con formato ZIP formando un fichero. La idea central es entonces regresar a la extensión `.zip` y al ser esta además un formato de compresión es posible extraer su contenido en una dirección de preferencia.

Para llevar a cabo esta idea primeramente se implementó el módulo *os* (línea 1), este provee una manera versátil de usar funcionalidades dependientes del sistema operativo y el módulo *ZipFile* (línea 4), el cual proporciona herramientas para crear, leer, escribir, agregar y listar un archivo ZIP. A esto le sigue definir la dirección de trabajo y por ende el lugar en el que se encuentra el script del algoritmo y el archivo KMZ; esto se logra mediante la variable `dir_actual` (línea 12).

A continuación se define una función (la cual comienza en la línea 14 y termina en la línea 39) que se encarga de: cambiar la extensión del archivo de `.kmz` a `.zip` (líneas 16 a 19); leer el contenido y reconocerlo (líneas 22 a 25); definir la dirección de descompresión y proceder a descomprimirlo (líneas 27 a 29); finalmente renombrar la imagen para poder manipularla con facilidad y regresa el archivo a su extensión original (líneas 31 a 39). En síntesis la función definida como `descomprimir()` extrae, en la dirección actual de trabajo, el archivo KML y la imagen PNG, y a esta última le asigna un nombre más accesible para futuras referencias.

3.2. Extracción de matriz de imagen PNG y coordenadas

3.2.1. Matriz de colores

Para poder explicar esta sección del código, es necesario analizar cómo funcionan los módulos de las librerías que fueron incluidos desde un inicio. Primeramente, se hace uso de la librería *NumPy* con su módulo *array*, el cual da soporte y nos permite trabajar con matrices relativamente grandes y multidimensionales. También se utiliza el módulo *Image* de la librería Pillow, la cual agrega la capacidad de procesado de imágenes.

Estas dos librerías serán de ayuda para extraer la imagen de su directorio de origen, y para definir las dimensiones de la matriz obtenida. La línea 46 del código se encarga de buscar el archivo en su destino y de abrirlo, mientras que la línea 47 se encarga del tema de la matriz.

La matriz obtenida es de $m \times n \times 4$, siendo m y n el ancho y largo de la imagen original (en píxeles), mientras que el 4 corresponde a cada uno de los canales rojo, azul, verde y alfa. Los primeros tres representando la cantidad de color presente y el cuarto representa la opacidad del píxel.

3.2.2. Extracción de coordenadas del archivo KML

Una vez obtenida la matriz de color correspondiente a la imagen PNG obtenida por el radar, el algoritmo se encarga de analizar el archivo KML para extraer la información relevante de este. Para el propósito del

presente trabajo esta información no es más que el conjunto de coordenadas terrestres en las que se encuentra ubicado el radar meteorológico.

Para implementar esto se utilizó la librería *pykml*, en especifico el módulo *parser* (línea 6), este módulo, tal y como lo indica su traducción, se utiliza principalmente para analizar archivos KML.

Debe notares que aunque este módulo es realmente sencillo y su ejecución no va más allá de 3 líneas de código, los archivos KML que arroja el radar poseen un elemento que es ajeno a la estructura original de un archivo KML; el elemento «**edge:**» no es reconocido por el módulo *parser*. Aunque desconocemos que fines prácticos posee este elemento no es necesario conocerlos ya que solo buscamos extraer del archivo KML las coordenadas geográficas del radar meteorológico. Por lo anterior se implementó un apartado que se encarga de eliminar el elemento en conflicto (líneas 51 a 66) y así obtener un archivo en condiciones de ser analizado por el modulo, en efecto en las líneas 68 y 69 se define la parte del algoritmo que analiza el archivo KML. Finalmente se le asigna una variable a cada coordenada para ser utilizadas posteriormente en la correspondencia entre reflectividad y coordenadas geográficas (líneas 71 a 74).

Para obtener una presentación cuidada al ejecutar el algoritmo se decidió añadir una sección que se encarga de eliminar todo posible archivo, carpeta o elemento que pudiese quedar como residuo al ejecutar la función `descomprimir()`.

3.3. Conversión a dBZ y asignación de coordenadas

La sección de código descrita hasta este punto extrae toda la información relevante de los archivos KMZ, esto es, la información de color de la imagen y los límites coordenados de dicha imagen. A continuación queda convertir los valores de color a dBZ, y asignar el punto coordenado correspondiente a cada medida; estos dos procesos se describen en la presente sección.

3.3.1. Conversión de valores de color a valores de reflectividad

De la sección 3.2.1 se obtuvo un arreglo matricial de dimensiones $586 \times 600 \times 4$, nombrado en el código como `Ar_im` (línea 47). Tal como se describió en la mencionada sección, el arreglo consiste en cuatro matrices concatenadas, cada una de tamaño 586×600, que describen el color en cada píxel de la imagen PNG. Dicho color se describe utilizando cuatro valores numéricos (a los que denominamos *canales*), el primero correspondiente a la cantidad de color rojo, el segundo al verde, el tercero al azul y, finalmente, el cuarto corresponde al nivel de opacidad de la imagen, siendo el color perfectamente visible cuando este valor es máximo y transparente cuando este valor es mínimo. Cada una de las matrices mencionadas contiene los valores de uno de los 4 canales para cada píxel de la imagen (evidentemente el tamaño de la imagen es 586×600).

Este modelo de color es generalmente llamado $RGBA$ ("Red, Green, Blue & Alpha") y cada uno de los canales toma valores enteros entre 0 y 255 (pues el número total de valores, 256, es la cantidad de configuraciones que un byte, i.e. 8 bits, puede codificar).

Buscamos entonces una función que tome una tupla (R, G, B, A), correspondiente a un color descrito por los cuatro canales $RGBA$, y asigne un valor en dBZ correspondiente a dicho color. Para plantear la función es necesario explorar el comportamiento de los canales en un archivo KMZ.

Iniciando por el canal Alfa, observamos que este toma solamente tres valores: un valor nulo (0) en todos los puntos transparentes, i.e. todos los puntos en los que no se midió reflectividad; el valor máximo (255) en todos los puntos del aro rojo que delimita la zona de medición (ver figura 1); y un valor fijo (206) en todos los puntos en los que se midió reflectividad.

