Détection de signaux émis par des protéines lors d'une expérience de localisation de particules uniques

Théo Moins et Florence Forbes

INRIA, 25 juillet 2018

Résumé. Ce document résume un processus de détection et d'analyse de signaux émis par des protéines, à partir d'une acquisition vidéo effectuée lors d'une observation microscopique de type 'SMLM' (*Single Molecular Localization Method*).

1 Position du probléme

1.1 Problématique biologique

La protéine Tau est une protéine présente dans le cerveau et dont le dysfonctionnement provoque de nombreuses pathologies dont l'une d'entre elles est la maladie d'Alzheimer. Appartenant à la famille des protéines MAPs, son rôle est de contrôler l'architecture et la stabilité des réseaux de microtubules, qui permettent le bon fonctionnement des neurones.

Pour étudier la façon dont Tau contrôle la dynamique des microtubules, une approche développée par Isabelle Arnal (GIN) est de reconstituer *in vitro* des réseaux de microtubules dynamiques en présence de Tau, et de les observer en temps réel par microscopie[1]. Depuis peu, l'équipe parvient à observer en temps réel la protéine Tau fluorescente en interaction avec les microtubules à l'échelle de la molécule unique.

L'objectif ici est donc de traiter les données issus de ces enregistrements, et en particulier, déterminer l'intensité de fluorescence d'une molécule unique.

En théorie, le profil de l'intensité des molécules au cours du temps est une fonction en escalier décroissante, et l'on cherche à faire deux estimations :

1. Le nombre de segments de chaque signal, afin de connaître l'état d'oligomérisation de chaque molécule et d'en déduire la proportion monomère/dimère de l'échantillon.
2. La hauteur de marche (qui doit être identique pour chaque signal), afin d'en déduire l'intensité de fluorescence d'une protéine Tau unique.

Cependant, cette analyse peut s'avérer très complexe à réaliser car le signal est fortement bruité.

1.2 Modélisation

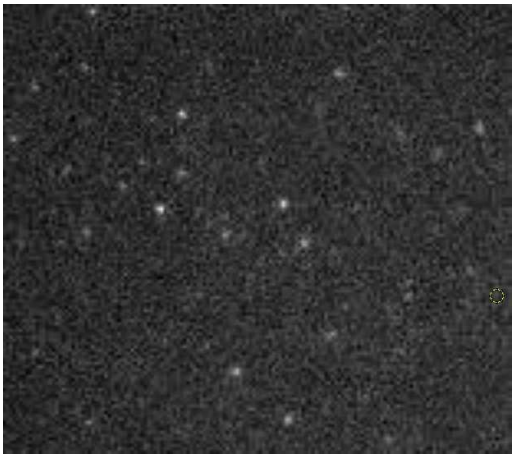

Fig. 1. Première *frame* d'une acquisition vidéo

L'analyse se divise en deux temps :

1. Détection des protéines sur les premières frames de la vidéo
2. Étude de l'intensité lumineuse de chaque protéine au cours du temps.

La première partie va être effectué sur `ImageJ`, un logiciel permettant l'analyse et le traitement de vidéos. Étant *open-source*, il offre la possibilité d'ajouter des plugins, mais aussi d'écrire des macros afin d'étendre ses fonctionnalités. Ainsi, parmi les *plugins* disponible pour `ImageJ`, `THUNDERSTORM` [2] est spécialement conçu pour la détection et l'analyse de particules observées à partir d'une méthode *SMLM* (c'est-à-dire molécule unique). Nous utiliserons ce plugin afin de détecter des protéines sur une vidéo et exporter les intensités lumineuses en fonction du temps.

La seconde partie de l'analyse s'effectue sur Rstudio. Le traitement effecuté sur les données se divise en plusieurs étapes :

1. On procède au calcul des marches pour chaque signal acquis sur la partie précédente (deux méthodes uvont être utilisées ici)
2. Ensuite, on supprime les signaux non exploitables.
3. Parmi les signaux restant, on calcule chaque hauteur de marche.
4. On détermine ensuite les signaux 'incertains', c'est-à-dire qui donnent des résultats qui peuvent paraître aberrants.
5. On estime la hauteur moyenne des marches, la proportion de signaux avec une marche, et la proportion avec deux marches.

2 Détection des protéines : utilisation du plugin `THUNDERSTORM` du logiciel *ImageJ*

On détaille ici les différentes étapes effectués pour la détection de points sur la vidéo, et automatisées dans une macro.

2.1 Configuration de la caméra

La configuration de la caméra choisie est la suivante :

Pixel size [nm]	267.0
Photoelectrons per A/D count	3.6
Base level [A/D counts]	414.0

Ces paramètres (ainsi que tout les suivants) peuvent être changés dans la macro.

