

```
1 # This is a sample Python script.
2
3 # Press Maiusc+F10 to execute it or replace it with
  your code.
4 # Press Double Shift to search everywhere for classes
  , files, tool windows, actions, and settings.
5
6
7 import numpy as np
8 from scipy.signal import butter, filtfilt
9 from scipy.interpolate import interp1d
10
11
12 # Nota: 'rotate' da scipy.ndimage non è più
  necessario in quanto l'analisi per lastre è
  ortogonale rettilinea
13
14
15 def mean_profile_depth(TX, OPT=None):
16     """
17     Calcola il Mean Profile Depth (MPD) di un singolo
      profilo lineare 1D.
18     Traduzione e adattamento dello script standard
      MATLAB di Peter Andrén,
19     conforme ai requisiti imposti dalla norma
      internazionale ISO 13473-1.
20
21     :param TX: Array 1D contenente le quote
      altimetriche del profilo (in mm).
22     :param OPT: Dizionario opzionale contenente i
      parametri di configurazione.
23     :return: Dizionario con MSD (Mean Segment Depth
      ), MPD del profilo e Deviazione Standard.
24     """
25     # Se non vengono passate opzioni, inizializza un
      dizionario vuoto per evitare errori
26     if OPT is None:
27         OPT = {}
28
29     # Estrazione dei parametri dal dizionario (con
      valori di default in caso di assenza)
```

```

30     profile_dx = OPT.get('profile_dx', 0.1) #
    Spaziatura di campionamento (passo del raster in mm)
31     spot_measurement = OPT.get('spot_measurement',
    True) # Se True, abilita le correzioni per provini
    di laboratorio
32
33     # Converte l'input in un vettore NumPy di tipo
    float a una sola dimensionale (appiattimento)
34     TX = np.asarray(TX, dtype=float).flatten()
35     DX = profile_dx # Assegnazione del passo
    spaziale alla variabile DX per brevità matematica
36
37     # -----
    -----
38     # FASE 1: GESTIONE E RIMOZIONE DEI DATI MANCANTI
    (NaN - Not a Number)
39
    # -----
    -----
40     if np.any(np.isnan(TX)):
41         # Identifica gli indici di tutti i punti
    validi (finiti e non NaN)
42         ok_samples = np.where(np.isfinite(TX))[0]
43
44         # Se più del 50% del profilo è composto da
    NaN, il profilo viene giudicato non idoneo
45         if len(ok_samples) < len(TX) * 0.5:
46             return {'MPD': np.nan, 'MSD': np.nan, '
    STD': np.nan}
47
48         # Genera l'asse delle X completo (da 0 a
    lunghezza profilo) per la ricostruzione
49         x_all = np.arange(len(TX))
50
51         # Interpolazione lineare per ricostruire i
    buchi di dato interni al profilo
52         f_lin = interp1d(ok_samples, TX[ok_samples],
    kind='linear', bounds_error=False, fill_value=np.nan)
53         TX_lin = f_lin(x_all)
54

```

```

55         # Interpolazione geometrica "nearest" (punto
        più vicino) per estrapolare i dati sui bordi esterni
56         f_near = interp1d(ok_samples, TX[ok_samples
        ], kind='nearest', bounds_error=False, fill_value='
        extrapolate')
57
58         # Unisce le due interpolazioni: usa la
        lineare dove possibile, la nearest dove la lineare
        fallisce (bordi)
59         TX = np.where(np.isnan(TX_lin), f_near(x_all
        ), TX_lin)
60
61
        # -----
        -----
62     # FASE 2: DETEZIONE ED ELIMINAZIONE DEGLI SPIKE (
        RUMORE STRUMENTALE)
63
        # -----
        -----
64     # Secondo ISO 13473-1, un punto è uno spike se la
        differenza di quota con il punto
65     # precedente supera 3 volte il passo di
        campionamento spaziale (3 * DX).
66     spike_pos = set()
67     diff_TX = np.abs(np.diff(TX)) # Calcola la
        differenza in valore assoluto tra punti adiacenti
68
69     # Controllo in avanti (Forward check): identifica
        anomalie da sinistra a destra
70     fwd_spikes = np.where(diff_TX >= 3 * DX)[0] + 1
71
72     # Controllo all'indietro (Backward check):
        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.
        diff(TX_rev)) >= 3 * DX)[0] + 1)
75
76     # Unisce gli indici degli spike trovati nei due
        controlli evitando duplicati grazie a un 'set'
77     spike_pos.update(fwd_spikes)