Antes de continuar notemos que la reflectividad se cuantifica con el factor de reflectividad Z, pero debido a que este factor puede variar por varios órdenes de magnitud entre medida y medida, se utilizan decibeles de este factor, dBZ, al exportar las medidas del radar. Por definición tenemos que

$$\mathrm{dB}Z = 10 \log_{10} Z, \tag{1}$$

en donde hemos ignorado las unidades de Z (mm^6/m^3). De esta definición es fácil notar que, cuando $Z = 0$, dB$Z \to -\infty$, por tanto, es imposible asignar un valor de dBZ en todos los puntos con reflectividad nula. En el archivo arrojado por el radar este problema se resuelve fijando un valor nulo del canal Alfa en los puntos con reflectividad nula, de modo que la imagen es transparente en dichos puntos. Dado que nuestro programa

arrojará un valor numérico, es imposible utilizar esta misma estrategia; por tanto, proponemos que nuestro código ignore las entradas con reflectividad nula e incluya en la matriz producto solo las entradas en las que Z es diferente de cero. Esta estrategia además optimiza enormemente el código, pues generalmente el 93-97 % de la imagen es transparente.

Para realizar dicho proceso volvemos a la observación de que el canal Alfa vale 0 en todas las entradas con reflectividad nula, por lo que solamente es necesario construir una nueva matriz (en el código llamada `Ar`) que incluya únicamente las entradas en las que $A \neq 0$.

Antes de construir dicha matriz se vuelve necesario eliminar el aro rojo de la imagen, pues en el caso contrario el algoritmo arrojaría un valor de reflectividad para cada punto del aro. Para esto utilizamos la observación de que Alfa vale 255 en todos los puntos del aro y solo en estos. Basta entonces con crear dos ciclos *for* que verifican el valor de Alfa en cada punto (i, j) de la imagen ($i = 0, \ldots, 585$ y $j = 0, \ldots, 599$), este proceso corresponde a las líneas 87-91 del código; si dicho valor es igual a 255, las coordenadas (i, j) del punto en cuestión se almacenan un un arreglo llamado `Aro` (creado en la línea 85), cuya primera columna es el valor de i y la segunda el valor de j. Una vez realizado este proceso se utiliza otro ciclo *for* (líneas 94 y 95) para cambiar el valor de Alfa en todos los puntos contenidos en `Aro`, se asigna un valor nulo de modo que la siguiente parte del código los desprecie tal como a las entradas con $Z = 0$. (La línea 92 simplemente borra el primer renglón del arreglo `Aro` que, por la forma en la que se creó, siempre será $[0, 0]$).

A continuación realizamos el proceso de discriminar las entradas nulas. En la línea 98 se define el arreglo `Ar` que tiene 5 columnas, en las primeras tres se almacenarán los valores R, G y B de cada punto (i, j) en el que la reflectividad no sea nula, y en las últimas dos se incluirán los índices i y j que especifican la ubicación de este punto en la imagen. Al igual que antes, este proceso se realiza con dos ciclos *for*, uno para el índice vertical y otro para el horizontal, escritos en las líneas 100-104. Utilizando la función *if* se determina si el valor de Alfa es no nulo y, si es así, se introducen los datos especificados en un nuevo renglón de `Ar`; esto para cada punto en los ciclos *for*, i.e. para cada punto de la imagen. La línea 105, como antes, elimina el primer renglón de `Ar` que será nulo por defecto.

La razón de que no se incluyera el valor de Alfa en `Ar` es que, tal como se observó anteriormente, este canal tiene un valor fijo para todos los puntos en los que se midió reflectividad, por lo que no aporta información extra a lo que ya se consideró.

Tornamos ahora nuestra atención a buscar una función que tome una tupla (R, G, B) y arroje un valor en dBZ (ya eliminamos la posibilidad de que este último valor tienda a $-\infty$, por lo que estará bien definido). Para ello es necesario contar con una referencia del protocolo que el radar utiliza para asignar colores a valores de reflectividad. Dicha referencia se presenta en la figura 2. La figura contiene 17 colores con su respectivo valor de reflectividad, un vistazo rápido a `Ar_im` para cualquier archivo KMZ confirma que los colores en la imagen PNG toman más de 17 valores, por lo que la tarea a mano es extrapolar los valores conocidos al resto del espacio RGB. Iniciamos por definir este último término:

Notamos que las tuplas de la forma (R, G, B), que especifican un color para cierto valor fijo del canal Alfa, forman un subconjunto de $\mathbb{R}^3$; a dicho conjunto se le llama **espacio RGB** y cada punto en tal conjunto corresponde a un color.

Utilizando un código análogo al de las líneas 46 y 47 se extrajeron los valores RGB de la imagen de referencia (figura 2). En la tabla 1 se presentan dichos valores, las tres primeras columnas corresponden a la tupla (R, G, B) en cuestión, mientras que la última contiene el valor de reflectividad asignado a esta tupla.

La tabla 1 se introdujo en el código como un arreglo matricial, `M`, en las líneas 107-110. Observando la tabla en cuestión se vuelve evidente que el canal rojo solamente toma un valor máximo o un valor mínimo, sin ocupar puntos intermedios; mientras que los canales verde y azul varían con mayor *suavidad*.

Explorando, como antes, el comportamiento del canal rojo para diferentes archivos KMZ obtenemos que, al igual que en los puntos de referencia, el canal rojo solo toma un valor máximo o mínimo. En este caso los valores máximo y mínimo no son 255 y 0, sino 246 y 17.

Este comportamiento del canal rojo nos motiva a considerar los subconjuntos $\{(R, G, B) : R = 255\}$ y $\{(R, G, B) : R = 0\}$ del espacio RGB. En dichos subespacios es posible realizar un gráfico de los puntos de referencia disponibles (que son 11 para el primer caso y 6 para el segundo). Las gráficas se presentan en la figura 3.

En base a las observaciones anteriores proponemos que le primer paso de la función para convertir tuplas (R, G, B) a valores de reflectividad (que en lo sucesivo llamaremos f) sea

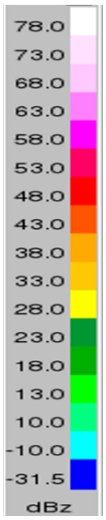


Figura 2: Escala de colores.

(a) Para una tupla dada (R, G, B). Determinar si el valor de R es máximo (R=246) o mínimo (R=17) y proyectar la tupla al subespacio correspondiente ($R = 255$ o $R = 0$, respectivamente).

