Modélisation hydraulique d'un tronçon de la Lesse en vue d'y établir un seuil de hauteur d'eau au passage des kayaks.

Auteur : Bruyère, Tobias
Promoteur(s) : Michez, Adrien; Dufrêne, Marc
Faculté : Gembloux Agro-Bio Tech (GxABT)
Diplôme : Master en bioingénieur : sciences et technologies de l'environnement, à finalité spécialisée
Année académique : 2024-2025
URI/URL : http://hdl.handle.net/2268.2/24123

Annexe :

Annexe 1 :

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 24 11:20:28 2025

@author: 01tob
"""

import seamsh
from seamsh.geometry import CurveType
import numpy as np
from osgeo import osr

# ================================
# 1. Création du domaine et projection
# ================================

domain_srs = osr.SpatialReference()
domain_srs.ImportFromEPSG(31370)  # Lambert belge 1972

domain = seamsh.geometry.Domain(domain_srs)

# ================================
# 2. Chargement des courbes depuis des shapefiles
# ================================

# Chargement des polylignes pour définir les contours
# domain.add_boundary_curves_shp(r"C:\master 2\TFE\watlab\entrainement\qgis\lit_majeur.shp", "type", CurveType.POLYLINE)
domain.add_boundary_curves_shp(r"C:\master 2\TFE\tobias\mesh\lit_mineur.shp", "type", CurveType.POLYLINE)

#domain.add_interior_curves_shp(r"C:\master 2\TFE\watlab\entrainement\qgis\lit_mineur.shp", "type", CurveType.POLYLINE)

# ================================
# 3. Extraction des valeurs bathymétriques depuis un raster
# ================================

# bath_field = seamsh.field.Raster(r"C:\master 2\TFE\watlab\entrainement\qgis\raster_fin.tif")
bath_field = seamsh.field.Raster(r"C:\master 2\TFE\tobias\mesh\raster_mesh.tif")
```

```
def mesh_size (x,projection):
    bath=bath_field(x,projection)
    return bath



# ================================
# 4. Génération du maillage
# ================================

output_srs = osr.SpatialReference()
output_srs.ImportFromEPSG(31370)  # Lambert belge 1972

seamsh.gmsh.mesh(domain, "maille_2m.msh", mesh_size,
intermediate_file_name="debug", output_srs=output_srs)

# ================================
# 5. Reprojection du maillage
# ================================

seamsh.gmsh.reproject("maille_2m.msh", domain_srs, "maille_2m.msh", output_srs)

# ================================
# 6. Conversion du maillage en format SIG
# ================================

seamsh.gmsh.convert_to_gis("maille_2m.msh", output_srs, "maille_2m.gpkg")
```

annexe 2:

```
Created on Thu Apr 24 10:44:21 2025

@author: 01tob
"""

from watlab import hydroflow
import numpy as np

Manning=np.arange(0.020,0.041,0.001 )
q_in=np.arange(4.2,8,0.2)
for i in range (len(q_in)):
    k=0.032
    q = 1.83
```

```
    mesh_path = "maille_2m.msh"

    mesh = hydroflow.Mesh(mesh_path)
    mesh.meshchecking = True
    mesh.set_nodes_elevation_from_tif("interpolation.tif")

    model = hydroflow.HydroflowModel(mesh)

    model.name = f"Lesse_test_q_in_{q:.2f}"


    model.Cfl_number = 0.95

    model.set_friction_coefficient('domain',k)

    model.set_transmissive_boundaries("q_out")
    model.set_boundary_water_discharge("q_in", q)

model.set_wall_boundaries(["rg4","rg5","rg6","rg7","rg8","rg9","rg11","rg12","rg13","rg14
","rg15","rg16","rg17","rd17","rd16","rd15","rd14","rd13","rd12","rd10","rd9","rd8","rd7",
"rd6","rd5","rd4","rd3","rd2","rd1","rg1","rg2"])

    model.set_discharge_measurement_section("q_out", time_step=600)

    model.starting_time=0
    model.ending_time = 1801
    model.set_picture_times(pic_array=np.arange(0,1801,1800))

    model.export.input_folder_name="input-cells"
    model.export.output_folder_name = f"output-cells-q-{q:.3f}"

    model.export_data()
    model.solve()
```

annexe 3:

