Readme code d'analyse de profilométrie

Anaëlle Bonnay

12 juillet 2024

Table des matières

1 Introduction

Le code prend en entrée une image de profilométrie sous forme d'un fichier de données xyz et permet d'obtenir des données sur l'image en plusieurs étapes :

1. Le traitement de l'image, qui permet de rendre l'image exploitable par la suite. En sortie les images doivent être orientées de la même manière, sur un même plan et planes.
2. Le calcul des différentes variables pouvant être utiles à l'analyse du profil
3. Le tracé de différentes courbes et de l'image en profilométrie
4. Le stockage des données dans un fichier csv et celui des figures dons différents dossiers

2 Traitement de l'image

Les différentes fonctions permettant de traiter l'image sont situées dans le fichier **Traitement1imageRemovePlanPy.py**, chacune d'elles effectue une opération sur l'image.

2.1 ouverture_fichier(file_name, scale)

La fichier est d'abord ouvert avec la fonction **ouverture_fichier(file_name, scale)** qui prend en entrée le fichier et l'échelle (μm/pixels). Elle renvoie trois tableaux de dimension 1000×1000 : x qui contient les abscisses de chaque point de mesure, y, les ordonnées et z l'altitude des points. Ces tableaux vont permettre le traitement des données.

2.2 remplissage(z)

Cette fonction permet de combler les données manquantes sur l'image. Elle prend en entrée z et parcourt tous les indices du tableau. Si une valeur nulle est rencontrée (**pd.isna**), elle est remplacée par la moyenne des plus proches voisins au dessus et au dessous c'est eux qui ont une valeur plus proche de la valeur manquante, les sillons allant de haut en bas de l'image.

Ici on suppose que les sillons sont parallèles à l'axe y. Pour certaines images ce n'est pas complètement vrai, elles sont un peu tournées. Pour être plus exact dans les valeurs remplacées, il faudrait redresser l'image et retirer une surface avant de remplir le tableau. Mais ces deux fonctions ne gèrent pas les Nans. Pour améliorer le programme il faudrait trouver un moyen qui permet aux fonctions de fonctionner normalement avec des valeurs Nans.

Une autre piste d'amélioration serait de trouver un moyen de ne pas parcourir le tableau indice par indice, de complexité quadratique (n^2), donc un peu long.

2.3 retrait_pente(x,y,z,ordre)

Le but de cette fonction est d'aplanir la surface qui a pu être penchée ou recourbée lors des mesures. On ne veut mesurer que la rugosité et non la forme de la surface. Il y a donc deux fonctions pour cela :

2.3.1 retrait_pente1(z)

Elle calcule la moyenne mobile du tableau sur en prenant un intervalle entrée qui peut être modifié (**uniform_filter**(z,intervalle, mode). Cela permet d'obtenir un tableau ne possédant que les variations significatives, la meilleure condition au bord que j'ai trouvée, c'est 'nearest', qui donne la valeur du bord aux points manquants pour calculer la moyenne mobile. Il n'y avait pas de condition collant bien avec ce qui était recherché.

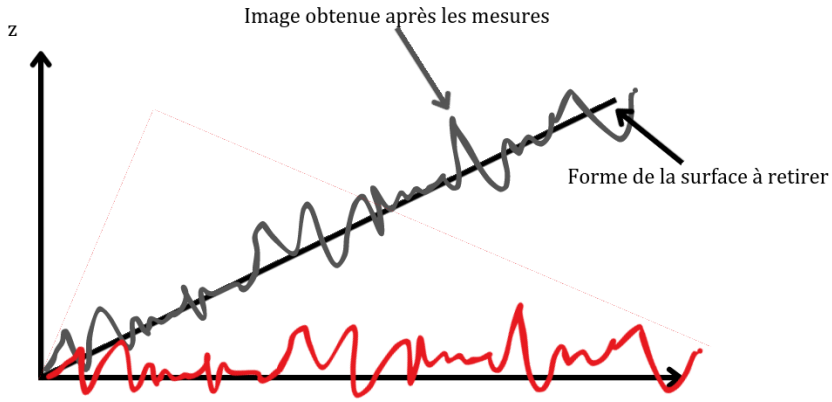

FIGURE 1 – Moyenne mobile

Théoriquement, après le retrait de la moyenne mobile sur z, on obtient une image plane, la pente et la courbure ont été retirées. Avec les images que nous avons ce n'est pas le cas, les sillons sont à la même échelle, voire beaucoup plus grands que la surface à retirer. Cette méthode ne marche que si les échantillons sont grands devant la rugosité.

