

```

1
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 from scipy.ndimage import maximum_filter,
  gaussian_filter
6 import os
7
8
9 # =====
  =====
10 # 1. MOTORE DI CALCOLO MODIFICATO (CON OPZIONI DI
    GEOMETRIA)
11 # =====
    =====
12 def esegui_analisi_mtd(file_path, res, finestra,
    lab_ref, diametro_test=None, tipo_investigazione=None
    ):
13     """
14     Esegue la simulazione numerica avanzata del Sand
    Patch Test (UNI EN 13036-1)
15     supportando geometrie circolari, quadrate o sull'
    intera matrice.
16
17     Parametri:
18     file_path          : str    -> Percorso del file
    di testo (.txt) contenente la matrice altimetrica Z.
19     res                : float  -> Risoluzione
    spaziale della griglia (passo del raster) espressa in
    mm.
20     finestra           : float  -> Dimensione della
    maschera mobile per il filtro di massimo (mm).
21     lab_ref            : float  -> Valore di MTD
    ottenuto fisicamente in laboratorio (termine di
    confronto).
22     diametro_test      : float  -> Diametro del
    cerchio o Lato del quadrato (mm). Ignorato se '
    completa'.
23     tipo_investigazione : str    -> 'circolare', '
    quadrata' o 'completa'. Se None, si auto-configura.
24     """

```

```

25     try:
26         # --- GESTIONE DEI PARAMETRI DI
CONFIGURAZIONE DI INGRESSO ---
27         if tipo_investigazione is None:
28             if diametro_test is None:
29                 tipo_investigazione = 'completa'
30             else:
31                 tipo_investigazione = 'circolare'
32
33         if tipo_investigazione == 'completa':
34             diametro_test = None
35
36         print(f"Inizio elaborazione dati con
impostazione investigazione: '{tipo_investigazione.
upper()}'...")
37
38         # --- CARICAMENTO E PREPARAZIONE MATRICE (
preprocessing) ---
39         df = pd.read_csv(file_path, sep=r'\s+',
header=None, decimal='.', engine='python')
40         z = df.values * 1000 # Conversione in mm
41         z = z - np.nanmin(z) # Normalizzazione quota
a partire da Z = 0
42
43         # --- MODIFICA 1: FILTRO ANTI-RUMORE GLOBALE
---
44         # Livella eventuali picchi isolati anomali (
spike di campionamento) al 99.9° percentile
45         quota_max_accettabile = np.percentile(z, 99.9
)
46         z = np.minimum(z, quota_max_accettabile)
47
48         rows, cols = z.shape
49
50         # Generazione coordinate spaziali meshgrid
51         xx, yy = np.meshgrid(np.arange(cols) * res,
np.arange(rows) * res)
52         window_px = int(finestra / res)
53
54         x_centro = (cols * res) / 2.0
55         y_centro = (rows * res) / 2.0

```

```

56
57         # Vincolo di confinamento laterale fissa (10
mm) per stabilizzare la stesa numerica della sabbia
58         finestra_bordo_fissa = 10.0 # mm
59         window_bordo_px = max(2, int(
finestra_bordo_fissa / res))
60
61         # Sotto-funzione per l'involucro perimetrale
62         def calcola_involucro_bordo(profilo_z,
step_px):
63             n = len(profilo_z)
64             nodi_indici = list(range(0, n, step_px))
65             if len(nodi_indici) == 0 or nodi_indici[-
1] != n - 1:
66                 nodi_indici.append(n - 1)
67
68             nodi_valori = []
69             for i in range(len(nodi_indici)):
70                 if i == 0:
71                     sotto_profilo = profilo_z[0:
nodi_indici[i + 1]]
72                     elif i == len(nodi_indici) - 1:
73                         sotto_profilo = profilo_z[
nodi_indici[i - 1]:n]
74                     else:
75                         sotto_profilo = profilo_z[
nodi_indici[i - 1]:nodi_indici[i + 1]]
76
77                     if len(sotto_profilo) > 0:
78                         # --- MODIFICA 2: ATTENUAZIONE
INVOLUCRO DI BORDO ---
79                         # Cambiato da percentile 100 (
massimo assoluto) a 95 per ignorare i picchi del
Kriging sul perimetro
80                         val = np.percentile(sotto_profilo
, 100)
81                     else:
82                         val = np.percentile(profilo_z,
100)
83                     nodi_valori.append(val)
84