```
from qgis.core import (
    QgsProject,
    QgsSpatialIndex,
    QgsPointXY,
    QgsGeometry
)
import numpy as np
from collections import defaultdict

# === Chargement des couches ===
```

```
points_layer = QgsProject.instance().mapLayersByName('pic_1800_00')[0]
transects_layer = QgsProject.instance().mapLayersByName('transect_kayak')[0]

tolerance = 1  # m
depth_threshold = 0.13  # m

# Index spatial
transect_index = QgsSpatialIndex(transects_layer.getFeatures())

# Dictionnaires
transect_projections = defaultdict(list)
transect_depths = defaultdict(list)

# === Traitement des points ===
for point_feat in points_layer.getFeatures():
    point_geom = point_feat.geometry()
    point_z = point_geom.constGet().z()

    if point_z is None or point_z < depth_threshold:
        continue

    candidate_ids = transect_index.intersects(point_geom.buffer(tolerance,
5).boundingBox())

    for fid in candidate_ids:
        transect_feat = transects_layer.getFeature(fid)
        transect_geom = transect_feat.geometry()

        if point_geom.distance(transect_geom) <= tolerance:
            _, closest_point, _, _ =
transect_geom.closestSegmentWithContext(point_geom.asPoint())

            transect_id = transect_feat["id"]  # ⚠ utilise bien le champ "id"
            transect_projections[transect_id].append(QgsPointXY(closest_point))
            transect_depths[transect_id].append(point_z)
            break

# === Affichage des résultats par transect ===
transect_widths = {}
for tid, points in transect_projections.items():
    if len(points) < 2:
        print(f"Transect {tid} : Pas assez de points")
        continue

    max_dist = max([
        points[i].distance(points[j])
```

```
            for i in range(len(points))
            for j in range(i + 1, len(points))
        ])
        mean_depth = np.mean(transect_depths[tid]) if transect_depths[tid] else float('nan')

        transect_widths[tid] = max_dist
        print(f"Transect {tid} → Largeur navigable (propre) : {max_dist:.2f} m – Profondeur
    moyenne : {mean_depth:.2f} m")

    # === Groupes de transects ===
    group_ids = {
        "Lignes 1 à 10": [1, 2, 3, 4,5,6,7,8,9,10],
        "Lignes 11 à 20": [11,12,13,14,15,16,17,18,19,20],
        "Lignes 21 à 30": [21,22,23,24,25,26,27,28,29,30]
    }

    # === Profondeur moyenne par groupe ===
    for group, ids in group_ids.items():
        all_depths = []
        for i in ids:
            if i in transect_depths:
                all_depths.extend(transect_depths[i])
        if all_depths:
            print(f"{group} → Profondeur moyenne : {np.mean(all_depths):.2f} m")
        else:
            print(f"{group} → Aucun point trouvé")

    # === Largeur moyenne par groupe ===
    for group, ids in group_ids.items():
        if all(i in transect_widths for i in ids):
            widths = [transect_widths[i] for i in ids]
            print(f"{group} → Largeur moyenne : {np.mean(widths):.2f} m")
        else:
            print(f"{group} → Aucune largeur calculable (au moins un transect incomplet)")
```

annexe 4 :

```
    # -*- coding: utf-8 -*-
    import pandas as pd
    import numpy as np
    from scipy.interpolate import griddata
    import matplotlib.pyplot as plt
    from matplotlib.colors import ListedColormap
    import rasterio
    from rasterio.transform import from_origin
```

```
fi = pd.read_csv(r"D:/TFE/figure/result-20m/variation_debit-20m/1.8300.csv",
encoding='latin1')

# 2. Conversion sécurisée en float (valeurs invalides deviennent NaN)
cols_to_check = ["x", "y", "h", "qx", "qy", "field_7"]
for col in cols_to_check:
    fi[col] = pd.to_numeric(fi[col], errors='coerce')