2.3.2 retrait_pente2(x,y,z,ordre)

Il y a donc une autre fonction qui marche mieux dans notre cas et qui permet d'aplanir l'échantillon. Elle prend en entrée les coordonnées des points et l'ordre de la surface à retirer. L'ordre 1 correspond à un plan et l'ordre 2 à une surface courbe.

Les tableaux de coordonnées sont transformés en listes (**tableau.flatten()**), leur permettant d'être exploitables par les modules utilisés ensuite.

La fonction **curve_fit**(f, (x,y), z) permet de trouver l'équation qui suit la forme de fonction f qui permet de coller le plus aux coordonnées des points en entrée. Les fonctions f sont des surfaces d'ordre 1 et d'ordre 2, avec des équations du type :

$z = ax + by + c$ pour l'ordre 1
$z = ax^2 + by^2 + cxy + dx + ey + f$ pour l'ordre 2

Ces deux fonctions correspondent à **ordre1**(xy,a,b,c) et **ordre2**(xy, a, b, c, d, e, f) dans le programme
Une fois les coefficients optimaux trouvés (a,b,c ou a,b,c,d,e,f), on retire la surface correspondante au tableau initial. On renvoie le tableau z, dont l'image a été mise à plat.

Ce qui est en commentaires permet d'afficher un graphique en 3D superposant la surface initiale avec la surface retirée.

2.4 rotation_img(z)

Cette fonction prend z en entrée et permet de faire tourner l'image pour aligner les sillons avec l'axe y.

Elle fonctionne avec le taux de variation de z selon y. La but est d'avoir le moins de variations possibles selon y. En supposant qu'on ait une structure linéaire parfaite, si la structure est parallèle à l'axe y, on n'a pas

de variations et le taux de variation est donc nul. Tandis que si les sillons ne sont pas alignés avec l'axe y, on va observer des variations de hauteur suivant y et donc un taux de variations non nul.

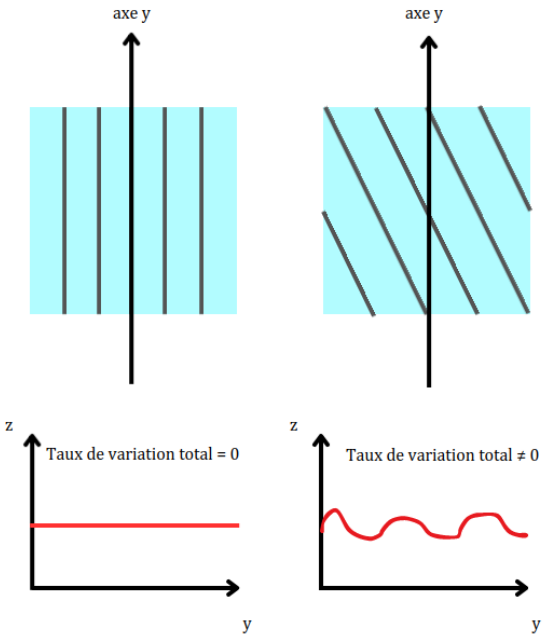

FIGURE 2 – Taux de variation sur une structure linéaire

Le but est donc de minimiser le taux de variation moyen, c'est à dire la pente moyenne en y (calculée par la fonction **vect_pente**(z), détaillée plus tard).

On calcule donc la pente moyenne en y au départ, servant seulement à initialiser la variable contenant la pente minimum.

Chercher à minimiser la pente en y ne marche que si on a une structure linéaire et sans bruit. Comme ce n'est pas le cas pour toutes nos images, on utilise la fonction **nd.gaussian_filter**(z, sigma) lisse l'image selon l'axe y (à peu près parallèlement aux sillons) et très peu selon l'axe x pour conserver la forme des sillons. Voici un exemple de ce que fait la fonction avec les paramètres entrés dans le code.