```

```

78     spike_pos.update(rev_spikes)
79     spike_pos = list(spike_pos)
80
81     # Se vengono rilevati spike, questi vengono
    rimossi e sostituiti mediante interpolazione
82     if spike_pos:
83         TX[spike_pos] = np.nan # Trasforma gli
    spike temporaneamente in NaN
84         ok_samples = np.where(np.isfinite(TX))[0]
85
86         if len(ok_samples) == 0:
87             return {'MPD': np.nan, 'MSD': np.nan, '
    STD': np.nan}
88
89         x_all = np.arange(len(TX))
90         # Ricostruisce i punti eliminati assegnando
    la quota del punto valido più vicino (nearest)
91         f_near = interp1d(ok_samples, TX[ok_samples
    ], kind='nearest', bounds_error=False, fill_value='
    extrapolate')
92         TX = f_near(x_all)
93
94
    # -----
    -----
95     # FASE 3: SUDDIVISIONE IN SEGMENTI STANDARD DA
    100 MM
96
    # -----
    -----
97     # La norma impone che il calcolo avvenga
    rigorosamente su segmenti di 100 mm.
98     # Calcola quanti pixel/punti servono per coprire
    100 mm (es: con dx=0.1 mm servono 1000 punti)
99     samples_per_segment = int(100 * (1 / DX))
100
101     # Se la porzione di profilo utile estratta è più
    corta di 100 mm, il calcolo non può essere eseguito
102     if len(TX) < samples_per_segment:
103         return {'MPD': np.nan, 'MSD': np.nan, 'STD'
    : np.nan}

```

```

104
105     # Calcola quanti segmenti interi da 100 mm
    entrano nel profilo
106     num_segments = len(TX) // samples_per_segment
107
108     # Ritaglia l'array eliminando i pixel in eccesso
    alla fine e lo rimodella (reshape) in una matrice
109     # dove ogni riga rappresenta un segmento
    indipendente lungo esattamente 100 mm.
110     TX_MAT = TX[:num_segments * samples_per_segment
    ].reshape((num_segments, samples_per_segment))
111
112
    # -----
    -----
113     # FASE 4: SOPPRESSIONE DELLA PENDENZA (Slope
    Suppression per Laboratorio)
114
    # -----
    -----
115     # Operazione cruciale: corregge l'eventuale
    inclinazione geometrica della lastra
116     # sul piatto dello scanner applicando una
    regressione lineare dei minimi quadrati.
117     if spot_measurement:
118         # Calcola la quota media di ogni segmento (
    riga)
119         mean_vals = np.mean(TX_MAT, axis=1, keepdims
    =True)
120         # Sottrae temporaneamente la media per
    centrare il profilo attorno allo zero
121         y_star = TX_MAT - mean_vals
122
123         # Genera il vettore delle coordinate
    spaziali X centrato sullo zero (es: da -50 mm a +50
    mm)
124         x_vec = np.arange(-100 / 2 + DX / 2, 100 / 2
    , DX)
125
126         # Calcolo analitico dei coefficienti della
    retta di regressione (Inclinazione BETA e Intercetta

```

```

126 ALPHA)
127     BETA = np.dot(y_star, x_vec) / np.sum(x_vec
    ** 2)
128     ALPHA = np.mean(TX_MAT, axis=1) - (100 / 2
    + DX / 2) * BETA
129
130     # Sottrae matematicamente la retta di
    pendenza da ogni singolo punto del segmento
131     for i in range(num_segments):
132         TX_MAT[i, :] -= (x_vec + 100 / 2 - DX /
    2 + DX) * BETA[i] + ALPHA[i]
133
134     # -----
    -----
135     # FASE 5: DETERMINAZIONE MATEMATICA DEL VALORE
    MSD E MPD
136
137     # -----
    -----
137     # Divide ogni riga da 100 mm in due
    semiregumenti da 50 mm ciascuno.
138     # Trova il picco massimo nella prima metà (
    Fascia A: da 0 a 50 mm)
139     PSA_max = np.max(TX_MAT[:, :samples_per_segment
    // 2], axis=1)
140     # Trova il picco massimo nella seconda metà (
    Fascia B: da 50 a 100 mm)
141     PSB_max = np.max(TX_MAT[:, samples_per_segment
    // 2:], axis=1)
142     # Calcola la quota media dell'intero segmento
    filtrato
143     PS_mean = np.nanmean(TX_MAT, axis=1)
144
145     # Formula ISO 13473-1 per il Mean Segment Depth
    : media dei due picchi meno la quota media
146     MSD = (PSA_max + PSB_max) / 2 - PS_mean
147
148     # Il Mean Profile Depth (MPD) finale del profilo
    è la media matematica dei valori di MSD
149     MPD = np.nanmean(MSD)