Realizada esta acción ya se habrá utilizado toda la información disponible en el canal R, por lo que a continuación se trabajará con la tupla (G, B) en el subespacio que se haya determinado. Notemos que los puntos de referencia mostrados en la figura 3 forman un "camino" en el subespacio en consideración, dicho camino se puede ver siguiendo los puntos ya sea en orden creciente o decreciente de reflectividad. Considerando el orden decreciente, en el caso $R = 255$ dicho camino inicia en $(G, B) = (255, 255)$, continúa hasta el punto $(G, B) = (0, 255)$, a continuación sigue hasta $(G, B) = (0, 0)$ y termina en $(G, B) = (255, 0)$; conteniendo un total de 11 puntos de referencia. Para el caso $R = 0$ el camino inicia en $(G, B) = (150, 50)$, etiquetado con el valor 23 de reflectividad; el siguiente punto de referencia hace el valor de B nulo y, a continuación, llega a $(G, B) = (255, 0)$; a partir de este punto el camino sigue a $(G, B) = (255, 255)$ y termina en $(G, B) = (0, 255)$. Este último caso cuenta un total de 6 puntos de referencia y es menos sencillo que el caso anterior.

Sabemos ya que no todos los valores arrojados por el radar coinciden con uno de los puntos de referencia. Aun así se esperaría que, en un caso ideal, dichos valores se encontrasen en algún punto del camino definido por los puntos de referencia. Dedicándonos de nuevo a la tarea de explorar los valores arrojados por el radar en busca de la mencionada relación observamos que no se satisface este requerimiento, pero todos los valores arrojados por el radar se encuentran cerca del camino. Con *cerca* nos referimos a que, si bien cierto valor no se encuentra en el camino, sí se encuentra suficientemente cerca de una sección de dicho camino como para que no exista ambigüedad al respecto de qué punto del camino corresponde al valor en cuestión.

Antes de asignar un valor de reflectividad a un punto arbitrario, consideremos la cuestión de asignar un valor de reflectividad a un punto $p := (G, B)$ que se encuentra en el camino definido por los puntos de referencia, ya sea en $R = 0$ o $R = 255$. Este punto se encontrará entre dos de los puntos de referencia, a los que llamaremos p_a y p_b; si denotamos con r_a y r_b a los valores de reflectividad en cada uno de estos puntos, con l_{ab} a la distancia que separa p_a de p_b, con l_a a la distancia entre p y p_a, y con l_b a la distancia entre p y p_b. La hipótesis de que p se encuentre en la línea $\overline{p_a p_b}$ es equivalente a

$$l_{ab} = l_a + l_b. \tag{2}$$

Si $p = p_a$ ($l_a = 0$), asignaríamos a p el valor de reflectividad r_a; en el caso $p = p_b$ ($l_b = 0$) asignaríamos r_b. Si p estuviera, por ejemplo, a la mitad entre p_a y p_b ($l_a = l_b$), se le asignaría un valor medio entre r_a y r_b. Sea r entonces el valor de reflectividad asignado a p, la función más simple y apropiada que se le puede asignar es una función lineal:

$$r = r_a + \left(\frac{r_b - r_a}{l_{ab}}\right) l_a = r_b - \left(\frac{r_b - r_a}{l_{ab}}\right) l_b, \tag{3}$$

R	G	B	dBZ
255	255	255	78.0
255	225	255	73.0
255	200	255	68.0
255	128	255	63.0
255	0	255	58.0
255	0	100	53.0
255	0	0	48.0
255	85	0	43.0
255	170	0	38.0
255	200	0	33.0
255	255	0	28.0
0	150	50	23.0
0	175	0	18.0
0	255	0	13.0
0	255	128	10.0
0	255	255	-10.0
0	0	255	-31.5

Tabla 1: Valores RGB y de reflectividad en los puntos de referencia.

en donde hemos asumido, sin pérdida de generalidad, que $r_b > r_a$. Notemos que las dos expresiones dadas son equivalentes por la hipótesis de la eq. (2).

Si el punto p no estuviera en la línea que une p_a y p_b, pero estuviera suficientemente cerca de esta línea como para usarla como referencia para asignar la refelctividad de p, la ecuación (2) no sería válida y, bajo el razonamiento anterior, tendríamos dos valores posibles de reflectividad, uno medido con respecto a p_a y otro con respecto a p_b, dados respectivamente por

$$r_{p_a} = r_a + \left(\frac{r_b - r_a}{l_{ab}}\right) l_a \quad \text{y} \quad r_{p_b} = r_b - \left(\frac{r_b - r_a}{l_{ab}}\right) l_b. \tag{4}$$

Considerando la condición de que p esté cercano a la línea, ambos valores, r_{p_a} y r_{p_b}, serán una buena aproximación para el valor de reflectividad de p de acuerdo a la escala disponible. Tomamos entonces el valor de reflectividad de p como el promedio de estos dos valores

$$r = \frac{r_{p_a} + r_{p_b}}{2}. \tag{5}$$

Si p se encontrara en la línea $\overline{p_a p_b}$, este valor y el asignado en (2) coincidirían. Por otro lado, el error posible de este valor crece conforme p se aleja de la línea, por lo que la hipótesis propuesta (que p esté cercano a la línea) es vital.

Queda pendiente la cuestión de determinar, para un $p = (G, B)$ dado, cuáles son los puntos de referencia p_a y p_b para los cuales p se encuentra más cerca de la línea $\overline{p_a p_b}$. Para esto basta con determinar cuál es el punto de referencia más cercano a p y cuál es el segundo más cercano. Si p está suficientemente cerca del camino formado por todos los puntos de referencia, entonces el punto más cercano a p y el segundo más cercano serán contiguos, de modo que estos serán p_a y p_b.

Al igual que antes, si p se encontrara muy alejado del camino, es posible que el punto más cercano y el segundo más cercano no sean contiguos, en este caso se vuelve imposible asignar un valor de reflectividad a p mediante el método propuesto. Este detalle es, en general, una ventaja; pues debido a que el código deja de funcionar si p estuviera demasiado alejado del camino formado por los puntos de referencia, tenemos garantía de que p en efecto será cercano a dicho camino. (Tal como se mencionará más adelante, hasta ahora el código ha funcionado correctamente, por lo que no se "rompe" esta hipótesis).

En base a la discusión previa proponemos los siguientes pasos a seguir para f:

(b) Identificar, dentro del subespacio determinado en el paso anterior ($R = 0$ o $R = 255$), el punto de referencia más cercano a (G, B) y el segundo punto de referencia más cercano.

