

```
1 import numpy as np
2 from scipy.signal import butter, filtfilt
3 from scipy.interpolate import interp1d
4 from scipy.ndimage import rotate # Utilizzato per la
   rotazione multidirezionale (a raggiera) della
   matrice
5
6
7 def mean_profile_depth(TX, OPT=None):
8     """
9     Calcola il Mean Profile Depth (MPD) di un singolo
   profilo lineare 1D.
10     Traduzione e adattamento dello script standard
   MATLAB di Peter Andrén,
11     conforme ai requisiti imposti dalla norma
   internazionale ISO 13473-1.
12
13     :param TX: Array 1D contenente le quote
   altimetriche del profilo (in mm).
14     :param OPT: Dizionario opzionale contenente i
   parametri di configurazione.
15     :return: Dizionario con MSD (Mean Segment Depth
   ), MPD del profilo e Deviazione Standard.
16     """
17     # Se non vengono passate opzioni, inizializza un
   dizionario vuoto per evitare errori
18     if OPT is None:
19         OPT = {}
20
21     # Estrazione dei parametri dal dizionario (con
   valori di default in caso di assenza)
22     profile_dx = OPT.get('profile_dx', 0.1) #
   Spaziatura di campionamento (passo del raster in mm)
23     spot_measurement = OPT.get('spot_measurement',
   True) # Se True, abilita le correzioni per provini
   di laboratorio
24
25     # Converte l'input in un vettore NumPy di tipo
   float a una sola dimensione (appiattimento)
26     TX = np.asarray(TX, dtype=float).flatten()
27     DX = profile_dx # Assegnazione del passo
```

```

27 spaziale alla variabile DX per brevità matematica
28
29
    # -----
    -----
30     # FASET 1: GESTIONE E RIMOZIONE DEI DATI MANCANTI
    (NaN - Not a Number)
31
    # -----
    -----
32     if np.any(np.isnan(TX)):
33         # Identifica gli indici di tutti i punti
    validi (finiti e non NaN)
34         ok_samples = np.where(np.isfinite(TX))[0]
35
36         # Se più del 50% del profilo è composto da
    NaN, il profilo viene giudicato non idoneo
37         if len(ok_samples) < len(TX) * 0.5:
38             return {'MPD': np.nan, 'MSD': np.nan, '
    STD': np.nan}
39
40         # Genera l'asse delle X completo (da 0 a
    lunghezza profilo) per la ricostruzione
41         x_all = np.arange(len(TX))
42
43         # Interpolazione lineare per ricostruire i
    buchi di dato interni al profilo
44         f_lin = interp1d(ok_samples, TX[ok_samples],
    kind='linear', bounds_error=False, fill_value=np.nan)
45         TX_lin = f_lin(x_all)
46
47         # Interpolazione geometrica "nearest" (punto
    più vicino) per estrapolare i dati sui bordi esterni
48         f_near = interp1d(ok_samples, TX[ok_samples
    ], kind='nearest', bounds_error=False, fill_value='
    extrapolate')
49
50         # Unisce le due interpolazioni: usa la
    lineare dove possibile, la nearest dove la lineare
    fallisce (bordi)
51         TX = np.where(np.isnan(TX_lin), f_near(x_all

```

```

51 ), TX_lin)
52
53
54     # -----
55     # FASE 2: DETEZIONE ED ELIMINAZIONE DEGLI SPIKE (
56     # RUMORE STRUMENTALE)
57     # -----
58
59     # Secondo ISO 13473-1, un punto è uno spike se la
60     # differenza di quota con il punto
61     # precedente supera 3 volte il passo di
62     # campionamento spaziale (3 * DX).
63     spike_pos = set()
64     diff_TX = np.abs(np.diff(TX)) # Calcola la
65     # differenza in valore assoluto tra punti adiacenti
66
67     # Controllo in avanti (Forward check): identifica
68     # anomalie da sinistra a destra
69     fwd_spikes = np.where(diff_TX >= 3 * DX)[0] + 1
70
71     # Controllo all'indietro (Backward check):
72     # ribalta il profilo e analizza da destra a sinistra
73     TX_rev = np.flip(TX)
74     rev_spikes = (len(TX) - 1) - (np.where(np.abs(np.
75     diff(TX_rev)) >= 3 * DX)[0] + 1)
76
77     # Unisce gli indici degli spike trovati nei due
78     # controlli evitando duplicati grazie a un 'set'
79     spike_pos.update(fwd_spikes)
80     spike_pos.update(rev_spikes)
81     spike_pos = list(spike_pos)
82
83     # Se vengono rilevati spike, questi vengono
84     # rimossi e sostituiti mediante interpolazione
85     if spike_pos:
86         TX[spike_pos] = np.nan # Trasforma gli spike
87         # temporaneamente in NaN
88         ok_samples = np.where(np.isfinite(TX))[0]