# Après conversion en float et avant toute interpolation
fi = fi[
    (fi["x"] > 189000) & (fi["x"] < 189600) &
    (fi["y"] > 100850) & (fi["y"] < 101010)
]

x =np.array( fi["x"])
y =np.array( fi["y"])
h = np.array(fi["qx"])            # Vérifie si c'est bien la colonne de hauteur
qx =np.array( fi["qy"])             # Vérifie si ce n'est pas une inversion
qy = np.array(fi["field_7"])

# 2. Grille régulière
grid_yi, grid_xi = np.mgrid[y.min():y.max():200j, x.min():x.max():200j]

# 3. Interpolation
grid_h = griddata((x, y), h, (grid_xi, grid_yi), method='linear')
grid_qx = griddata((x, y), qx, (grid_xi, grid_yi), method='linear')
grid_qy = griddata((x, y), qy, (grid_xi, grid_yi), method='linear')

# 3bis. Inversion Y pour affichage correct
grid_h_corr = np.flipud(grid_h)
grid_qx_corr = np.flipud(grid_qx)
grid_qy_corr = np.flipud(grid_qy)

# 3ter. Calcul de la vitesse totale
grid_speed = np.sqrt(grid_qx_corr**2 + grid_qy_corr**2)

# 4. Sauvegarde en raster
pixel_size_x = (x.max() - x.min()) / 200
pixel_size_y = (y.max() - y.min()) / 200
transform = from_origin(x.min(), y.max(), pixel_size_x, pixel_size_y)

def save_raster(filename, data, transform):
    with rasterio.open(filename, 'w', driver='GTiff',
                       height=data.shape[0], width=data.shape[1],
                       count=1, dtype=data.dtype,
                       crs='EPSG:31370',
                       transform=transform) as dst:
```

```
        dst.write(data, 1)

save_raster("hauteur_eau.tif", grid_h_corr, transform)
save_raster("vitesse_totale.tif", grid_speed, transform)

# 5. Carte rouge avec 0 = noir et échelle max 4 m
red_cmap = plt.cm.Reds
red_colors = red_cmap(np.linspace(0, 1, 256))
red_colors[0] = [0, 0, 0, 1]  # noir pour 0
custom_red_cmap = ListedColormap(red_colors)

plt.figure(figsize=(8, 6))
plt.imshow(grid_h_corr, origin='upper',
           extent=(x.min(), x.max(), y.min(), y.max()),
           cmap=custom_red_cmap, vmin=0, vmax=4)  # Limites d'échelle ajoutées ici
plt.colorbar(label="Hauteur d'eau (m)")
plt.xlabel("x")
plt.ylabel("y")
plt.tight_layout()
plt.savefig("hauteur_rouge_zero_noir.png", dpi=300)
plt.show()


# 6. Carte binaire orange/bleu + noir pour 0
binary_rgb = np.zeros((*grid_h_corr.shape, 3), dtype=np.uint8)
mask_orange = grid_h_corr > 0.13
mask_blue = (grid_h_corr <= 0.13) & (grid_h_corr > 0)
mask_black = grid_h_corr == 0

binary_rgb[mask_orange] = [255, 140, 0]    # Orange
binary_rgb[mask_blue] = [30, 144, 255]     # Bleu
binary_rgb[mask_black] = [0, 0, 0]         # Noir

percent_orange = 100 * np.sum(mask_orange) / (np.sum(mask_orange) +
np.sum(mask_blue))

plt.figure(figsize=(8, 6))
plt.imshow(binary_rgb, origin='upper', extent=(x.min(), x.max(), y.min(), y.max()))
plt.xlabel("x")
plt.ylabel("y")
plt.tight_layout()
plt.savefig("hauteur_orange_bleu.png", dpi=300)
plt.show()

# éviter la division par zéro ou les NaN
grid_speed_real = np.sqrt(grid_qx_corr**2 + grid_qy_corr**2) / np.where(grid_h_corr > 0,
grid_h_corr, np.nan)
```