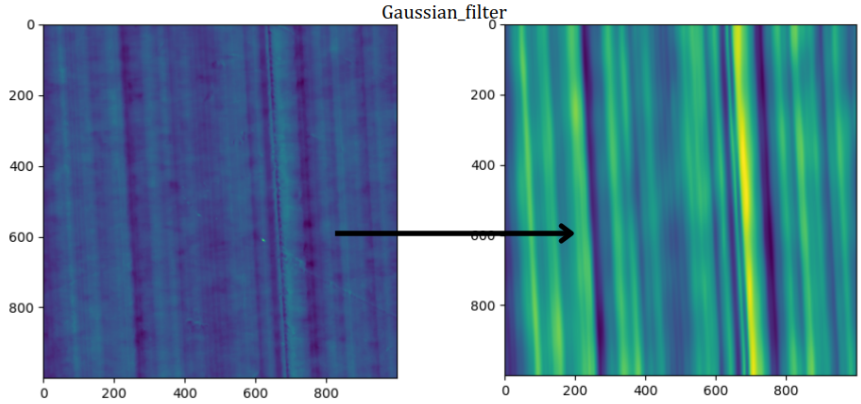

FIGURE 3 – Lissage de l'image

Une fois qu'on a une image lissée et non plus le tableau d'origine on cherche pour 100 angles dans un rayon de [-5° ; 5°] quel angle minimise la pente en y. On mesure donc la pente en y pour chaque angle imposé.

L'angle minimal trouvé permet ensuite de faire tourner l'image d'origine avec la fonction textbfsitk.Euler2DTransform(coordonnées du centre de rotation, angle). La fonction retourne l'image pivotée sous forme de tableau.

2.5 crop_to_data(image)

Cette fonction permet d'éliminer les valeurs Nan générées par la rotation de l'image. Elle cherche le plus grand rectangle pour lequel il n'y a pas de Nans dans l'image.

On enlève d'abord toutes les colonnes et lignes remplies de Nans. Puis suppression de la ligne ou colonne qui contient le plus de valeurs Nans tant qu'il reste encore des Nans dans l'image.

2.6 initialisation(z)

Cette dernière fonction permet de mettre à 0 le plan moyen de l'image afin de faciliter les calculs

Enfin la dernière opération met à jour les coordonnées de x et de y. Après la fonction **crop_to_data**(z), les dimensions de z on été modifiées, il faut donc mettre à jour celles de x et de y en utilisant **np.mgrid** qui génère des tableaux.

3 Calcul des différentes variables

Les valeurs des différentes variables sont réunies dans la fonction **calculs_param**(x,y,z) elle appelle les fonctions calculant un paramètre de surface et le stocke dans différentes variables.

3.1 rugosite_moyenne(z)

La fonction prend en entrée le tableau des hauteurs et renvoie la moyenne de la valeur absolue des valeurs du tableau grâce à des modules numpy.

$$\frac{1}{n} \times \sum_{1}^{n} |z_i|$$

3.2 rugosite_moyenne_quadratique(z)

Même fonctionnement que pour la fonction précédente avec la moyenne quadratique :

$$\frac{1}{n} \times \sqrt{\sum_{1}^{n} z_i^2}$$

3.3 skewness(z)

idem avec la formule suivante :

$$S_{sk} = \frac{1}{R_{sk}^3} \frac{1}{n} \sum_{1}^{n} z_i^3$$

La valeur retournée est ensuite divisée par S_q^3 dans la fonction **calculs_param**(x,y,z) pour obtenir le paramètre d'asymétrie. S_{sk}

3.4 kurtosis(z)

idem avec la formule suivante :

$$\frac{1}{S_{ku}} \frac{1}{n} \sum_{1}^{n} z_i^4$$

On divise ensuite par S_q^4 pour avoir S_{ku}

3.5 mean_width(z)

Calcul de la largeur moyenne des éléments de profil (S_{sm}). Cela équivaut à la mesure d'une période dans un signal périodique. Pour repérer une période, comme l'échantillon a été aplani et possède un plan moyen centré en zéro, la fonction repère donc les indices (stockés dans **periods**) ou le profil passe des valeurs positives aux négatives (avec **np.where**)pour chaque profil (= ligne du tableau).