2.2 Méthode de détection de points utilisés

Pour détécter les points blancs sur chaque *frame* de la vidéo, `THUNDERSTORM` procède en 3 étapes :

1. Filtrage de l'image : Avant de seuiller l'intensité de chaque pixel, un filtrage est effectué afin de 'lisser' l'image et prendre en compte le voisinage de chaque pixels pour la détection. Parmi l'ensemble des filtrages disponible c'est le *Wavelet Filter* que nous utilisons.
2. Détection 'approximative' de chaque molécule : détection des maxima locaux sur l'image filtrée, par une méthode de seuillage
3. Localisation précise des molécules : en revenant sur l'image initial, on utilise une méthode de *fit* par maximum de vraisemblance d'une fonction d'étalement de point (PSF Gaussienne) en recherchant des points trouvés approximativement à l'étape précédente.

L'ensemble de cette configuration et des paramètres associés (seuillage, etc.) a été suggéré par Izeddin et al. [3].

2.3 Traitement après la détection

Après la détection des points sur chaque frame de la vidéo, on procède ensuite au traitement suivant :

- On ne considère que les points détectés lors des premières *frames* de la vidéo.
- Parmi les valeurs retournées par `THUNDERSTORM`, certaines peuvent nous permettre de détecter des faux positifs : ainsi, on retire les points qui ont une valeur de *sigma* trop élevé, ce qui correspond très grand étalement (et donc souvent due à une mauvaise détection).

- On retire également les points pour lequel la valeur *offset* est trop faible, ce qui correspond à une estimation de l'intensité du bruit de fond quasi nulle autour du point, ce qui est également signe d'une mauvaise détection.
- Fusion des points : sur les *frames* que l'on conserve pour détecter nos points, si l'on détecte un point sur chaque *frame* avec des coordonnées quasi-identiques, ces points seront fusionnés dans le tableau des résultats.

Les constantes associées à chacun des filtres ont été fixées empiriquement, et peuvent être modifié dans une boîte de dialogue qui s'affiche pendant l'execution de la macro.

2.4 Sélection des ROI *(Region Of Interest)*

De la feuille de résultat obtenue, on extrait les coordonnées de chaque point, ainsi que la valeur de *sigma*. Pour chaque point, on crée une région d'intérêt qui est un cercle centré sur les coordonnées détectées, et de rayon *sigma*.

Ensuite, le niveau de gris est calculé en faisant la moyenne sur chacun des disques, et ce pour chaque *frame*.

Le tableau de résultat est ensuite exporté en csv. Ce fichier contient en colonne les moyennes de gris de chaque point en fonction du temps.

2.5 Génération de deux images d'illustration

En plus du fichier contenant les résultats, on récupère la première *frame* de la vidéo afin de l'afficher et de pouvoir faire le lien entre les signaux et la vidéo. La première image récupérée n'est qu'une capture de la première *frame* (voir Fig.1), la deuxième également mais les points détectés sont entourés en jaune, et où figurent les labels correspondant à la numérotation de chaque protéine dans le fichier `csv` (voir Fig.2, les labels sont peu lisibles ici car l'image est rétrécie).

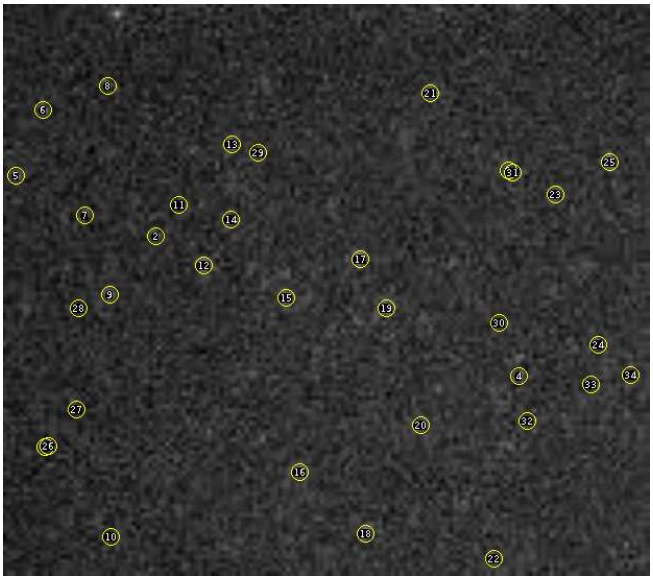

Fig. 2. Deuxième image exportée avec les résultats

3 Détection des marches dans les signaux : utilisation du package de R `Segmentor3IsBack` [4]

3.1 Calcul des marches

Pour chaque signal, on effectue la segmentation, puis on stocke le résultat dans 3 tableaux, référençant le nombre de marches, la hauteur de chaque palier, et l'abscisse où a lieu chaque marche.