```
# 7. Vitesse totale uniquement
plt.figure(figsize=(8, 6))
plt.imshow(grid_speed_real, origin='upper', extent=(x.min(), x.max(), y.min(), y.max()))
plt.colorbar(label="Vitesse (m/s)")
plt.xlabel("x")
plt.ylabel("y")
plt.tight_layout()
plt.savefig("vitesse_reelle.png", dpi=300)
plt.show()
# 8. Histogramme des hauteurs d'eau (en %)
h_vals = grid_h_corr[~np.isnan(grid_h_corr)].flatten()
n_total = len(h_vals)
weights = np.ones_like(h_vals) / n_total * 100

plt.figure(figsize=(8, 6))
plt.ylim(0, 50)

plt.hist(h_vals, bins=50, weights=weights, color='skyblue', edgecolor='black')
plt.axvline(x=0.13, color='red', linestyle='--', linewidth=2, label='Seuil minimum (13 cm)')
plt.xlabel("Hauteur d'eau (m)")
plt.ylabel("Fréquence (%)")
plt.legend()
plt.tight_layout()
plt.savefig("histogramme_hauteur_eau_pourcentage.png", dpi=300)
plt.show()
# Calcul des pourcentages
total_valid = np.sum(mask_orange) + np.sum(mask_blue) + np.sum(mask_black)

percent_orange = 100 * np.sum(mask_orange) / total_valid
percent_blue = 100 * np.sum(mask_blue) / total_valid
percent_black = 100 * np.sum(mask_black) / total_valid

print(f"Zone > 13 cm : {percent_orange:.2f} %")
print(f"Zone entre 0 et 13 cm : {percent_blue:.2f} %")
print(f"Zone à sec (0 m) : {percent_black:.2f} %")
```

annexe 5 :

```
from qgis.core import (
    QgsProject, QgsGeometry, QgsPointXY, QgsFeature, QgsDistanceArea
)
import matplotlib.pyplot as plt
import numpy as np

# Récupérer les couches
```

```
points_layer = QgsProject.instance().mapLayersByName("points_en_dessous")[0]
traces_layer = QgsProject.instance().mapLayersByName("deux chemins")[0]

# Paramètres
espacement = 0.5  # en mètres
seuil_hauteur = 0.13  # 13 cm
buffer_distance = 2  # tolérance de recherche des points autour du tracé

# Distance calc avec CRS Lambert 1972
dist_calc = QgsDistanceArea()
dist_calc.setSourceCrs(traces_layer.crs(), QgsProject.instance().transformContext())
dist_calc.setEllipsoid('GRS80')  # ou autre selon besoin

# Fonction pour interpoler les points à intervalle régulier
def interpolate_along_line(geom, distance):
    total_length = geom.length()
    nb = int(total_length // distance) + 1
    points = [geom.interpolate(i * distance).asPoint() for i in range(nb)]
    return points

# Pour chaque tracé
for trace_feat in traces_layer.getFeatures():
    trace_geom = trace_feat.geometry()

    # 1. Échantillonner tous les 1.5 m
    sample_points = interpolate_along_line(trace_geom, espacement)

    distances = []
    hauteurs = []

    for i, pt in enumerate(sample_points):
        pt_geom = QgsGeometry.fromPointXY(QgsPointXY(pt))

        # Chercher les points dans un buffer autour du point (petite tolérance)
        nearby = points_layer.getFeatures(
            QgsGeometry.fromPointXY(QgsPointXY(pt)).buffer(buffer_distance,
5).boundingBox()
        )

        nearest_point = None
        nearest_dist = float("inf")
        nearest_qx = None

        for pf in nearby:
            p_geom = pf.geometry()
            dist = pt_geom.distance(p_geom)
            if dist < nearest_dist:
```

```
                nearest_dist = dist
                nearest_point = p_geom
                nearest_qx = pf["qx"]

            if nearest_qx is not None:
                distances.append(i * espacement)
                hauteurs.append(nearest_qx)

        if distances:
            # Tri par distance
            distances = np.array(distances)
            hauteurs = np.array(hauteurs)

