BatiM9

August 2, 2023

1 Importación de datos de un archivo mat a Python

Autor: Gabriel Ruiz Martínez IMTA. 2023

Cuando se utiliza el perfilador portatil Sontek Surveyor M9 para medir flujos de agua y batimetría en un río o embalse de de agua, el dispositivo electrónico exporta los valores registrados en un archivo mat. Este tipo de archivos requiere que se tenga Matlab para procesar los datos. Sin embargo, Matlab es un programa comercial que en algunas ocasiones no se tiene la licencia para realizar el postprocesamiento de los datos del M9; por tal razón, a continuación se presenta una metodología para importar los datos de un archivo mat a Python.

Importando los módulos:

[1]:
```
import numpy as np
import scipy.io as sio
import pandas as pd
```

Para importar los datos que se encuentran almacenados en el archivo mat, vamos a usar el método `loadmat` del modulo Scipy. Este método importa los datos hacia un diccionario. Cada variable que se encuentra en el archivo mat, le corresponde un clave y valor en el diccionario.

Los nombres de las claves corresponden a los nombres de las estructuras de Matlab. A continuación, vamos a importar los datos de un archivo mat a la variable `datos`. Cabe resaltar que este archivo mat fue creado a partir del post-proceso que se realizó a un archivo riv del dispositivo Sontek Surveyor M9, usado para obtener la batimetría de un cuerpo de agua.

[2]:
```
datos = sio.loadmat('20230719091839', appendmat=True)
```

Vamos a revisar las etiquetas del diccionario con los datos importados.

[3]:
```
sorted(datos.keys())
```

[3]:
```
['BottomTrack',
 'Compass',
 'GPS',
 'Processing',
 'RawGPSData',
 'Setup',
 'SiteInfo',
```

```
 'Summary',
 'System',
 'SystemHW',
 'Transformation_Matrices',
 'WaterTrack',
 '__globals__',
 '__header__',
 '__version__']
```

Para este ejemplo, los datos que nos interesan se encuentran en las etiquetas BottomTrack y GPS. Extraígamos los datos que se encuentran en BottomTrack y vamos a almacenarlos en la variable `databBT`.

[4]:
```
dataBT = datos['BottomTrack']
```

Vamos a identificar el tipo de datos que contiene la variable `dataBT`:

[5]:
```
dataBT.dtype
```

[5]:
```
dtype([('VB_Depth', 'O'), ('BT_Depth', 'O'), ('BT_Vel', 'O'), ('BT_Beam_Depth',
'O'), ('BT_Frequency', 'O'), ('Units', 'O')])
```

Podemos identificar que Scipy lee las estructuras de Matlab como arreglos estructurados de Numpy (ndarray object of Numpy module) y el tipo de dato será "objeto" ('O'). También podemos observar que la estructura BottomTrack, contiene 6 subestructuras donde se encuentran los datos registrados por el dispositivo electrónico M9. Vamos a extraer los datos de las profundidades (valor z) y almacenarlos en sus respectivas variables:

[6]:
```
dBT = datos["BottomTrack"]["BT_Depth"]
```

A continuación, verifiquemos el tamaño y el número de elementos que tienen las variables con los datos de interés

[7]:
```
dBT.size
```

[7]:
```
1
```

[8]:
```
dBT.shape
```

[8]:
```
(1, 1)
```

El tamaño del arreglo estructurado es igual al tamaño de la estructura de Matlab, pero por default, Scipy proporciona una objeto bidimensional. Para comprender que ocurre con el tamaño y número de datos, vamos a revisar el contenido de la variable:

[9]:
```
dBT
```

[9]:
```
array([[array([[0.41599999],
               [0.415     ],
               [0.43399999],
```

```
       …,
       [0.71400002],
       [0.70900003],
       [0.70500002]])]], dtype=object)
```

Observa que el número de datos almacenados es uno, pero el objeto tiene dos dimensiones. Para tener objetos unidimensionales es necesario agregar el parámetro `squeeze_me=True` en el método `loadmat`.

Vamos a borrar todas las variables que hemos generado para volver a cargar los datos, pero ahora vamos a obtener arreglos estructurados de una dimensión:

[10]:
```
del datos, dataBT, dBT
```

[11]:
```
datos = sio.loadmat('20230719091839', appendmat=True, squeeze_me=True)
dBT = datos["BottomTrack"]["BT_Depth"]
```

Vamos a revisar el tamaño, dimensión, tipo de dato y los valores de la variable `dBT`:

[12]:
```
dBT.size
```

[12]:
```
1
```

[13]:
```
dBT.shape
```

[13]:
```
()
```

[14]:
```
dBT.dtype
```

[14]:
```
dtype('O')
```

[15]:
```
dBT
```

[15]:
```
array(array([0.41599999, 0.415     , 0.43399999, …, 0.71400002, 0.70900003,
       0.70500002]), dtype=object)
```

