File di esempio per la lettura dei dati Raman (spettrometro BWSpec)

Pubblichiamo in questa cartella alcuni file di esempio che mostrano come leggere i file di dati prodotti dal software BWSpec ® 4.11_1 software con settings di default (files .txt).

Gli esempi sono scritti sia in Python (testato con Python 3.11.1 su Windows 10) che in Matlab (testato con MATLAB 9.13 (R2022b) Update 3 e con GNU Octave 7.3.0, entrambi su Windows 10).

Per ciascuno dei due linguaggi mettiamo a disposizione

- Una funzione che legge i dati dal file, da due colonne selezionate dall'utente
- Due script che, appoggiandosi alla funzione, rispettivamente plottano i dati e salvano le due colonne selezionate.

Accompagna il software un file dati di test (`Raman_spectrum_file.txt`).

Per provare i programmi basta copiare tutti i file (incluso il file di dati) nella stessa directory e eseguire gli script.

Il file "read" è la funzione che legge i file. "plot" e "save" sono script. Lo script "plot" genera un plot dei dati presenti nelle colonne selezionate dall'utente. Lo script "save" scrive su disco un file contenente solo le colonne selezionate; il nome del file sul quale le colonne vengono scritte viene composto apponendo al nome del file dati originale il suffisso "_columns".

I programmi vengono forniti **senza garanzie**. In particolare, raccomandiamo di fare una copia dei vostri dati prima di usarli come dati di input di questi programmi.

Trovate di seguito una tavola con la lista dei programmi e una descrizione dettagliata delle funzioni, che si rivolge agli utenti che non hanno, o non hanno ancora, esperienza di programmazione con Python o Matlab. Gli script contengono anche alcune linee di commento che speriamo siano sufficienti per chiarire il loro funzionamento.

Lista dei programmi

Linguaggio	Nome del file	Cosa fa
Python	`read_BWSpec_data_file.py`	Funzione – legge i dati dal file BWSpec
Python	`plot_BWSpec_Raman_spectrum.py`	Script – plotta i dati
Python	`save_BWSpec_Raman_spectrum_columns.py`	Script – salva su disco le colonne selezionate
Matlab	`read_BWSpec_data_file.m`	Funzione – legge i dati dal file BWSpec
Matlab	`plot_BWSpec_Raman_spectrum.m`	Script – plots data
Matlab	`save_BWSpec_Raman_spectrum_columns.m`	Script – salva su disco le colonne selezionate

Descrizione dettagliata delle funzioni

I programmi "read" sono funzioni che leggono i file di dati e scrivono i valori in una array a due colonne che è il valore che viene restituito dalla chiamata.

Le funzioni scansionano il file linea per linea finché incontrano la linea di intestazione dei dati. Una volta che le funzioni arrivano alla linea di intestazione, esse leggono il resto dei dati linea per linea, analizzano la linea utilizzando il punto e virgola come separatore di colonna e scrivono i dati contenuti nelle colonne selezionate dall'utente in una riga dell'array di output.

Python - read_BWSpec_data_file.py

Il programma apre il file di dati mediante un blocco `with`. Il blocco `with` in Python è un cosiddetto "gestore di contesto" (in inglese "context manager"); "gestisce" l'allocazione e la liberazione di una risorsa di sistema, permettendo all'utente di utilizzarla senza dover scrivere il codice per la gestione stessa. Nel nostro caso la risorsa è un file; l'istruzione `with` apre il file e consente a Python di chiuderlo quando l'esecuzione del blocco è terminata, indipendentemente dalla maniera in cui il termine dell'esecuzione del blocco è avvenuto (*id est* Python chiude il file anche se l'esecuzione del codice è fermata da una condizione di errore).

Nel file cerchiamo la linea di intestazione dei dati linea per linea mediante il comando `readline()`, che legge la linea alla posizione attuale e avanza alla linea seguente; `readline()` è eseguito in un blocco `while`, controllando ad ogni iterazione se si è raggiunta la linea di intestazione. Dato che il numero di linee prima dell'intestazione può variare (cfr. manuale del software BWSpec), non possiamo trovare l'inizio dei dati contando le linee, e cercare l'intestazione è necessario.

Una volta che abbiamo raggiunto la linea di intestazione, continuiamo a leggere il file linea per linea, questa volta con un ciclo `for`, analizzando ciascuna linea; l'analizzatore (`line_split = line.split(";")`) suddivide la linea in una lista di stringhe in corrispondenza dei punti e virgola. Dalla lista di stringhe scegliamo poi le colonne richieste dall'utente per scriverle su un array, che abbiamo preparato in anticipo, mediante il comando

```
spectrum[p,:] = [float(line_split[q])
  for q in [abscissa_column, spectrum_column]]
```

La sintassi nel codice sopra presentato, ponendo l'essenziale in forma di esempio, è

```
newList = [process(q) for q in list_of_qs]
```

È un costrutto di Python chiamato in inglese "list comprehension" (una ricerca su Google dell'equivalente italiano "comprensione di lista" dà poche decine di risultati), che permette di generare una lista (`newList` nel nostro esempio) partendo da una lista di input (`list_of_qs`), applicando una funzione (`process`) a tutti i suoi elementi e raccogliendo i risultati in una lista di dimensione uguale a quella della lista di input.