            # Tracé
            plt.figure(figsize=(10, 5))
            plt.plot(distances, hauteurs, marker='o', color='blue', label='Hauteur mesurée')
            plt.axhline(y=seuil_hauteur, color='green', linestyle='--', label='Seuil 13 cm')
            plt.xlabel("Distance le long du tracé (m)")
            plt.ylabel("Hauteur d'eau (qx) en m")
            plt.title(f"Profil hauteur d'eau - Tracé {trace_feat.id()}")
            plt.grid(True)
            plt.legend()
            plt.tight_layout()
            plt.show()
```

annexe 6:

```
# -*- coding: utf-8 -*-

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import unicodedata
import calendar

# === 1. Chargement du fichier ===
df = pd.read_excel(r"D:\TFE\figure\debit_gendron\DGH_5991_Q_DayMean.xlsx")

# Normaliser les noms de colonnes
df.columns = df.columns.str.strip().str.lower()  # 'Timestamp' → 'timestamp', 'Value' →
'value'

# Convertir les dates et les débits
df["date"] = pd.to_datetime(df["timestamp"], format="%d-%m-%Y", errors="coerce")
df["debit"] = df["value"].astype(str).str.replace(",", ".").astype(float)
```

```python
# Nettoyage
df = df.dropna(subset=["date", "debit"])

# Colonnes utiles
df["annee"] = df["date"].dt.year
df["jour_annee"] = df["date"].dt.strftime('%m-%d')

# === 2. Q95 par année ===
q95_par_an = df.groupby("annee")["debit"].quantile(0.05)

# === 3. Statistiques globales ===
nb_jours_total = len(df)
nb_jours_sous_1_5 = (df["debit"] < 3).sum()
frequence = nb_jours_sous_1_5 / nb_jours_total
debit_moyen = df["debit"].mean()
q95_global = df["debit"].quantile(0.05)
q5_global = df["debit"].quantile(0.95)
debit_median = df["debit"].median()
debit_min = df["debit"].min()
debit_max = df["debit"].max()

print("=== Statistiques globales ===")
print(f"Nombre total de jours : {nb_jours_total}")
print(f"Nombre de jours avec débit < 1.5 m³/s : {nb_jours_sous_1_5}")
print(f"Fréquence : {frequence:.2%}")
print(f"Débit moyen : {debit_moyen:.2f} m³/s")
print(f"Médiane : {debit_median:.2f} m³/s")
print(f"Débit min : {debit_min:.2f} m³/s")
print(f"Débit max : {debit_max:.2f} m³/s")
print(f"Q95 global (5e percentile) : {q95_global:.2f} m³/s")
print(f"Q5 global (95e percentile) : {q5_global:.2f} m³/s")

# === 4. Q95 par année ===
plt.figure(figsize=(10, 5))
q95_par_an.plot(marker='o')
plt.title("Q95 annuel (débit journalier au 5e percentile)")
plt.xlabel("Année")
plt.ylabel("Q95 [m³/s]")
plt.grid(True)
plt.tight_layout()
plt.show()

# === 5. Moyenne par jour de l'année ===
moyenne_par_jour = df.groupby("jour_annee")["debit"].mean()

plt.figure(figsize=(14, 5))
```

```
plt.plot(moyenne_par_jour.index, moyenne_par_jour.values, label="Débit moyen journalier")
plt.axhline(3, color='red', linestyle='--', label="Seuil 1.5 m³/s")
plt.title("Débit moyen pour chaque jour de l'année (2000–2025)")
plt.xlabel("Jour de l'année (mois)")
plt.ylabel("Débit moyen [m³/s]")
mois_labels = [f"{str(m).zfill(2)}-15" for m in range(1, 13)]
plt.xticks(ticks=mois_labels, labels=calendar.month_abbr[1:], rotation=0)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# === 6. Fréquence des jours < 1.5 m³/s ===
df["est_sous_1_5"] = df["debit"] < 3
frequence_basse = df.groupby("jour_annee")["est_sous_1_5"].mean() * 100