Pour obtenir la largeur d'une période, en indices (pas encore à l'échelle), on mesure le nombre d'indices entre chaque case du tableau **periods**, avec **np.diff**. La valeur retournée est la moyenne de tous les écarts. On la multiplie ensuite par l'échelle (**scale**) pour obtenir S_{sm}.

3.6 Calcul des pentes

On utilise premièrement la fonction **slopes**(z) qui calcule les pentes selon x et selon y. Puis la fonction **rms_slope** la moyenne quadratique des pentes pas encore à l'échelle. La fonction **vect_pente**(z), renvoie les pentes moyennes selon x et selon y.

La première fonction calcule les pentes selon x, puis selon y avec la formule du taux de variation :

$$\frac{dz}{dx} = \frac{z(x+h) - z(x)}{(x+h) - x} = \frac{z_{i,j+1} - z_i, j}{x_{i,j+1} - x_i, j} = \frac{z_{i,j+1} - z_i, j}{1}$$

$$\frac{dz}{dy} = \frac{z(y+h) - z(y)}{(y+h) - x} = \frac{z_{i+1,j} - z_i, j}{x_{i+1,j} - x_i, j} = \frac{z_{i+1,j} - z_i, j}{1}$$

La valeur de la pente correspond donc à la différence entre les indices en colonnes d'un côté et en ligne de l'autre (dans le cas où x et y sont tous espacés d'une unité. On utilise donc le module **np.diff** pour effectuer le calcul pour les x. Pour calculer la pente en y, il faut transposer la matrice au préalable (**np.transpose**).

La fonction retourne donc deux tableaux avec une colonne en moins par rapport à z pour les x et une ligne en moins pour les y. Ils contiennent les valeurs des pentes pour chaque point, selon x ou y.

Ces tableaux sont utilisés pour deux fonctions : **rms_slope** et **vect_pente**, qui fonctionnent toutes les deux sur le même principe que **rugosite_moyenne_quadratique(z)**.

Les formules utilisées sont les suivantes :

$$S_{sm} = \frac{1}{n} \times \sqrt{\sum_{1}^{n} Pentex_i^2 + Pentey_i^2}$$

$$Pentemoyennex = \frac{1}{n} \times \sqrt{\sum_{1}^{n} Pentex_i^2}$$

$$Pentemoyennex = \frac{1}{n} \times \sqrt{\sum_{1}^{n} Pentey_i^2}$$

3.7 rapport_matière(z)

Cette fonction calcule les différents paramètres provenant de la courbe de rapport de matière. Cette courbe correspond aux probabilités cumulés en fonction des hauteurs décroissantes.

On commence donc par prendre le tableau z, des hauteurs en entrée et à le mettre sous forme d'une liste (**.flatten().tolist()**) décroissante (**.sort(reverse=True)**). La probabilité d'une hauteur est de $\frac{1}{n}$, pour chaque valeur de z, la probabilité augmente de $\frac{1}{n}$. On crée un tableau **proba**, dont la valeur à chaque indice correspond à la probabilité cumulée de z au même indice. Un case du tableau est donc de la forme :

$$\frac{1+i}{len(z)}$$

Pour le construire, on utilise la fonction **np.linspace** qui fait ça automatiquement.

La deuxième partie de la fonction consiste à trouver la séquente de plus faible pente qui contient 40% des valeurs. On initialise d'abord la variable **pente_min** qui sert à stocker la pente la plus faible. Une boucle parcourt 60% des premières valeurs, et calcule la pente de la séquente pour chacune. 40% des valeurs de hauteur situées entre les points d'intersection de la séquente, correspondent à 40% des indices du tableau z, c'est à dire $0.4 \times len(z)$ indices. Si le premier point de la séquente est à l'indice i, le second est donc à l'indice $i + 0.4 \times len(z)$. On obtient donc la formule de pente suivante :

$$Pente = \left|\frac{\Delta z}{\Delta x}\right| = \left|\frac{z_{i+0.4\times len(z)} - z_i}{proba_{i+0.4\times len(z)} - proba_i}\right|$$

En sortie de boucle, on a la séquente de pente minimale et les deux points qui la définissent. On peut en déduire son ordonnée à l'origine, Rpk et l=sa valeur en 1 : Rvk. Mr1 et Mr2 sont respectivement les probabilités correspondantes lues sur la courbe du rapport de matière.