```

```

78         if len(ok_samples) == 0:
79             return {'MPD': np.nan, 'MSD': np.nan, '
STD': np.nan}
80
81         x_all = np.arange(len(TX))
82         # Ricostruisce i punti eliminati assegnando
la quota del punto valido più vicino (nearest)
83         f_near = interp1d(ok_samples, TX[ok_samples
], kind='nearest', bounds_error=False, fill_value='
extrapolate')
84         TX = f_near(x_all)
85
86
# -----
-----
87     # FASE 3: SUDDIVISIONE IN SEGMENTI STANDARD DA
100 MM
88
# -----
-----
89     # La norma impone che il calcolo avvenga
rigorosamente su segmenti di 100 mm.
90     # Calcola quanti pixel/punti servono per coprire
100 mm (es: con dx=0.05 mm servono 2000 punti)
91     samples_per_segment = int(100 * (1 / DX))
92
93     # Se la porzione di profilo utile estratta è più
corta di 100 mm, il calcolo non può essere eseguito
94     if len(TX) < samples_per_segment:
95         return {'MPD': np.nan, 'MSD': np.nan, 'STD'
: np.nan}
96
97     # Calcola quanti segmenti interi da 100 mm
entrano nel profilo (per i provini di lab di solito
è 1)
98     num_segments = len(TX) // samples_per_segment
99
100     # Ritaglia l'array eliminando i pixel in eccesso
alla fine e lo rimodella (reshape) in una matrice
101     # dove ogni riga rappresenta un segmento
indipendente lungo esattamente 100 mm.

```

```

102     TX_MAT = TX[:num_segments * samples_per_segment
103                  ].reshape((num_segments, samples_per_segment))
104
105     # -----
106     # FASE 4: SOPPRESSIONE DELLA PENDENZA (Slope
107     # Suppression per Provini Lab)
108     # -----
109     # Operazione cruciale in laboratorio: corregge l
110     # 'eventuale inclinazione del provino
111     # sul piatto dello scanner applicando una
112     # regressione lineare dei minimi quadrati.
113     if spot_measurement:
114         # Calcola la quota media di ogni segmento (
115         # riga)
116         mean_vals = np.mean(TX_MAT, axis=1, keepdims
117                             =True)
118         # Sottrae temporaneamente la media per
119         # centrare il profilo attorno allo zero
120         y_star = TX_MAT - mean_vals
121
122         # Genera il vettore delle coordinate
123         # spaziali X centrato sullo zero (es: da -50 mm a +50
124         # mm)
125         x_vec = np.arange(-100 / 2 + DX / 2, 100 / 2
126                           , DX)
127
128         # Calcolo analitico dei coefficienti della
129         # retta di regressione (Inclinazione BETA e Intercetta
130         # ALPHA)
131         BETA = np.dot(y_star, x_vec) / np.sum(x_vec
132         ** 2)
133         ALPHA = np.mean(TX_MAT, axis=1) - (100 / 2
134         + DX / 2) * BETA
135
136         # Sottrae matematicamente la retta di
137         # pendenza da ogni singolo punto del segmento
138         for i in range(num_segments):

```

```

124             TX_MAT[i, :] -= (x_vec + 100 / 2 - DX /
125             2 + DX) * BETA[i] + ALPHA[i]
126
127             # -----
128             # FASE 5: DETERMINAZIONE MATEMATICA DEL VALORE
129             MSD E MPD
130             # -----
131             # Divide ogni riga da 100 mm in due segmenti da
132             50 mm ciascuno.
133             # Trova il picco massimo nella prima metà (
134             Fascia A: da 0 a 50 mm)
135             PSA_max = np.max(TX_MAT[:, :samples_per_segment
136             // 2], axis=1)
137             # Trova il picco massimo nella seconda metà (
138             Fascia B: da 50 a 100 mm)
139             PSB_max = np.max(TX_MAT[:, samples_per_segment
140             // 2:], axis=1)
141             # Calcola la quota media dell'intero segmento
142             filtrato
143             PS_mean = np.nanmean(TX_MAT, axis=1)
144
145             # Formula ISO 13473-1 per il Mean Segment Depth
146             : media dei due picchi meno la quota media
147             MSD = (PSA_max + PSB_max) / 2 - PS_mean
148
149             # Il Mean Profile Depth (MPD) finale del profilo
150             è la media matematica dei valori di MSD
151             MPD = np.nanmean(MSD)
152             STD = np.nanstd(MSD) # Deviazione standard (
153             indica la variabilità della tessitura)
154
155             return {'MSD': MSD, 'MPD': MPD, 'STD': STD}
156
157 def process_dem(DEM, OPT=None, row_step=1, col_step=
158 1):
159     """