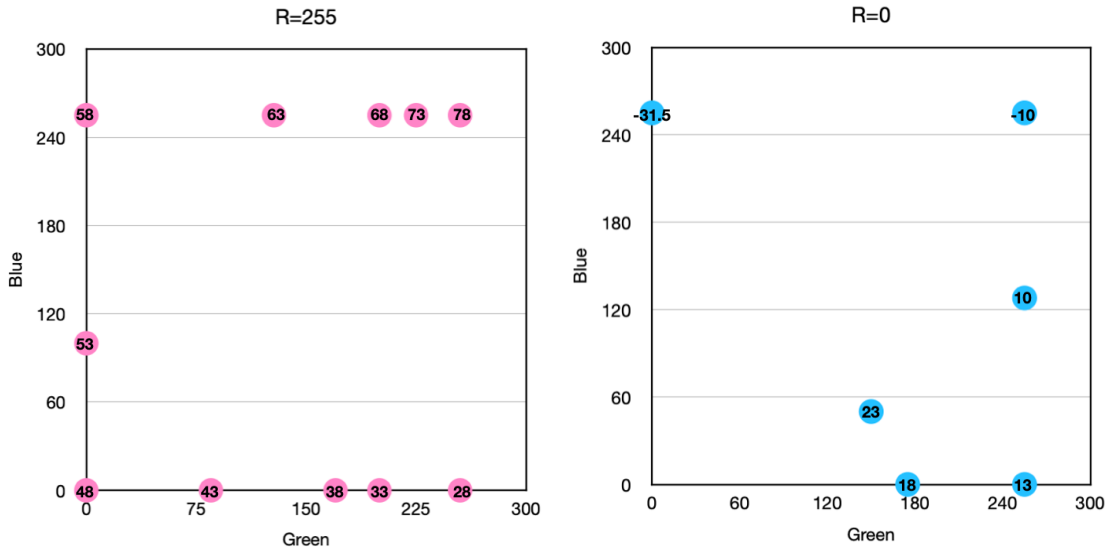


Figura 3: Posición de los puntos de referencia en los subespacios $R = 255$ y $R = 0$.

(c) Si los puntos de referencia son contiguos, calcular la distancia de (G, B) a cada uno de ellos y calcular los valores r_{p_a} y r_{p_b} de la ecuación (4) tomando a uno de los puntos como p_a y al otro como p_b, tal que $r_b > r_a$.

(d) Asignar a (R, G, B) el valor de reflectividad correspondiente al promedio (5) para los valores calculados en el paso anterior.

Podemos notar que los pasos (b) y (c) estarán "mezclados" en el código, pues para realizar (b) es necesario calcular la distancia de (G, B) a todos los puntos de referencia, entre ellos los puntos de interés; así en (c) solamente es necesario tomar las distancias ya calculadas y hacer el cálculo de r_{p_a} y r_{p_b}.

A continuación describimos la implementación en el código de la función descrita. Ya se mencionó que los datos de la tabla 1 se introdujeron al código en las líneas 107-110 en un arreglo matricial denotado como `M`. La distancia entre puntos contiguos de referencia, l_{ab} $(b = a - 1 = 0, \ldots, 15)$, se calculó e introdujo en un vector de nombre `Md` en las líneas 112-115. Dado que son 17 puntos de referencia, se calcularon en total 16 de esas distancias; notemos además que la distancia entre los puntos de referencia 10 y 11[1] no es de relevancia alguna, pues el punto 10 es el último en el subespacio $R = 255$ y el 11 es el primero en el subespacio $R = 0$.

Del mismo modo que antes, se calculó la variación en valores de reflectividad entre puntos de referencia, $r_b - r_a$ $(b = a - 1 = 0, \ldots, 15)$, en las líneas 117-120. Estos valores se introdujeron en un arreglo de nombre `Mv`.

Como un último paso antes de definir f, se realizó el cálculo de los factores lineales de la eq. (4) en las líneas 122-125. Los valores obtenidos se introdujeron en un arreglo de nombre `Ml`. Este cálculo se realizó utilizando los valores ya obtenidos de `Md` y `Mv`.

Estos tres pasos se realizaron de forma previa con el propósito de optimizar la función, pues dicha función será evaluada de 5000 a 30000 veces, dependiendo del archivo KMZ. Por tanto es más apropiado calcular los 15 factores lineales relevantes previamente y "llamarlos" cada vez que se evalúa la función, que hacer el cálculo cada vez que se evalúa la función.

En la línea 127 inicia la definición de la función, en este caso llamamos a la tupla de entrada (r, g, b), a diferencia del texto en el que se utilizaron mayúsculas. En el primer paso se utiliza la función *if* para determinar si el canal rojo toma un valor mínimo (menor a 30) o máximo (mayor a 225); línea 128 para el primer caso y 152 para el segundo. Evidentemente se está considerando una tolerancia de 30 puntos, esto motivado por el hecho de que en los archivos KMZ el valor rojo toma valores máximos/mínimos sin llegar al límite.

[1] En este caso, y en el resto del texto, se toman los índices iniciando desde el cero, como sucede en el programa. Por lo que la entrada 10 será la decimoprimera entrada, mientras que la entrada 11 será la decimosegunda.

El siguiente paso de la función (en cualquiera de los dos casos) es calcular la distancia del punto (g, b) a cada uno de los puntos de referencia en el subespacio que se haya determinado; esto se realiza en las líneas 129-131 para el caso r mínimo, y en las líneas 153-155 para el caso r máximo. En ambos casos los valores obtenidos se introducen en un vector `A` que tiene 6 u 11 entradas, respectivamente. A continuación (líneas 132-136 en el primero caso y 157-161 en el segundo) se determina el valor mínimo del arreglo `A`, se etiqueta como `xmin` y se determina el índice que corresponde a dicho valor, al que se le asigna el nombre `k`. Hecho esto se elimina el valor `k`-ésimo del arreglo `A` y se vuelve a calcular el valor mínimo del arreglo obtenido junto con su índice, que se denotan respectivamente como `xmin2` y `k2`. El punto de referencia `k`-ésimo será entonces el punto más cercano a (g, b), mientras que el punto de referencia `k2`-ésimo será el segundo más cercano. Los respectivos valores `xmin` y `xmin2` serán las distancias l_a y l_b a estos puntos.

Notemos que es posible tener `xmin`$= l_a$ y `xmin2`$= l_b$, o `xmin`$= l_b$ y `xmin2`$= l_a$; dependiendo si `k2` sigue a `k` o viceversa. En el caso en el que el punto más cercano sea k y el segundo más cercano sea $k+1$ tendremos `k2`=`k`, pues al eliminar la entrada `k`-ésima del arreglo `A`, `k2` toma el mismo valor numérico que `k`. En el caso contrario, cuando k es el punto más cercano y $k-1$ es el segundo, el hecho de eliminar la entrada `k`-ésima no afecta a la entrada `k2`-ésima y tenemos `k2`=`k`-1. Cuando el punto más cercano es el primero o el último punto de referencia del subespacio en consideración solo existe una posibilidad, por lo que no es necesario hacer las consideraciones anteriores.