```

```

85         return np.interp(np.arange(n),
nodi_indici, nodi_valori)
86
87     # === CREAZIONE DELLE MASCHERE E CONDIZIONI
AL CONTORNO STRUTTURATE ===
88     z_confinato = z.copy()
89
90     if tipo_investigazione == 'completa':
91         # La maschera attiva coincide con tutta
la matrice spaziale disponibile
92         maschera = np.ones_like(z, dtype=bool)
93         # Nessun riempimento esterno necessario
, i bordi naturali della matrice confinano la sabbia
94
95     else:
96         if tipo_investigazione == 'circolare':
97             raggio_test = diametro_test / 2.0
98             distanze_dal_centro = np.sqrt((xx -
x_centro) ** 2 + (yy - y_centro) ** 2)
99             maschera = distanze_dal_centro <=
raggio_test
100
101             # Generazione dei punti discreti sul
bordo circolare
102             perimetro_px = int(2 * np.pi *
raggio_test / res)
103             angoli = np.linspace(0, 2 * np.pi,
perimetro_px, endpoint=False)
104             x_peri = x_centro + raggio_test * np
.cos(angoli)
105             y_peri = y_centro + raggio_test * np
.sin(angoli)
106
107         elif tipo_investigazione == 'quadrata':
108             lato = diametro_test
109             maschera = (xx >= x_centro - lato /
2) & (xx <= x_centro + lato / 2) & \
110             (yy >= y_centro - lato /
2) & (yy <= y_centro + lato / 2)
111
112             # Generazione dei punti discreti sul

```

```

112 bordo quadrato tramite proiezione radiale dei raggi
113         perimetro_px = int(4 * lato / res)
114         angoli = np.linspace(0, 2 * np.pi,
perimetro_px, endpoint=False)
115         r_peri = np.zeros_like(angoli)
116         for i, a in enumerate(angoli):
117             cos_a = np.cos(a)
118             sin_a = np.sin(a)
119             if abs(cos_a) > abs(sin_a):
120                 r_peri[i] = (lato / 2.0) /
abs(cos_a)
121             else:
122                 r_peri[i] = (lato / 2.0) /
abs(sin_a)
123         x_peri = x_centro + r_peri * np.cos(
angoli)
124         y_peri = y_centro + r_peri * np.sin(
angoli)
125
126         # Estrazione e calcolo del confinamento
perimetrale per i casi geometrici definiti
127         cols_peri = np.clip(np.round(x_peri /
res).astype(int), 0, cols - 1)
128         rows_peri = np.clip(np.round(y_peri /
res).astype(int), 0, rows - 1)
129
130         profilo_perimetro_reale = z[rows_peri,
cols_peri]
131         profilo_perimetro_confinato =
calcola_inviluppo_bordo(profilo_perimetro_reale,
window_bordo_px)
132
133         z_confinato[rows_peri, cols_peri] =
profilo_perimetro_confinato
134
135         # Estensione radiale isotropa per
bloccare matematicamente il deflusso esterno della
sabbia
136         angoli_matrice = np.arctan2(yy -
y_centro, xx - x_centro)
137         angoli_matrice[angoli_matrice < 0] += 2

```

```

137 * np.pi
138         indici_angoli = np.round(angoli_matrice
    / (2 * np.pi) * perimetro_px).astype(int) %
    perimetro_px
139         z_confinato[~maschera] =
    profilo_perimetro_confinato[indici_angoli[~maschera
    ]]
140
141         # --- APPLICAZIONE DEI FILTRI DI STRATO (
    Morfologia + Assestamento Gaussiano) ---
142         sabbia_rigida = maximum_filter(z_confinato,
    size=window_px, mode='nearest')
143
144         DIMENSIONE_GRANELLO_SABBIA = 0.20 # mm
145         sigma_assestamento =
    DIMENSIONE_GRANELLO_SABBIA / res
146         sabbia = gaussian_filter(sabbia_rigida,
    sigma=sigma_assestamento)
147         sabbia = np.maximum(sabbia, z) #
    Interpenetrazione fisica (condizione di appoggio)
148
149         # === CALCOLO METRICHE MATRICIALI SULL'AREA
    DI INDAGINE SELEZIONATA ===
150         vuoti = sabbia - z
151         vuoti_attivi = vuoti[maschera]
152         mtd = np.mean(vuoti_attivi)
153
154         punti_contatto_3d = np.isclose(sabbia, z,
    atol=res * 2)
155         punti_contatto_attivi = punti_contatto_3d &
    maschera
156         num_massimi_totale = np.sum(
    punti_contatto_attivi)
157
158         # === OUTPUT REQUISITI IN TERMINALE ===
159         print(f"\n{'=' * 60}")
160         print(f"          RISULTATI ANALISI
    MORFOLOGICA ({tipo_investigazione.upper()}")
161         print(f"{'-' * 60}")
162         if tipo_investigazione == 'completa':
163             print(f" Area di Analisi