4 Affichage des images et des graphes

Une fois les variables calculées, le code permet d'obtenir des informations complémentaires : l'image de la semelle en profilométrie, la fonction de densité des hauteurs et la courbe de rapport de matière.

4.1 img_profilo(x,y,z,file_name, taille_police, scale)

Cette fonction permet de construire un graphique affichant une image similaire à celle obtenue en profilométrie et dont l'échelle est variable en fonction de l'amplitude des hauteurs présentes. L'image est sauvegardée dans le même dossier que celui d'origine, avec l'extension **_profilo** à son nom.

4.2 courbe_distrib_hauteurs(z, file_name, taille_police)

La fonction contruit un graphe de la distibution des hauteurs et l'enregistre avec le nom d'origine + **_distrib**
Elle utilise la fonction **distrib_hauteurs**(z) qui donne les coordonnées des points à relier formant la distibution. Elle construit un histogramme avec le module **np.histogram**, puis récupère le centre des intervalles de l'histogramme associé à leur fréquence.

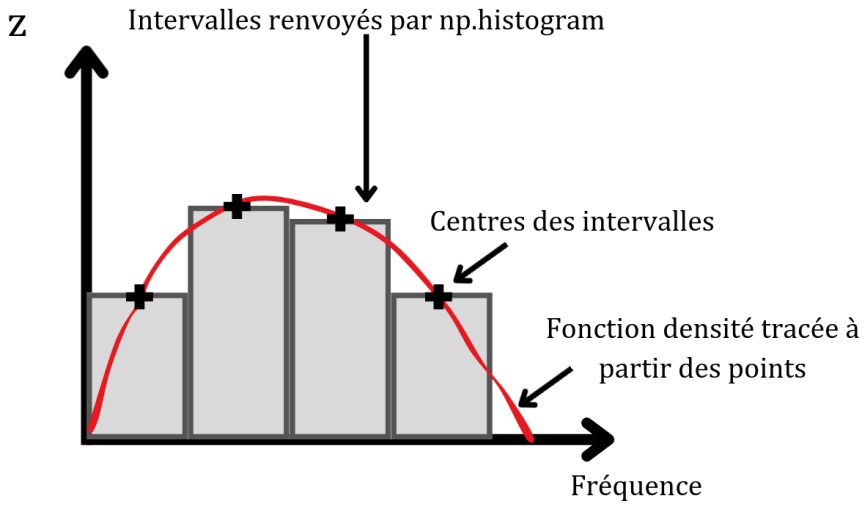

FIGURE 4 – Construction du graphe de la répartition des hauteurs

4.3 courbe_rapport_matiere(z, Rpk, Rvk, Mr1, Mr2, pente_min, file_name, taille_police)

Premièrement, construction de deux tableaux, un contenant les ordonnées (= z sous forme de liste décroissante) et les abscisses : **proba**, allant de 0 à 1 en incrémentant de $\frac{1}{len(z)}$ à chaque case. Ces points représentent la courbe de rapport de matière.

Ensuite tracé de la droite en indiquant les coordonnées de ses extrémités (R_{pk}, R_{vk})

Puis ajout des points caractéristiques, et des aires des pics et des sillons. Le code aurait pu être fait en beaucoup moins de lignes je pense.

La partie en commentaires en fin de fichier **main** permettait de construire et d'afficher un tableau avec les paramètres calculés.

4.4 Récupération des données

Les données sont récupérées avec le fichier **param_tab**, qui parcourt chaque dossier contenant les données pour lire les fichiers .xyz. Chaque variable calculée est stockée dans un tableau, et les fonctions traçant les graphes enregistrent directement l'image

Puis la fonction **conversion_csv**(tab, fichier_csv) retranscrit toutes les valeurs calculées dans un fichier csv existant.