En base a la discusión anterior consideramos tres casos: (i) `k`$= 0$, (ii) `k`$= 5$ o `k`$= 10$ (5 en el caso r mín., 10 en el caso r máx.) y (iii) `k`$= 1, \ldots, 4$ o `k`$= 1, \ldots, 9$ (al igual que antes, las dos opciones corresponden a los casos r mín. y máx.)

Consideremos primero cada uno de los casos cuando r es mínimo. Para (i) tenemos que el punto de referencia más cercano es el punto 11 de `M`, i.e., el punto con valor de reflectividad 23 de la figura 3. Por lo que el siguiente punto de referencia más cercano será el 12 (valor de reflectividad 18 en la figura 3) y el factor de crecimiento lineal será la entrada 11 de `Ml`. En las líneas 138-139 del código se escribe este caso, se utiliza `Z` para el valor de reflectividad asignado (a diferencia del texto, en el que se usó r). `M[11,3]` es el valor de reflectividad de la entrada 11 de `M` (en este caso 23), y `Ml[11]` es el valor 11 de `Ml`, este valor multiplica a `xmin` pues, como ya vimos, esta es la distancia de (g, b) al punto en cuestión. Finalmente, el signo menos se debe a que de las entradas 11 a 12 de `M` el valor de reflectividad decrece.

Para (ii) nos encontramos con que el punto más cercano es el punto 16 de `M` (valor de reflectividad -31.5 en la figura 3), por tanto el segundo punto más cercano será 15, y el factor de crecimiento lineal será `Ml[15]`. En las líneas 140-141 se presenta este caso, en donde ahora se utiliza un signo positivo pues del punto 16 al punto 15 la reflectividad crece.

Finalmente, para el caso (iii) se consideran dos "sub-casos": (1) si `k2`=`k` tenemos que el punto más cercano será la entrada `k+11` de `M` (notemos que estamos en el subespacio $R = 0$, por lo que los puntos de referencia inician desde el índice 11), mientras que el segundo más cercano será `(k+11)+1`; por lo que el factor lineal a usar es `Ml[11+k]`. En la línea 144 se calcula el valor de reflectividad `Z1` con respecto al punto `k` (usando `xmin`), mientras que en la línea 145 se calcula el valor `Z2` con respecto al punto `k+1` (usando `xmin2`). En la línea 146, finalmente, se calcula el promedio `Z`. (2) En este caso `k2`=`k`-1 y el punto más cercano será `k+11`, mientras que el segundo será `(k+11)-1`; el factor de crecimiento lineal será `Ml[11+(k-1)]`=`Ml[10+k]` y el cálculo análogo al caso anterior se realiza en las líneas 147-150.

Cuando r es máximo se realiza un proceso análogo al anterior en las líneas 163-175. Con la diferencia de que en este caso los índices no están desplazados, i.e., comienzan a contar desde el 0. Finalmente, la línea 176 ordena al código retornar el valor de `Z`, con el que etiquetamos a la reflectividad obtenida en cada uno de los casos, cada vez que se evalúe f.

Hecho esto, ya está definida la función que toma una tupla `(r,g,b)` y arroja un valor de reflectividad `Z`. Tomamos entonces el arreglo `Ar` en el que, como ya se mencionó, se incluyen las tuplas RGB junto con su ubicación (i, j) en la imagen; y evaluamos la función en cada una de las tuplas contenidas en este arreglo. El resultado se guarda en una nueva matriz `Ar_Z`, que esta vez contiene el valor de reflectividad en la primera columna, y los índices (i, j) en las dos siguientes. Este proceso se realiza en las líneas 178-182.

Lo único que falta a continuación es asignar una latitud y longitud a cada punto (i, j) de la imagen. Este proceso es el tópico de la siguiente sección.

3.3.2. Asignación de coordenadas

Recordemos que la imagen PNG es de dimensiones 586×600, esto significa que, vista como un arreglo matricial, tiene 586 entradas verticales y 600 horizontales. Parametrizando las primeras con un índice $i = 0, \ldots, 585$ y las segundas con un índice $j = 0, \ldots, 599$; podemos expresar cualquier punto en la imagen como una tupla (i, j). Esto es lo que se realizó al construir la matriz `Ar` y se conserva hasta la matriz `Ar_Z`, producto de la sección anterior, en donde la segunda y tercera columnas contienen los valores de i y j, respectivamente, que especifican el punto en el que se midió el valor de reflectividad contenido en la primera columna. El proceso de asignar coordenadas corresponde entonces a asignar un valor de latitud y longitud para cada tupla (i, j).

Para realizar la tarea en cuestión observamos que el archivo KML especifíca a Google Earth dónde sobreponer la imagen a partir de un conjunto de cuatro coordenadas, dos de latitud y dos de longitud, que marcan los límites que tomará la imagen en el mapa. En la sección 3.2.2 se extrajeron dichas coordenadas del archivo KML y se denotaron con las letras n, s, e, w en el código (líneas 71-74). n representa el límite norte de la imagen, s el límite sur, e el límite este y w el límite oeste. A continuación denotamos a un punto coordenado en el mapa como (x, y), en donde x es la latitud y y la longitud.

Despreciando la curvatura de la tierra en la sección que ocupa la imagen, consideramos al mapa como un plano coordenado. Bajo esta hipótesis obtenemos que cada píxel de la imagen abarca la misma distancia en el mapa, por lo que la imagen divide la sección entre $x = n$ y $x = s$ en 586 *trozos* y la sección entre $y = e$ y $y = w$ en 600. Las longitudes de cada uno de estos intervalos son

$$\delta_v = \frac{n - s}{586} \quad \& \quad \delta_h = \frac{e - w}{600}. \tag{6}$$

En donde hemos usado δ_v para los intervalos verticales y δ_h para los horizontales. Notemos que ambos números son positivos, pues las coordenadas son crecientes de sur a norte y de oeste a este.

Si consideramos entonces el índice vertical $i \in [0, 585] \subseteq \mathbb{N}$, la latitud correspondiente será

$$x = n - \left(i + \frac{1}{2}\right) \delta_v, \tag{7}$$

en donde el signo $-$ se debe a que i crece de norte a sur, mientras que x decrece; y el $\frac{1}{2}$ se ha añadido para asignar la latitud en el medio del píxel (i, j).

Del mismo modo, para el índice horizontal $j \in [0, 599] \subseteq \mathbb{N}$, la longitud correspondiente será

$$y = w + \left(j + \frac{1}{2}\right) \delta_h, \tag{8}$$

en este caso j incrementa de oeste a este, al igual que la longitud y.