Podemos identificar que `dBT` tiene una dimensión cero, por ello, Python maneja el arreglo como unidimensional. Para mayor detalle, consulta la sección de Calculation en https://numpy.org/doc/stable/reference/arrays.ndarray.html

También se identifica que el tipo de dato sigue siendo un objeto. A continuación vamos a convertir el arreglo "objeto" a un arreglo numérico. Para ello, vamos a realizar una copia del arreglo y lo colapsaremos a un arreglo de una dimensión:

[16]:
```
dBT1d = dBT.flatten()
```

Revisamos la dimensión del nuevo arreglo:

[17]:
```
dBT1d.shape
```

[17]:
```
(1,)
```

[18]:
```
dBT1d
```

[18]:
```
array([array([0.41599999, 0.415     , 0.43399999, …, 0.71400002, 0.70900003,
              0.70500002])                                                    ],
      dtype=object)
```

Vamos a poner el arreglo de manera vertical, después visualizamos el arreglo, así como el tipo de dato que tiene en arreglo:

[19]:
```
dBT1d = np.vstack(dBT1d)
print(dBT1d)
print('Tipo de dato:', dBT1d.dtype)
```

```
[[0.41599999 0.415      0.43399999 … 0.71400002 0.70900003 0.70500002]]
Tipo de dato: float64
```

Para finalizar vamos a crear un arreglo columna, con el número de datos que tiene el arreglo:

[20]:
```
del dBT
dBT = dBT1d.reshape(dBT1d.size)
```

El arreglo, tamaño, tipo de dato y dimension son:

[21]:
```
dBT
```

[21]:
```
array([0.41599999, 0.415     , 0.43399999, …, 0.71400002, 0.70900003,
       0.70500002])
```

[22]:
```
print(dBT.size, 'datos')
print(dBT.shape)
print('Tipos de datos:', dBT.dtype)
print('Tipo de variable:', type(dBT))
```

```
4321 datos
(4321,)
Tipos de datos: float64
Tipo de variable: <class 'numpy.ndarray'>
```

Vamos a importar los datos registrados por el sensor vertical de profundidad del M9:

[23]:
```
dvb = datos["BottomTrack"]["VB_Depth"]
dVb = np.vstack(dvb.flatten())
dVB = dVb.reshape(dVb.size)
del dvb, dVb
```

También, vamos a extraer las coordenadas geográficas de los puntos muestreados (coordenadas espaciales x y y, o latitud y longitud) y almacenarlos en sus respectivas variables. Las coordenadas se encuentran en la etiqueta GPS y en las subestructuras 'UTM', 'longitude' y 'latitude'.

[24]:
```
longi = datos["GPS"]["Longitude"]
longit = np.vstack(longi.flatten())
```

```
lon = longit.reshape(longit.size)
del longi, longit

latid = datos["GPS"]["Latitude"]
lati = np.vstack(latid.flatten())
lat = lati.reshape(lati.size)
del latid, lati

utm = datos["GPS"]["UTM"]
utmd = datos["GPS"]["UTM"]
utm1 = np.stack(utmd.flatten())
utm = utm1.reshape(utm1.size)
del utmd, utm1
```

En el caso de las coordenadas UTM, los datos se encontraban en un arreglo con dos columnas, pero al momento de convertir el arreglo estructurado (objeto) a un arreglo de una columna por n líneas, el arreglo que obtuvimos fue unidimensional, por tal razón, es necesario "filtrar" las coordenadas x y y de dicho arreglo.

[25]:
```
yind = np.where(utm > 1000000)
y = utm[yind]
xind = np.where(utm < 600000)
x = utm[xind]
del yind, xind
```

A manera de respaldo, vamos a exportar todos los datos que fueron importados del archivo mat a un archivo del tipo csv, que puede abrirse con un programa de hojas de cálculo electrónico o un programa que permita abrir archivos de texto plano. La exportación la haremos usando el modulo Pandas. Por tal razón, vamos almacenar todos los arreglos que nos interesan en un DataFrame de Pandas:

[26]:
```
data = pd.DataFrame({'lon(°)':lon, 'lat(°)':lat, 'x(m)':x, 'y(m)':y, 'BT(m)':
  ↪dBT, 'VB(m)':dVB})
print(data.head())
```

```
      lon(°)     lat(°)           x(m)          y(m)  BT(m)  VB(m)
0 -92.847107  15.203079  516421.539201  1.680793e+06  0.416  0.412
1 -92.847107  15.203079  516421.539201  1.680793e+06  0.415  0.437
2 -92.847107  15.203079  516421.539201  1.680793e+06  0.434  0.435
3 -92.847107  15.203079  516421.539201  1.680793e+06  0.436  0.000
4 -92.847107  15.203079  516421.539201  1.680793e+06  0.419  0.000
```

A continuación vamos a exportar el DataFrame al archivo csv:

[27]:
```
data.to_csv('datos.csv', header=False, index=False)
```