```

```

150     STD = np.nanstd(MSD) # Deviazione standard (
    indica la variabilità della tessitura)
151
152     return {'MSD': MSD, 'MPD': MPD, 'STD': STD}
153
154
155 def process_dem(DEM, OPT=None, row_step=1, col_step=
    1):
156     """
157     Elabora la matrice DEM per una LASTRA PIANA (es
    . Granito),
158     eseguendo una scansione a griglia ortogonale
    standard (Righe e Colonne)
159     centrata rispetto al cuore della matrice.
160     Tollera variazioni di quota millimetriche,
    microscopiche o negative tipiche del granito.
161
162     :param DEM: Matrice 2D (NumPy array) contenente
    le quote della lastra in millimetri.
163     :param OPT: Dizionario delle opzioni di calcolo.
164     :param row_step: Passo di campionamento delle
    righe (1 = analizza ogni riga).
165     :param col_step: Passo di campionamento delle
    colonne (1 = analizza ogni colonna).
166     :return: Dizionario contenente le medie globali
    stimate di MPD ed ETD della lastra.
167     """
168     # Inizializzazione delle opzioni di default se
    non fornite
169     if OPT is None:
170         OPT = {'profile_dx': 0.1, 'spot_measurement'
    : True, 'calculate_EDT': True}
171
172     dx = OPT.get('profile_dx', 0.1) # Risoluzione
    spaziale del pixel in mm
173     calculate_EDT = OPT.get('calculate_EDT', True)
    # Flag per abilitare il calcolo dell'ETD norma ISO
174     rows, cols = DEM.shape
175
176     # Individua il centro geometrico (in pixel)
    della lastra piana

```

```

177     center_r, center_c = rows // 2, cols // 2
178
179     # Calcola quanti pixel servono a destra e a
    sinistra del centro per fare 50+50 = 100 mm totali
180     punti_semi_profilo = int(50 / dx)
181
182     mpd_rows = []
183
184     # -----
185     # 1. SCANSIONE ORIZZONTALE (ANALISI DELLE RIGHE)
186     # -----
187     # Campiona una fascia centrale di 100 righe
    distribuite attorno al centro geometrico della
    lastra
188     for i in range(center_r - 50, center_r + 50,
    row_step):
189         if 0 <= i < rows:
190             # Ritaglia il segmento lineare da 100 mm
    centrato rispetto alla colonna centrale
191             profile = DEM[i, center_c -
    punti_semi_profilo: center_c + punti_semi_profilo]
192             # Controllo geometrico: assicura che il
    profilo estratto contenga l'esatto numero di pixel
    richiesti
193             if len(profile) < (punti_semi_profilo *
    2):
194                 continue
195
196             # Calcola l'MPD del singolo profilo
    lineare mediante la funzione standard ISO
197             res = mean_profile_depth(profile, OPT)
198             if not np.isnan(res['MPD']):
199                 mpd_rows.append(res['MPD'])
200
201     mpd_cols = []
202
203     # -----

```

```

202 -----
203     # 2. SCANSIONE VERTICALE (ANALISI DELLE COLONNE)
204
205     # -----
206     # Campiona una fascia centrale di 100 colonne
207     # distribuite attorno al centro geometrico della
208     # lastra
209     for j in range(center_c - 50, center_c + 50,
210                    col_step):
211         if 0 <= j < cols:
212             # Ritaglia il profilo lineare verticale
213             # da 100 mm centrato rispetto alla riga centrale
214             profile = DEM[center_r -
215                          punti_semi_profilo: center_r + punti_semi_profilo, j
216                          ]
217
218             # Controllo geometrico di consistenza
219             # della lunghezza del segmento
220             if len(profile) < (punti_semi_profilo *
221                               2):
222                 continue
223
224             # Calcola l'MPD del profilo verticale
225             res = mean_profile_depth(profile, OPT)
226             if not np.isnan(res['MPD']):
227                 mpd_cols.append(res['MPD'])
228
229     # -----
230
231     # 3. ELABORAZIONE STATISTICA COMBINATA (RIGHE +
232     # COLONNE)
233
234     # -----
235
236     # Unisce tutti i profili validi estratti
237     # ortogonalmente sulla lastra
238     tutti_i_profili = mpd_rows + mpd_cols
239
240     # -----
241
242     # Calcola il valore medio globale dell'MPD della