Así, las coordenadas en el punto medio del píxel (i, j) serán

$$\left(n - \left(i + \frac{1}{2}\right) \delta_v, w + \left(j + \frac{1}{2}\right) \delta_h\right). \tag{9}$$

(latitud en la primera entrada, longitud en la segunda).

Por tanto, en el código hace falta simplemente incluir una sección que cambie cada entrada (i, j) en las dos segundas columnas de `Ar_Z` por los valores correspondientes de (9).

En la línea 188 del código simplemente de cambia la clase en la que están registradas las coordenadas (en la sección 3.2.2 se definen como "strings", mientras que a continuación las utilizaremos como entradas numéricas). A continuación las lineas 190 y 191 calculan δ_v y δ_h como están en la ecuación 6, y finalmente las líneas 193-197 utilizan un ciclo *for* para realizar el cálculo de la ecuación 9 con cada renglón de `Ar_Z`. Los resultados obtenidos se registran en una matriz llamada `Ar_Rc`, esta matriz contiene en su primera columna valores de reflectividad y en la segunda y tercera las coordenadas (iniciando por la latitud) en las cuales se midió tal valor.

Esta última matriz es el producto buscado del algoritmo, que ya puede ser exportada o utilizada como sea conveniente para tareas posteriores.

3.4. Exportación del producto del algoritmo

Si bien el algoritmo puede modificarse a voluntad para exportar la matriz `Ar_Rc` como sea conveniente, en nuestro caso la exportamos como un archivo CSV (*Comma-Separated Values*). Este formato puede ser importado y leído en la mayoría de lenguajes de programación y software de análisis numérico, tales como Python, MATLAB, Maxima, etc., e incluso puede ser desplegado y editado en programas como Microsoft Excel y Numbers. Esto sin mencionar que es leíble por el usuario cuando se abre como un archivo de texto simple.

Para exportar la matriz utilizamos la paquetería `pandas`. En la línea 203 se define un nuevo arreglo a partir de `Ar_Rc`, cuya clase es diferente. El nuevo objeto es una clase de arreglo 2-dimensional propio de la paquetería `pandas`, mientras que `Ar_Rc` es un arreglo matricial propio de la paquetería `numpy`. A continuación, en la línea 204, se utiliza la mencionada paquetería para exportar el archivo en el formato descrito. Dicho archivo será guardado con el nombre `Reflectividad.csv` en la carpeta en la que se esté trabajando (i.e. la carpeta que contiene al código y al archivo KMZ).

4. Resultados, Discusión y Conclusiones

4.1. Producto del algoritmo para ciertos archivos de prueba

Se ha implementado el algoritmo final a ciertos archivos proporcionados por el Instituto de Astronomía y Meteorología del Departamento de Física de la Universidad de Guadalajara. Así se han obtenido resultados satisfactorios en lo que concierne a la concordancia entre los valores en bruto de la reflectividad arrojados por el radar meteorológico y aquellos reconstruidos a partir de la extracción de datos del archivo KMZ.

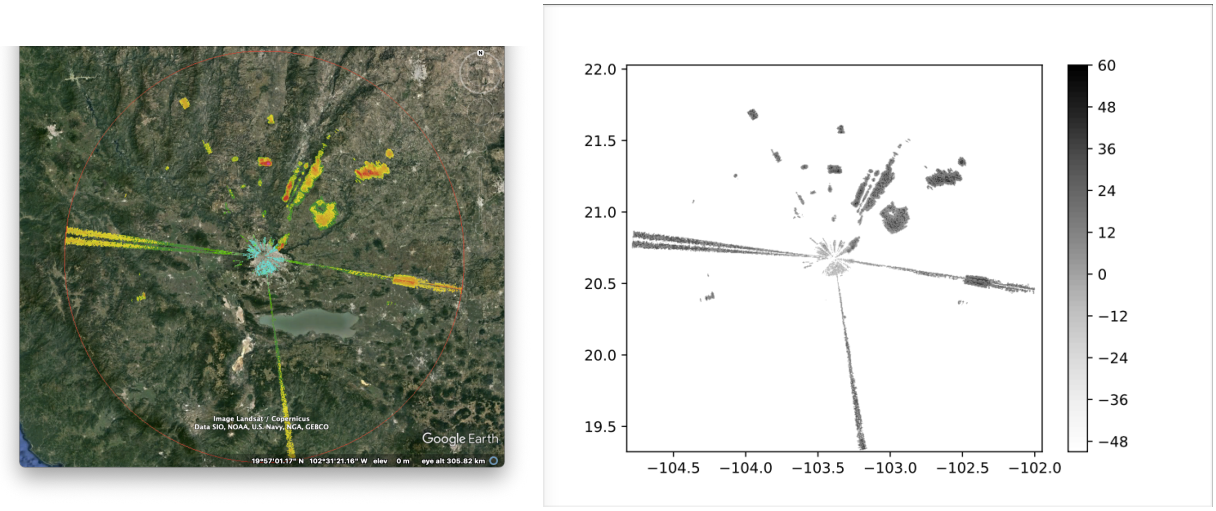
4.2. Gráfico de gradiente de color

La única función de generar un gráfico a partir de la matriz obtenida del algoritmo es permitir la comparación visual del producto del algoritmo y el gráfico original, a modo de evaluación. Esta no es una tarea sencilla, esencialmente porque en el código se despreciaron las entradas nulas, de modo que deberán ser *rellenadas* para poder crear una gráfica. Este proceso no sería un problema de no ser por el hecho ya descrito de que, debido a que la reflectividad está medida en dBZ, cuando la medida es nula ($Z = 0$), dB$Z \to -\infty$. Para sortear este problema se fijó un valor mínimo de -50 dBZ, correspondiente a aproximadamente $Z = 0$ y se rellenaron las entradas nulas con este valor.

El código que realiza dicho proceso se presenta en el apéndice, después del código principal. Dicha sección de código es ejecutada en conjunto con el código principal y funciona como sigue: En las líneas 3-5 se crea un vector de latitud (rellenando las entradas despreciadas en `Ar_Rc`); de la línea 7 a la 9 se crea el vector análogo de longitud; en las líneas 11-18 se crea un arreglo matricial de tamaño 586×600, los puntos de este arreglo que tienen entrada en `Ar_Rc` toman el valor de reflectividad asignado, mientras que al resto de puntos se les asigna el valor de -50 dBZ. Finalmente, las líneas 20-23 generan el gráfico utilizando la librería `matplotlib` y lo exportan como una imagen en formato PDF.