plt.figure(figsize=(14, 4))
plt.plot(frequence_basse.index, frequence_basse.values, color='#1f77b4')
plt.ylim(0,75)
plt.title("Fréquence (%) des jours < 1.5 m³/s par jour de l'année")
plt.xlabel("Jour de l'année (mm-jj)")
plt.ylabel("Fréquence [%]")
plt.xticks(ticks=frequence_basse.index[::15], rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
# --- Paramètres de seuils cohérents partout ---
SEUIL_ROUGE = 1.5  # m³/s
SEUIL_VERT = 3.4   # m³/s

# (… ton code inchangé jusqu'au calcul de q95_par_an …)

# === 4. Q95 par année ===
plt.figure(figsize=(10, 5))
q95_par_an.plot(marker='o')
plt.title("Q95 annuel (débit journalier au 5e percentile)")
plt.xlabel("Année")
plt.ylabel("Q95 [m³/s]")

# Lignes de seuils demandées
plt.axhline(SEUIL_ROUGE, linestyle='--', linewidth=1.5, label=f"Seuil {SEUIL_ROUGE} m³/s", color='red')
plt.axhline(SEUIL_VERT,  linestyle='--', linewidth=1.5, label=f"Seuil {SEUIL_VERT} m³/s",  color='green')

plt.grid(True)
```

```
plt.legend()
plt.tight_layout()
plt.show()

# === 5. Moyenne par jour de l'année (toutes années) ===
moyenne_par_jour = df.groupby("jour_annee")["debit"].mean()

plt.figure(figsize=(14, 5))
plt.plot(moyenne_par_jour.index, moyenne_par_jour.values, label="Débit moyen journalier
(toutes années)")
plt.axhline(SEUIL_ROUGE, linestyle='--', label=f"Seuil {SEUIL_ROUGE} m³/s",
color='red')
plt.axhline(SEUIL_VERT,  linestyle='--', label=f"Seuil {SEUIL_VERT} m³/s",
color='green')
plt.title("Débit moyen pour chaque jour de l'année")
plt.xlabel("Jour de l'année (mois)")
plt.ylabel("Débit moyen [m³/s]")
mois_labels = [f"{str(m).zfill(2)}-15" for m in range(1, 13)]
plt.xticks(ticks=mois_labels, labels=calendar.month_abbr[1:], rotation=0)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# === 6. Fréquence des jours < 1,5 m³/s (corrigé) ===
df["est_sous_1_5"] = df["debit"] < SEUIL_ROUGE
frequence_basse = df.groupby("jour_annee")["est_sous_1_5"].mean() * 100

plt.figure(figsize=(14, 4))
plt.plot(frequence_basse.index, frequence_basse.values)
plt.ylim(0, 75)
plt.title(f"Fréquence (%) des jours < {SEUIL_ROUGE} m³/s par jour de l'année")
plt.xlabel("Jour de l'année (mm-jj)")
plt.ylabel("Fréquence [%]")
plt.xticks(ticks=frequence_basse.index[::15], rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()

# === 7. NOUVEAU — Débit moyen journalier des 25 dernières années en excluant les Q5
===
# Déterminer la fenêtre des 25 dernières années selon les données
annee_max = int(df["annee"].max())
annee_min_25 = annee_max - 24
df_25 = df[(df["annee"] >= annee_min_25) & (df["annee"] <= annee_max)].copy()