```

```
149     Elabora l'intera matrice 2D del Modello Digitale
      di Elevazione (DEM).
150     Applica una strategia d'analisi a raggiera (360
      °) ruotando geometricamente il provino.
151
152
153     :param DEM: Matrice 2D (NumPy array) contenente
      le quote del provino in millimetri.
154     :param OPT: Dizionario delle opzioni di calcolo.
155     :param row_step: Passo di campionamento delle
      righe centrali (1 = analizza ogni riga).
156     :param col_step: Non più influente in modalità
      raggiera (mantenuto per compatibilità strutturale).
157     :return: Dizionario contenente le medie globali
      stimate di MPD ed ETD.
158     """
159     # Inizializzazione delle opzioni di default se
      non fornite
160     if OPT is None:
161         OPT = {'profile_dx': 0.05, 'spot_measurement
      ': True, 'calculate_EDT': True}
162
163     dx = OPT.get('profile_dx', 0.05) # Risoluzione
      del pixel in mm
164     calculate_EDT = OPT.get('calculate_EDT', True)
      # Flag per abilitare il calcolo dell'ETD norma ISO
165
166     mpd_tutti = [] # Lista in cui verranno
      accumulati i valori di MPD di tutti i profili validi
      scansionati
167
168     # -----
      -----
169     # CORE: SCANSIONE MULTIDIREZIONALE A RAGGIERA (
      ROTAZIONE A 360°)
170
171     # -----
      -----
171     # Ruota la matrice da 0° a 180° con scatti di 10
      gradi. Analizzando l'asse orizzontale
```

```

172     # ad ogni rotazione, andiamo di fatto a coprire
    l'intera circonferenza del provino.
173     for angolo in range(0, 180, 10):
174
175         # Ruota fisicamente la matrice DEM. I nuovi
    spazi vuoti geometrici generati agli angoli
176         # della matrice quadrata vengono riempiti
    forzatamente con valore 0.0 (cval=0.0)
177         dem_ruotata = rotate(DEM, angolo, reshape=
    False, order=1, cval=0.0)
178
179         rows, cols = dem_ruotata.shape
180         center_r, center_c = rows // 2, cols // 2
    # Individua il pixel centrale del provino
181
182         # Determina quanti pixel servono a destra e
    a sinistra del centro per fare 50+50 = 100 mm totali
183         punti_semi_profilo = int(50 / dx)
184
185         # Estrae una fascia fitta di righe (profili
    ) posizionate a ridosso del diametro centrale (+/-
    10 righe).
186         # Questo garantisce di campionare materiale
    reale sfruttando la massima estensione utile del
    cerchio.
187         for i in range(center_r - 10, center_r + 10
    , row_step):
188             if 0 <= i < rows:
189                 # Ritaglia il profilo lineare lungo
    esattamente 100 mm centrato rispetto al cuore del
    provino
190                 profile = dem_ruotata[i, center_c -
    punti_semi_profilo: center_c + punti_semi_profilo]
191
192                 # CONTROLLO DI SICUREZZA GEOMETRICA
    (Anti-Bordo)
193                 # Se il profilo intercetta anche
    solo un pixel con quota <= 0.1 mm, significa che
    siamo usciti
194                 # dalla faccia del conglomerato e
    siamo finiti sullo sfondo vuoto. Il profilo viene