Se utilizó una escala de grises para el gráfico con dos propósitos: (i) que no se saturara la imagen (otra escala podría asignar un color diferente a blanco al valor -50 dBZ, de modo que la mayor parte de la imagen estaría coloreada), y (ii) que no existiera confusión al momento de comparar el gráfico obtenido con la imagen PNG original arrojada por el radar, pues cualquier gradiente de color predeterminado de la librería diferiría del usado por el radar.

Podemos observar que al ejecutar el algoritmo para uno de los archivos de prueba el resultado es sumamente satisfactorio. No solo la reconstrucción es precisa (fig. 4), además al analizar a detalle encontramos que para una muestra grande de puntos, la variación entre el valor de reflectividad arrojado originalmente por el radar y el obtenido mediante el algoritmo es admisible ya que se encuentra dentro de la incertidumbre percibida en la variación de la escala representada en la figura 2, i.e. 5 dBZ (en general puede esperarse que el error sea menor que la mitad de este valor, i.e. 2.5 dBZ, por la forma en la que se hace la asignación lineal de la reflectividad). Debe añadirse a esto que una limitación importante acerca de la incertidumbre es que, debido a que estas son estimaciones hechas a partir de la forma en la que se planteó el algoritmo, no se tiene garantía de que este sea, en efecto, el error posible en los resultados del algoritmo.



(a) (b)

Figura 4: Comparación entre (4a) la reflectividad obtenida por el radar y (4b) la reconstrucción de la reflectividad hecha a partir del algoritmo implementado. Como es natural, el eje vertical representa latitud y el eje horizontal longitud.

A grandes rasgos, a partir de un archivo KMZ, el cual contiene una imagen que representa la reflectividad medida por el radar y una archivo KML con las coordenadas del radar, el algoritmo logra traducir el color de la imagen a valores de reflectividad cuantizables, mediante el uso de una escala y así generar exitosamente una tabla de valores en donde se puede consultar que reflectividad medida cada coordenada geografica.

Resulta interesante notar que el algoritmo desarrollado logra generar datos cuantitativos (la tabla de reflectividad), a partir de datos cualitativos (la imagen que genera el radar) y cómo esto resulta de suma utilidad al momento de buscar una mayor exactitud y rigor en el análisis meteorológico. Además de que, al exportar los datos en un formato amigable como lo es el CSV, es posible utilizar este algoritmo para trasladar la información que en principio es intuitiva pero inexacta a un módulo de análisis que podría arrojar resultados concluyentes de la información obtenida por el radar. Por ejemplo, a partir de estos datos es en principio posible entrenar a una red neuronal en la tarea de identificar fenómenos meteorológicos.

Por otro lado, el algoritmo desarrollado en el presente trabajo podría ser perfeccionado para que sea más eficiente, añadiendo una interfaz de usuario, o permitiendo la posibilidad de analizar más de un solo archivo a la vez para comparar datos entre distintos archivos KMZ. Uno de los aspectos más importantes a considerar para futuras modificaciones, junto con aquel que se refiere a la incertidumbre, es la limitación del alcance del algoritmo, es decir al tipo de archivos KMZ que puede analizar el algoritmo, ya que al tener únicamente disponibles los archivos proporcionados por el radar del Instituto de Astronomía y Meteorología del Departamento de Física de la Universidad de Guadalajara, se desarrolló todo el algoritmo con base en estos. Una futura proyección es entonces generalizar el algoritmo.

Finalmente, el algoritmo presentado podría ser complementado con diferentes formatos de exportación para el producto del algoritmo según se requiera.

Referencias

[1] Keyhole Markup Language. `https://developers.google.com/kml`. Consultado el 18 de abril de 2021.

[2] KML Documentation Introduction. `https://developers.google.com/kml/documentation`. Consultado el 18 de abril de 2021.