```

```

226 lastra
227     global_mpd = np.mean(tutti_i_profili) if
    tutti_i_profili else np.nan
228
229     # Applica l'equazione empirica di conversione
    standard ISO (1.1 * MPD) per ottenere l'ETD
230     global_etd = 1.1 * global_mpd if not np.isnan(
    global_mpd) and calculate_EDT else np.nan
231
232     # Restituisce i dati organizzati
233     return {
234         'avg_MPD_rows': np.mean(mpd_rows) if
    mpd_rows else np.nan,
235         'avg_MPD_cols': np.mean(mpd_cols) if
    mpd_cols else np.nan,
236         'global_MPD': global_mpd,
237         'global_ETD': global_etd,
238         'total_profiles_analyzed': len(
    tutti_i_profili)
239     }
240
241
242 # =====
    =====
243 #                PANNELLO DI CONFIGURAZIONE GUIDATA
    (MAIN SCRIPT)
244 # =====
    =====
245 if __name__ == '__main__':
246
247     # Definizione del percorso assoluto o relativo
    del file DEM della lastra in granito (.txt)
248     percorso_file= r"C:\Users\vinny\Desktop\
    Tesi_Falcini\DATA PROGETTO PHYTON\per codice MPD\
    Lastra in Granito\kriging dopo SOR\matrice lastra in
    granito [zona di sabbia dopo SOR] con s=0,05 mm.txt
    "
249     print("Caricamento del file DEM in corso (
    operazione richiesta per matrici grandi)...")
250     try:
251         # Carica la matrice dei numeri isolati da

```

```
251 spazi bianchi nel formato NumPy Array ad alte
    prestazioni
252         dem_matrix = np.loadtxt(percorso_file)
253         print(f"-> File caricato! Dimensione matrice
    : {dem_matrix.shape[0]}x{dem_matrix.shape[1]} pixel
    .")
254     except Exception as e:
255         print(f"▲ Errore nel caricamento del file: {
    e}")
256         exit()
257
258     # IMPOSTAZIONE DEI SALTI DI CAMPIONAMENTO (
    DENSITÀ DI ANALISI SULLA GRIGLIA)
259     # Impostando a 1 analizziamo consecutivamente
    tutte le linee centrali (X e Y) per la massima
    precisione.
260     passo_righe = 3
261     passo_colonne = 2
262
263     # IMPOSTAZIONI METROLOGICHE DELL'ALGORITMO PER
    LASTRA IN GRANITO
264     opzioni_mpd = {
265         'profile_dx': 0.05, # PASSO DI SCANSIONE
    REALE (in mm) configurato durante la rasterizzazione
    su CloudCompare
266         'spot_measurement': True, # Attiva la
    soppressione della pendenza del provino (Slope
    Suppression)
267         'calculate_EDT': True, # Ordina all'
    algoritmo di calcolare il parametro ETD (1.1 * MPD)
268         'verbose': False # Se True, stamperebbe a
    console i log matematici intermedi di ogni riga
269     }
270
271     print("\nAvvio elaborazione MPD con filtro di
    campionamento centrale per lastre...")
272     # Esecuzione del calcolo principale sulla
    matrice caricata
273     risultati = process_dem(dem_matrix, OPT=
    opzioni_mpd, row_step=passo_righe, col_step=
    passo_colonne)
```

```

274
275
    # -----
    -----
276     # SCHERMATA FINALE DI STAMPA DEI RISULTATI
VALIDATI
277
    # -----
    -----
278     print("\n" + "=" * 45)
279     print("          RISULTATI DELL'ANALISI CORRETTA
        ")
280     print("=" * 45)
281     print(
282         f"Passo selezione profili      -> Righe: ogni
        {passo_righe} | Colonne: ogni {passo_colonne} |
        Config: Griglia Ortogonale")
283     print(f"Profili interni analizzati -> {
        risultati['total_profiles_analyzed']}")
284     print("-" * 45)
285     print(f"MPD Medio dei profili RIGA      : {
        risultati['avg_MPD_rows']:.4f} mm")
286     print(f"MPD Medio dei profili COLONNA: {
        risultati['avg_MPD_cols']:.4f} mm")
287     print("-" * 45)
288     print(f"MPD GLOBALE DELLA LASTRA      : {risultati
        ['global_MPD']:.4f} mm")
289     print(f"ETD STIMATO (1.1 x MPD)      : {risultati
        ['global_ETD']:.4f} mm")
290     print("=" * 45)
291     # See PyCharm help at https://www.jetbrains.com/help
        /pycharm/
292

```