```

```

163 :                               INTEREZZA DELLA MATRICE")
164         elif tipo_investigazione == 'quadrata':
165             print(f" Lato Quadrato Area di Analisi
:             {diametro_test:.2f} mm")
166         else:
167             print(f" Diametro Area di Analisi
:             {diametro_test:.2f} mm")
168             print(f" MTD RISULTATO AREA ATTIVA
:             {mtd:.4f} mm")
169             print(f" NUMERO PUNTI DI MASSIMO (NELL'AREA
): {num_massimi_totale} px")
170             print(f" Area di contatto effettiva
calcolata: {(num_massimi_totale * (res ** 2)):.2f}
mm2")
171             print(f"{'-' * 60}")
172             print(f" Scostamento dal riferimento di Lab
: {mtd - lab_ref:.4f} mm")
173             print(f"{'=' * 60}\nGenerazione dei report
grafici in corso...")
174
175
176         # =====
177         # === REPORT GRAFICI ===
178
179         # =====
180
181         # 1. Grafico Condizioni al Contorno (Solo
per Circolare e Quadrata)
182         if tipo_investigazione != 'completa':
183             sabbia_visualizzazione_bordi = sabbia.
copy()
184             sabbia_visualizzazione_bordi[rows_peri,
cols_peri] = profilo_perimetro_confinato
185             z_reale_tot = profilo_perimetro_reale
186             z_inviluppo_tot =
sabbia_visualizzazione_bordi[rows_peri, cols_peri]
187
188             fig1, ax1 = plt.subplots(figsize=(15, 5
))

```

```

187
188         if tipo_investigazione == 'circolare':
189             fig1.suptitle(
190                 fr"Condizioni al Contorno:
Sviluppo della Circonferenza di Indagine ( $\Phi$  = {
diametro_test:.1f} mm)",
191                 fontsize=14, fontweight='bold')
192                 sviluppo_x = angoli * raggio_test
193                 ax1.set_xlabel('Sviluppo Circolare
Progressivo dell\'Arco (mm)')
194                 labels_settori = ['Settore I (0°-90
°)', 'Settore II (90°-180°)', 'Settore III (180°-270
°)',
195                                     'Settore IV (270°-
360°)']
196         else:
197             fig1.suptitle(
198                 f"Condizioni al Contorno:
Sviluppo del Perimetro Quadrato di Indagine (Lato =
{diametro_test:.1f} mm)",
199                 fontsize=14, fontweight='bold')
200                 sviluppo_x = np.linspace(0, 4 *
diametro_test, perimetro_px, endpoint=False)
201                 ax1.set_xlabel('Sviluppo Perimetrale
Progressivo del Quadrato (mm)')
202                 labels_settori = ['Lato 1 (Sud)', '
Lato 2 (Est)', 'Lato 3 (Nord)', 'Lato 4 (Ovest)']
203
204                 ax1.plot(sviluppo_x, z_reale_tot, label=
'Profilo Reale sul Perimetro', color='gray', alpha=0
.8)
205                 ax1.plot(sviluppo_x, z_inviluppo_tot,
label='Velo di Sabbia Confinato', color='red',
linestyle='--')
206
207                 c_len = sviluppo_x[-1] + (sviluppo_x[1
] - sviluppo_x[0])
208                 ax1.axvline(c_len * 0.25, color='black'
, linestyle=':', alpha=0.6)
209                 ax1.axvline(c_len * 0.50, color='black'
, linestyle=':', alpha=0.6)