Apéndice

A continuación se presenta el código tal como es ejecutado en la consola.

```
import os
import numpy as np
import pandas as pd
from zipfile import ZipFile
from PIL import Image
from pykml import parser
import fileinput
from shutil import rmtree

############### Descomprimir KMZ ###############

dir_actual = os.getcwd() # Directorio original
def descomprimir():
    archivo_kmz = input('ingrese el nombre del archivo KMZ: ')

    nom, ext = os.path.splitext(archivo_kmz)
    os.rename(archivo_kmz, nom + ".zip")

    archivo_zip = nom + ".zip"


    with ZipFile(file = archivo_zip, mode = "r", allowZip64 = True) as file:
        archivo = file.open(name = file.namelist()[0], mode = "r")
        # print(archivo.read())
        archivo.close()

        direccion = "."
        print("Descomprimiendo...")
        file.extractall(path = direccion)

    nom, ext = os.path.splitext(archivo_zip)
    os.rename(archivo_zip, nom + ".kmz")

    os.chdir('/Users/alejandro/Downloads/files') #Ruta de la imagen

    for file in os.listdir():
        nom_ori = file
        nom_nv = 'im_radar' + ".png" # Nuevo nombre de la imagen
        os.rename(nom_ori, nom_nv)

descomprimir()

############### Extraer matriz de imagen PNG ###############

print('Obteniendo matriz RGBA...')
Im = Image.open('/Users/alejandro/Downloads/files/im_radar.png')
Ar_im = np.array(Im)

############### Extraer coordenadas de KML ###############

os.chdir(dir_actual)
print('Obteniendo coordenadas...')

nom, ext = os.path.splitext('doc.kml')
```

```
os.rename('doc.kml', nom + ".txt")
archivo_txt = nom + ".txt"

for line in fileinput.FileInput(archivo_txt, inplace = 1): # Eliminar el Edge
    if line.startswith("                <edge:Moment>"):
        new_line = line.replace(line,"          <Moment>Filtered Intensity(
            Horizontal)</Moment><MomentAbbreviation>ZH</MomentAbbreviation>")
        print(new_line)
    else:
        print(line, end='')

nom, ext = os.path.splitext(archivo_txt)
os.rename(archivo_txt, nom + ".kml")

with open('doc.kml', 'r') as f:
    root = parser.parse(f).getroot()

n = root.Folder.GroundOverlay.LatLonBox.north.text   #Coordenadas de referencia
s = root.Folder.GroundOverlay.LatLonBox.south.text
e = root.Folder.GroundOverlay.LatLonBox.east.text
w = root.Folder.GroundOverlay.LatLonBox.west.text

############## Borrar la evidencia ##############

os.remove("doc.kml")
rmtree("files")

############## Matriz RGBA a dBZ ################

print('Calculando valores de reflectividad...')

Aro = np.array([[0,0]])

for i in np.arange(0,np.size(Ar_im,0)):     # Eliminar aro rojo
    for j in np.arange(0,np.size(Ar_im,1)):
        if Ar_im[i,j,3]==255:
            A = np.array([[i,j]])
            Aro = np.concatenate((Aro,A))
Aro = np.delete(Aro,0,0)

for i in np.arange(0,np.size(Aro,0)):
    Ar_im[Aro[i,0],Aro[i,1],3] = 0


Ar = np.array([[0,0,0,0,0]])

for i in np.arange(0,np.size(Ar_im,0)):       # Discriminar las entradas nulas
    for j in np.arange(0,np.size(Ar_im,1)):
        if Ar_im[i,j,3] != 0:
            A = np.array([[Ar_im[i,j,0],Ar_im[i,j,1],Ar_im[i,j,2],i,j]])
            Ar = np.concatenate((Ar,A))
Ar = np.delete(Ar,0,0)

M = np.array([[255,255,255,78],[255,225,255,73],[255,200,255,68],[255,128,255,63],
     #Puntos de referencia
              [255,0,255,58],[255,0,100,53],[255,0,0,48],[255,85,0,43],
              [255,170,0,38],[255,200,0,33],[255,255,0,28],[0,150,50,23],
```

```
110             [0,175,0,18],[0,255,0,13],[0,255,128,10],[0,255,255,-10],[0,0,255,-31.5]])
111
112 Md = np.zeros(16)   # Distancia entre puntos de referencia
113 for i in np.arange(0,16):
114     h = np.sqrt((M[i+1,1]-M[i,1])**2 + (M[i+1,2]-M[i,2])**2)
115     Md[i] = h
116
117 Mv = np.zeros(16)  # Variacion de dBZ entre puntos de referencia
118 for i in np.arange(0,16):
119     h = M[i,3]-M[i+1,3]
120     Mv[i] = h
121
122 Ml = np.zeros(16)  # Factores lineales
123 for i in np.arange(0,16):
124     h = Mv[i]/Md[i]
125     Ml[i] = h
126
127 def f(r,g,b):      # Funcion RGB a dBZ
128     if r <= 30:
129         A = np.zeros(6)
130         for i in np.arange(11,17):
131             A[i-11] = np.sqrt((g-M[i,1])**2 + (b-M[i,2])**2)
132         xmin = min(A)
133         k = np.where(A==xmin)[0][0]
134         A = np.delete(A,k)
135         xmin2 = min(A)
136         k2 = np.where(A==xmin2)[0][0]
137
138         if k==0:
139             Z = M[11,3] - (Ml[11])*xmin
140         elif k==5:
141             Z = M[16,3] + (Ml[15])*xmin
142         else:
143             if k2==k:
144                 Z1 = M[11+k,3] - (Ml[11+k])*xmin
145                 Z2 = M[12+k,3] + (Ml[11+k])*xmin2
146                 Z = (Z1+Z2)/2
147             else:
148                 Z1 = M[11+k,3] + (Ml[10+k])*xmin
149                 Z2 = M[10+k,3] - (Ml[10+k])*xmin2
150                 Z = (Z1+Z2)/2
151
152     if r >= 225:
153         A = np.zeros(11)
154         for i in np.arange(0,11):
155             A[i] = np.sqrt((g-M[i,1])**2 + (b-M[i,2])**2)
156
157         xmin = min(A)
158         k = np.where(A==xmin)[0][0]
159         A = np.delete(A,k)
160         xmin2 = min(A)
161         k2 = np.where(A==xmin2)[0][0]
162
163         if k == 10:
164             Z = M[10,3] + (Ml[9])*xmin
165         elif k == 0:
166             Z = M[0,3] - (Ml[0])*xmin
```

```
        else:
            if k2==k:
                Z1 = M[k,3] - (Ml[k])*xmin
                Z2 = M[k+1,3] + (Ml[k])*xmin2
                Z = (Z1+Z2)/2
            else:
                Z1 = M[k,3] + (Ml[k-1])*xmin
                Z2 = M[k-1,3] - (Ml[k-1])*xmin2
                Z = (Z1+Z2)/2
    return Z

Ar_Z=np.zeros((np.size(Ar,0),3))      # Matriz en dBZ
for i in np.arange(0,np.size(Ar,0)):
    Ar_Z[i,0] = f(Ar[i,0],Ar[i,1],Ar[i,2])
    Ar_Z[i,1] = Ar[i,3]
    Ar_Z[i,2] = Ar[i,4]

############### Asignar coordenadas ################

print('Asignando coordenadas...')

n,s,e,w = float(n),float(s),float(e),float(w)

dvertical = (n-s)/np.size(Ar_im,0)
dhorizontal = (e-w)/np.size(Ar_im,1)

Ar_Rc = np.zeros((np.size(Ar,0),3))      # Matriz escalada
for k in np.arange(0,np.size(Ar_Z,0)):
    Ar_Rc[k,0] = round(Ar_Z[k,0],1)
    Ar_Rc[k,1] = n - (Ar_Z[k,1] + 1/2)*dvertical
    Ar_Rc[k,2] = w + (Ar_Z[k,2] + 1/2)*dhorizontal

############### Exportar Matriz obtenida ###############

print('Exportando datos...')

DF = pd.DataFrame(Ar_Rc)
DF.to_csv("Reflectividad.csv")

print('Listo \(^_^)/')
```

La siguiente sección de código, al ser añadida al *script* anterior e importando la librería `matplotlib.pyplot` como `plt`, genera un gráfico de gradiente de color (ver sección 4).

```
############### Plot ###############

Lat = np.zeros(np.size(Ar_im,0))
for i in np.arange(0,np.size(Ar_im,0)):
    Lat[np.size(Ar_im,0)-(i+1)] = n - (i+1/2)*dvertical

Lon = np.zeros(np.size(Ar_im,1))
for i in np.arange(0,np.size(Ar_im,1)):
    Lon[i] = w + (i+1/2)*dhorizontal

Refmesh = np.zeros((np.size(Ar_im,0),np.size(Ar_im,1)))
for i in np.arange(0,np.size(Ar_im,0)):
    for j in np.arange(0,np.size(Ar_im,1)):
        if Ar_im[i,j,3] == 0:
            h = -50
        else:
```

```
17                 h = f(Ar_im[i,j,0],Ar_im[i,j,1],Ar_im[i,j,2])
18             Refmesh[np.size(Ar_im,0)-(i+1),j] = h
19
20 fig = plt.figure(1)
21 C = plt.contourf(Lon,Lat,Refmesh,100, cmap='Greys')
22 plt.colorbar(C)
23 plt.savefig('Plot.pdf')
```