# Exclure les Q5 (95e percentile) AVANT la moyenne
```

```
q95_25 = df_25["debit"].quantile(0.95)
df_25_filtre = df_25[df_25["debit"] <= q95_25]

moyenne_par_jour_25 = df_25_filtre.groupby("jour_annee")["debit"].mean()

plt.figure(figsize=(14, 5))
plt.plot(moyenne_par_jour_25.index, moyenne_par_jour_25.values, label=f"Moyenne
journalière {annee_min_25}-{annee_max} (sans Q5)")
plt.axhline(SEUIL_ROUGE, linestyle='--', label=f"Seuil {SEUIL_ROUGE} m³/s",
color='red')
plt.axhline(SEUIL_VERT,  linestyle='--', label=f"Seuil {SEUIL_VERT} m³/s",
color='green')
plt.title(f"Débit moyen par jour (dernières 25 années, {annee_min_25}–{annee_max})\nQ5
exclus de la moyenne")
plt.xlabel("Jour de l'année (mois)")
plt.ylabel("Débit moyen [m³/s]")
mois_labels = [f"{str(m).zfill(2)}-15" for m in range(1, 13)]
plt.xticks(ticks=mois_labels, labels=calendar.month_abbr[1:], rotation=0)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# === 8. Comptages sous seuils — en MOYENNE PAR AN (pas de cumul) ===
def jours_moyen_par_an_sous(df_in, seuil):
    # nb de jours sous le seuil par année
    par_an = df_in.groupby("annee")["debit"].apply(lambda s: (s < seuil).sum())
    # moyenne annuelle
    return float(par_an.mean()), par_an

# Moyenne annuelle sur toute la série
moy_jours_sous_1_5_all, details_1_5_all = jours_moyen_par_an_sous(df,
SEUIL_ROUGE)
moy_jours_sous_3_4_all, details_3_4_all = jours_moyen_par_an_sous(df, SEUIL_VERT)

# Moyenne annuelle sur les 25 dernières années
moy_jours_sous_1_5_25, details_1_5_25 = jours_moyen_par_an_sous(df_25,
SEUIL_ROUGE)
moy_jours_sous_3_4_25, details_3_4_25 = jours_moyen_par_an_sous(df_25,
SEUIL_VERT)


# === Indicateur "climatologique" : nb de jours (mm-jj) dont la moyenne sur toutes les
années < seuil ===
moyenne_par_jour_all = df.groupby("jour_annee")["debit"].mean()

nb_jours_clim_sous_1_5 = int((moyenne_par_jour_all < SEUIL_ROUGE).sum())
```

```
nb_jours_clim_sous_3_4 = int((moyenne_par_jour_all < SEUIL_VERT).sum())

print("\n=== Jours sous seuils — MESURÉS vs CLIMATOLOGIE ===")
print(f"[Mesuré] Moyenne annuelle de jours < {SEUIL_ROUGE} m³/s =
{moy_jours_sous_1_5_all:.1f}")
print(f"[Mesuré] Moyenne annuelle de jours < {SEUIL_VERT} m³/s =
{moy_jours_sous_3_4_all:.1f}")
print(f"[Climatologie] Jours (mm-jj) dont la moyenne < {SEUIL_ROUGE} m³/s =
{nb_jours_clim_sous_1_5}")
print(f"[Climatologie] Jours (mm-jj) dont la moyenne < {SEUIL_VERT} m³/s =
{nb_jours_clim_sous_3_4}")
```

annexe 7:

Manning	RMSE	Sr
0.031	0.158860	NaN
0.032	0.158665	-0.021849
0.033	0.158644	-0.009610
0.034	0.158573	-0.016556
0.035	0.158489	-0.017343
0.036	0.158416	-0.018862
0.037	0.158323	-0.010909
0.038	0.158323	0.007942
0.039	0.158389	0.012042
0.040	0.158572	0.022319
0.041	0.158703	-0.0245231
0.042	0.158799	0.032918
0.043	0.158952	0.052483
0.044	0.159187	0.059032
0.045	0.159379	0.052697
0.046	0.159422	0.011246
0.047	0.159591	-40.283571
0.048	0.159964	0.097579
0.049	0.160242	0.101468
0.051	0.160959	0.115128
0.052	0.161332	0.144550
0.053	0.161856	0.135963
0.054	0.162162	0.070232
0.055	0.162278	0.149698
0.056	0.163045	0.192311
0.057	0.163398	0.134133
0.058	0.163814	0.148761

0.059	0.164238	NaN