```

```

210             ax1.axvline(c_len * 0.75, color='black'
, linestyle=':', alpha=0.6)
211
212             ymin, ymax = np.min(z_reale_tot), np.max
(z_reale_tot)
213             y_pos = ymin + (ymax - ymin) * 0.85
214             ax1.text(c_len * 0.125, y_pos,
labels_settori[0], ha='center', fontsize=10,
fontweight='bold', color='blue')
215             ax1.text(c_len * 0.375, y_pos,
labels_settori[1], ha='center', fontsize=10,
fontweight='bold', color='blue')
216             ax1.text(c_len * 0.625, y_pos,
labels_settori[2], ha='center', fontsize=10,
fontweight='bold', color='blue')
217             ax1.text(c_len * 0.875, y_pos,
labels_settori[3], ha='center', fontsize=10,
fontweight='bold', color='blue')
218
219             ax1.set_ylabel('Quota Z (mm)')
220             ax1.legend(loc='upper right')
221             ax1.grid(True, alpha=0.3)
222             plt.tight_layout()
223
224             # 2. Sezione Trasversale Centralizzata
225             riga_ottimale = rows // 2
226             x_line = xx[riga_ottimale, :]
227             z_line = z[riga_ottimale, :]
228             s_line = sabbia[riga_ottimale, :]
229             mask_line = maschera[riga_ottimale, :]
230             contatti = np.isclose(z_line, s_line, atol=
res * 2) & mask_line
231
232             fig2, (ax2a, ax2b) = plt.subplots(2, 1,
figsize=(14, 10))
233             fig2.suptitle(f"Sezione Trasversale e Zoom
Modello - Riga Centrale n° {riga_ottimale}",
fontsize=14,
234                             fontweight='bold')
235
236             ax2a.plot(x_line, z_line, label='Profilo

```

```

236 Superficie (Sassi)', color='black', alpha=0.5)
237         ax2a.plot(x_line, s_line, color='darkorange'
, linewidth=1.5)
238         ax2a.fill_between(x_line, z_line, s_line,
where=mask_line, color='orange', alpha=0.4,
239         label='Volume Sabbia di
Test')
240         ax2a.fill_between(x_line, z_line, s_line,
where=~mask_line, color='gray', alpha=0.15,
241         label='Volume Esterno al
Test')
242
243         limiti_x = x_line[mask_line]
244         if len(limiti_x) > 0 and tipo_investigazione
!= 'completa':
245             ax2a.axvline(limiti_x[0], color='blue',
linestyle=':', linewidth=1.5, label='Confine Area
Indagine')
246             ax2a.axvline(limiti_x[-1], color='blue'
, linestyle=':', linewidth=1.5)
247
248         if np.any(contatti):
249             ax2a.scatter(x_line[contatti], z_line[
contatti], color='red', s=30, label='Punti di
Contatto Reale',
250             zorder=5)
251         ax2a.set_title("Sezione Completa (Evidenza
grafica dell'area attiva di analisi)")
252         ax2a.set_xlabel("X (mm)")
253         ax2a.set_ylabel("Z (mm)")
254         ax2a.legend()
255         ax2a.grid(True, alpha=0.3)
256
257         # Plot Zoom Centralizzato
258         c_mid = cols // 2
259         zoom_span = 50
260         z_start, z_end = max(0, c_mid - zoom_span),
min(cols, c_mid + zoom_span)
261         ax2b.step(x_line[z_start:z_end], z_line[
z_start:z_end], where='mid', label='Profilo Sassi (
Step DEM)',