```

```

194 SCARTATO.
195             if np.any(profile <= 0.1):
196                 continue
197
198             # Se il profilo passa il controllo
di sicurezza, viene inviato all'algoritmo di calcolo
dell'MPD
199             res = mean_profile_depth(profile,
OPT)
200
201             # Se il calcolo ha successo (
risultato valido e non NaN), salva il valore nella
lista globale
202             if not np.isnan(res['MPD']):
203                 mpd_tutti.append(res['MPD'])
204
205
# -----
-----
206     # FASE 6: STATISTICA GLOBALE ED ESTIMATED
TEXTURE DEPTH (ETD)
207
# -----
-----
208     # Calcola l'MPD Globale del provino facendo la
media di centinaia di profili estratti a raggiera
209     global_mpd = np.mean(mpd_tutti) if mpd_tutti
else np.nan
210
211     # Formula di correlazione empirica definita
dalla norma ISO 13473-1 per convertire
212     # l'indice bidimensionale MPD nella stima
volumetrica tridimensionale ETD (equivalente all'
altezza d'inondazione della sabbia MTD)
213     global_etd = 1.1 * global_mpd if not np.isnan(
global_mpd) and calculate_EDT else np.nan
214
215     # Restituisce i risultati strutturati in un
dizionario pronto per la visualizzazione
216     return {
217         'avg_MPD_rows': global_mpd, # Mappato sul

```

```

217 globale per preservare la struttura di stampa
    originaria
218         'avg_MPD_cols': global_mpd,
219         'global_MPD': global_mpd,
220         'global_ETD': global_etd,
221         'total_profiles_analyzed': len(mpd_tutti)
    # Numero totale di profili effettivi usati nella
    statistica
222     }
223
224
225 # =====
    =====
226 #                PANNELLO DI CONFIGURAZIONE GUIDATA
    (MAIN SCRIPT)
227 # =====
    =====
228 if __name__ == '__main__':
229
230     # Definizione del percorso assoluto o relativo
    del file DEM esportato in formato testo (.txt)
231     percorso_file = r"C:\Users\vinny\Desktop\
    Tesi_Falcini\DATA PROGETTO PHYTON\per codice MPD\
    Provino in Bitume\kriging\provino di bitume, scalato
    in mm, s=0.1 mm.txt"
232
233     print("Caricamento del file DEM in corso (
    operazione richiesta per matrici grandi)...")
234     try:
235         # Carica la matrice dei numeri isolati da
    spazi bianchi nel formato NumPy Array ad alte
    prestazioni
236         dem_matrix = np.loadtxt(percorso_file)
237         print(f"-> File caricato! Dimensione matrice
    : {dem_matrix.shape[0]}x{dem_matrix.shape[1]} pixel
    .")
238     except Exception as e:
239         print(f"▲ Errore nel caricamento del file: {
    e}")
240         exit()
241

```

```
242     # IMPOSTAZIONE DEI SALTI DI CAMPIONAMENTO (
    DENSITÀ DI ANALISI)
243     # Impostando a 1 analizziamo consecutivamente
    tutte le linee della fascia radiale centrale per la
    massima precisione.
244     passo_righe = 2
245     passo_colonne = 3
246
247     # IMPOSTAZIONI METROLOGICHE DELL'ALGORITMO
248     opzioni_mpd = {
249         'profile_dx': 0.1, # PASSO DI SCANSIONE
    REALE (in mm) configurato durante la rasterizzazione
    su CloudCompare
250         'spot_measurement': True, # Attiva la
    soppressione della pendenza del provino (Slope
    Suppression)
251         'calculate_EDT': True, # Ordina all'
    algoritmo di calcolare il parametro ETD (1.1 * MPD)
252         'verbose': False # Se True, stamperebbe a
    console i log matematici intermedi di ogni riga
253     }
254
255     print("\nAvvio elaborazione MPD con filtro di
    campionamento centrale...")
256     # Esecuzione del calcolo principale sulla
    matrice caricata
257     risultati = process_dem(dem_matrix, OPT=
    opzioni_mpd, row_step=passo_righe, col_step=
    passo_colonne)
258
259     # -----
    -----
260     # SCHERMATA FINALE DI STAMPA DEI RISULTATI
    VALIDATI
261
    # -----
    -----
262     print("\n" + "=" * 45)
263     print("          RISULTATI DELL'ANALISI CORRETTA
    ")
```

```
264     print("=" * 45)
265     print(f"Passo selezione profili      -> Righe:
ogni {passo_righe} | Direzioni: 360° a raggiera")
266     print(f"Profili interni analizzati  -> {
risultati['total_profiles_analyzed']}")
267     print("-" * 45)
268     print(f"MPD Medio dei profili RIGA    : {
risultati['avg_MPD_rows']:.4f} mm")
269     print(f"MPD Medio dei profili COLONNA: {
risultati['avg_MPD_cols']:.4f} mm")
270     print("-" * 45)
271     print(f"MPD GLOBALE DEL PROVINO      : {risultati
['global_MPD']:.4f} mm")
272     print(f"ETD STIMATO (1.1 x MPD)      : {risultati
['global_ETD']:.4f} mm")
273     print("=" * 45)
```