Una volta che il ciclo `for` è terminato, esaurendo le linee del file di dati, l'array `spectrum` contiene le colonne di dati richieste e diventa il valore restituito dalla funzione.

read_BWSpec_data_file.py - Notes

La funzione controlla se la linea di intestazione esiste; se il ciclo di lettura non trova un intestazione, la funzione si ferma con un codice di errore. A parte il controllo sull'esistenza dell'intestazione, la funzione non esegue nessun altro controllo sulla correttezza del file di dati: né sul formato di ciascuna linea di dati, né sulla presenza di 2028 linee di dati; il buon funzionamento della funzione dipende dalla correttezza del file scritto dal software dello spettrometro e dato come input alla funzione dall'utente.

La lettura del file e la scrittura dei dati su un array avrebbero potuto essere realizzati con costrutti Python più sintetici.

Per esempio, avremmo potuto definire una funzione per analizzare le linee (`parse`) che estraesse le colonne richieste e avremmo potuto applicarla su tutte le linee dati del file eseguendo

```
spectrum = [parse (line) for line in f_spectrum]
```

dopo aver trovato l'intestazione; dopo di ché sarebbe stato possibile convertire `spectrum`, ora una lista di liste, in un array della classe `numpy`.

Un'altra possibilità è la funzione `loadtxt` del pacchetto NumPy (pagina di aiuto loadtxt), che per i file generati da BWSpec deve essere usata in congiunzione ad una ricerca dell'intestazione dei dati.

Abbiamo deciso di scrivere il programma in maniera esplicita (anche se più lunga) sperando che questo aiuti gli eventuali utenti che si stiano familiarizzando con la programmazione.

Matlab (and Octave) - read_BWSpec_data_file.m

La funzione scritta per Matlab segue la stessa logica della funzione scritta in Python: scansiona il file cercando la linea di intestazione, poi legge linea per linea, analizza ciascuna linea e scrive le colonne selezionate dall'utente in un array, che infine restituisce come valore della chiamata.

Una importante differenza fra il codice Python e quello Matlab è che in Matlab non esistono "gestori di contesti" espliciti come in Python. Un modo semplice di scrivere il nostro codice è quello di aprire e chiudere il file di dati in maniera esplicita, in particolare chiudendo il file prima di dichiarare condizioni di errore.

Una lettura potenzialmente interessante sulla gestione dei file in Matlab è un post di Loren Shure nel suo blog "Loren on the art of Matlab". L'autore della funzione che stiamo mettendo a disposizione non è in grado di prendere posizione su quale sia il metodo migliore per scrivere codice che legga da file in Matlab; nella funzione, diamo l'istruzione `fclose` prima di dichiarare una condizione di errore.

read_BWSpec_data_file.m - Notes

Importanti differenze fra il codice Matlab e quello Python sono

- L'istruzione `fgetl` di Matlab elimina le andate a capo (help page di fgetl) mentre `readline` di Python non lo fa; il confronto fra la linea di intestazione e le linee lette dal file deve tenerne conto. Per informazioni sulle andate a capo in Python vedere Input and output tutorial; dettagli importanti sui caratteri di andata a capo si trovano anche nella documentazione dell'argomento di input `newline` della funzione `open`.

- Il comando `strsplit` di Matlab, quando è usato con le impostazioni di default, fa coalescere delimitatori successivi (pagina di help di strsplit), mentre il metodo `split` di Python, quando riceve un argomento di input (ad esempio il nostro `line.split(";")`), non lo fa (documentazione di str.split)

- Matlab non ha la "list comprehension", e, sebbene abbia una funzione per il mapping degli array (che avremmo potuto usare ottenendo lo stesso effetto), abbiamo deciso di costruire esplicitamente ogni riga dell'array contenente lo spettro con

```
spectrum(p,:) = [str2double(line_list(abscissa_column))
str2double(line_list(spectrum_column))];
```

Avremmo potuto scrivere il codice Matlab utilizzando alcune funzioni di Matlab che risparmiano lavoro all'utente; per esempio `readlines` legge tutte le linee di un file di testo resituendo in output un array di stringhe e `find` può trovare l'indice di un elemento in un array (e quindi avremmo potuto usarlo per trovare l'intestazione dei dati). In questo caso avremmo scritto codice più conciso; come per il programma in Python, però, abbiamo preferito codice scritto in maniera dettagliata a codice conciso, sperando di essere d'aiuto agli utenti che si stiano familiarizzando con la programmazione.

Così facendo abbiamo anche ottenuto codice utilizzabile sia con Matlab che con Octave. In questo senso, se si desidera scrivere codice conciso, è necessario tenere presente che la compatibilità tra Matlab e Octave non è completa, quindi può essere necessario scrivere o per l'uno oppure per l'altro dei due programmi.