```

```

262             color='black', linewidth=1.2)
263         ax2b.step(x_line[z_start:z_end], s_line[
            z_start:z_end], where='mid', label='Velo di Sabbia'
            , color='darkorange',
264             linewidth=1.5)
265         ax2b.fill_between(x_line[z_start:z_end],
            z_line[z_start:z_end], s_line[z_start:z_end], step='
            mid',
266             color='orange', alpha=0.4
            , label='Riempimento Vuoti')
267
268         c_zoom = contatti[z_start:z_end]
269         if np.any(c_zoom):
270             ax2b.scatter(x_line[z_start:z_end][
            c_zoom], z_line[z_start:z_end][c_zoom], color='red'
            , s=45, zorder=5,
271                 label='Punti di Appoggio')
272         ax2b.set_title("Zoom Dettagliato Centrale (
            Evidenza a gradini dei punti di appoggio)")
273         ax2b.set_xlabel("X (mm)")
274         ax2b.set_ylabel("Z (mm)")
275         ax2b.legend()
276         ax2b.grid(True, alpha=0.3)
277         plt.tight_layout()
278
279         # =====
            =====
280         # 3. NUOVA RAPPRESENTAZIONE TRIDIMENSIONALE
            A SUPERFICI CONTINUATIVE
281
            # =====
            =====
282         fig3 = plt.figure(figsize=(12, 9))
283         ax3 = fig3.add_subplot(111, projection='3d')
284         fig3.suptitle("Modello 3D Continuous delle
            Superfici Interagenti", fontsize=14, fontweight='
            bold')
285
286         # Calcolo di uno stride (passo) adattivo per
            mantenere fluido il rendering matplotlib su matrici

```

```

286 grandi
287         stride_dinamico = max(1, max(rows, cols) //
120)
288
289         # Plot 3D della Matrice Altimetrica (Provino
in conglomerato)
290         surf_z = ax3.plot_surface(xx[:,
stride_dinamico, ::stride_dinamico],
291                                   yy[:,
stride_dinamico, ::stride_dinamico],
292                                   z[:,
stride_dinamico, ::stride_dinamico],
293                                   cmap='gray', alpha
=0.8, edgecolor='none')
294
295         # Plot 3D della Superficie di Ricoprimento (
Velo di Sabbia calcolato dal codice)
296         surf_sabbia = ax3.plot_surface(xx[:,
stride_dinamico, ::stride_dinamico],
297                                       yy[:,
stride_dinamico, ::stride_dinamico],
298                                       sabbia[:,
stride_dinamico, ::stride_dinamico],
299                                       color='orange
', alpha=0.45, edgecolor='none')
300
301         # Creazione di proxy per la corretta
associazione in legenda
302         import matplotlib.lines as mlines
303         proxy_matrice = mlines.Line2D([], [], color=
'gray', marker='s', markersize=10, linestyle='None',
304                                       label='
Superficie Matrice DEM (Z)')
305         proxy_sabbia = mlines.Line2D([], [], color='
orange', marker='s', markersize=10, linestyle='None'
, alpha=0.6,
306                                       label='
Superficie Ricoprimento (Sabbia)')
307         ax3.legend(handles=[proxy_matrice,
proxy_sabbia], loc='upper right')
308

```

```

309         ax3.set_xlabel('X (mm)')
310         ax3.set_ylabel('Y (mm)')
311         ax3.set_zlabel('Z (mm)')
312         ax3.view_init(elev=35, azimuth=45)
313
314         plt.show()
315
316     except Exception as e:
317         print(f"Errore durante l'esecuzione: {e}")
318
319
320 # =====
    =====
321 # 2. AREA DI CONFIGURAZIONE (PANNELLO DI CONTROLLO
    AGGIORNATO)
322 # =====
    =====
323 if __name__ == "__main__":
324     # --- INPUT DATI ---
325     FILE_DA_ELABORARE =
326     RISOLUZIONE_RASTER = # mm/pixel
327
328     # --- CALIBRAZIONE FILTRO ---
329     LUNGHEZZA_MASCHERA = # Dimensione piattello
    mobile (mm)
330
331
    # =====
    =====
332     # NUOVA IMPOSTAZIONE SELEZIONE DINAMICA
    INVESTIGAZIONE
333
    # =====
    =====
334     TIPO_INVESTIGAZIONE = 'circolare' # Cambia in '
    circolare', 'quadrata' o 'completa'
335     DIAMETRO_TEST_LAB = # mm (Diametro o Lato a
    seconda della scelta sopra)
336
337     # --- RIFERIMENTI DI LABORATORIO ---
338     VALORE_SAND_PATCH = # mm

```

```
339
340     # --- ESECUZIONE CORRETTA ---
341     esegui_analisi_mtd(
342         file_path=FILE_DA_ELABORARE,
343         res=RISOLUZIONE_RASTER,
344         finestra=LUNGHEZZA_MASCHERA,
345         lab_ref=VALORE_SAND_PATCH,
346         diametro_test=DIAMETRO_TEST_LAB,
347         tipo_investigazione=TIPO_INVESTIGAZIONE
348     )
```