Pour le calcul de la segmentation, le package `Segmentor3IsBack` nous propose plusieurs possibilités :

- Le choix de la distribution des données : ici, 3 possibilités sont des candidats potentielles. Nous pouvons choisir une loi Gaussienne avec variance constante, une loi de Poisson, ou une loi binomiale négative. Nous avons pu observer des résultats similaires sur les modèles Gaussien et de Poisson, et meilleurs que ceux avec la loi binomiale négative. **Le modèle que nous avons choisi par défaut est le Gaussien.**
- Le choix du nombre maximal de marche : pour utiliser `Segmentor3IsBack`, il est nécessaire d'indiquer de combien de segments est composé au maximum un signal. Cet argument semble être clé dans la bonne détection des marches, car même si nous savons que nous souhaitons toujours moins de 3 segments, indiquer une valeur de 3 ne donne pas forcément les meilleurs résultats. Une première méthode a été donc de fixer une plus grande

valeur, puis de fusionner des segments lorsqu'ils sont proches. **Actuellement, nous avons choisi de considérer jusqu'à 10 marches dans un signal, puis de fusionner deux marches très proches.**

- Fusion de deux segments proches : le critère pour fusionner deux segments est le suivant : **si la différence de deux hauteurs de marche consécutives est inférieur à $\frac{i_{max}-i_{min}}{8}$ (choix empirique), avec i_{max} et i_{min} le maximum et minimum d'intensité du signal, alors le tout est remplacé par un seul et unique segment.**

3.2 Suppression des signaux mal analysés

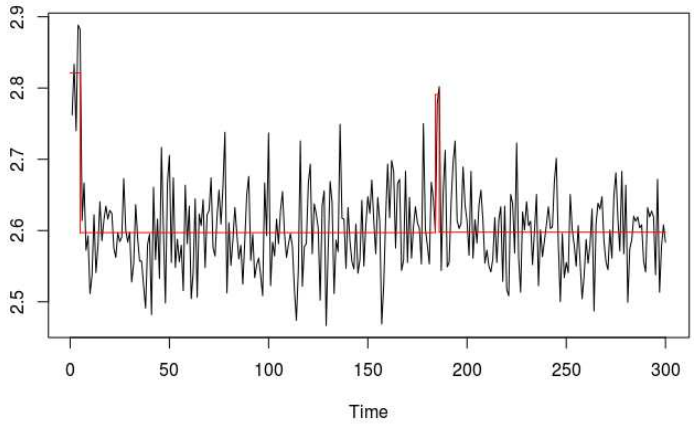

Fig. 3. Exemple d'une courbe exploitable mais pour laquelle la détection d'une fausse marche due au bruit élimine la protéine associée

Le modèle physique nous assure que chaque signal n'a qu'entre une et deux marches, et les hauteurs de marche sont 'décroissantes', c'est à dire que le pallier avant la marche est toujours supérieur à celui d'après. Or, ces critères ne sont pas imposés lors du calcul des marches, par conséquent nous les utilisons pour trier les faux positifs de la partie précédente. Il peut aussi arriver qu'un signal qui était analysable ne vérifie pas ces critères. Ces cas sont problématiques et nous cherchons à les éliminer (voir Fig.3).

4 Amélioration de la détection : apprentissage de la pénalité pour le nombre de marches avec le package `penaltyLearning` [6]

Parmi les signaux dont l'analyse est erroné, deux types d'erreurs se distinguent :

1. Le nombre de marches n'est pas celui attendu.
2. Le nombre de marches est le bon mais les marches sont mal positionnées par rapport au signal.

Au vue des résultats obtenus avec l'analyse des signaux avec le package `Segmentor3IsBack`, nous constatons que la majorité des erreurs proviennent de la deuxième étape, c'est-à-dire une mauvaise estimation du nombre de marches.

Détaillons le fonctionnement du calcul fait par `Segmentor3IsBack` pour comprendre le problème. Notons :

- $(y_t)_{1\leq t\leq n}$ le signal observé, réalisations de n variables aléatoires $(Y_t)_{1\leq t\leq n}$,
- K le nombre de segments,
- $(t_1, ..., t_{K-1})$ les instants de rupture.

Pour notre problème nous avons supposé que

$$Y_t \sim \mathcal{N}(\mu_k, \sigma), \text{ avec } t_{k-1} \leq t \leq t_k \text{ et } 1 \leq k \leq K-1$$

Ainsi, la recherche de la meilleur segmentation se fait en deux temps :

1. Calcul de la vraisemblance $\hat{y}_t^k = \underset{u\in\mathcal{M}_k}{\operatorname{argmin}} \left\{||y_t - u||_2^2\right\}$, avec $\mathcal{M}_k$ l'ensemble des partitions de $\{1, ..., n\}$ en k éléments, $\forall k$ de 1 à K,
2. Sélection du nombre de segments $\hat{k}$.

Il est clair que si l'on choisit $\hat{k} = \underset{1\leq k\leq K}{\operatorname{argmin}} \left\{||y_t - \hat{y}_t^k||_2^2\right\}$, le fait d'ajouter un paramètre à notre modèle va toujours permettre de réduire la vraisemblance (intuitivement, un segment supplémentaire même de taille 1 permet forcément un meilleur *fit* des données) et donc on prendrait toujours la valeur maximale K. C'est pourquoi l'on introduit une fonction appelé *pénalité*, que l'on ajoute à la vraisemblance et qui pénalise un modèle selon différents critères. On obtient comme forme générale :

$$\hat{k} = \underset{1\leq k\leq K}{\operatorname{argmin}} \left\{||y_t - \hat{y}_t^k||_2^2 + f(\hat{y}_t^k, n)\right\}$$

Il existe dans la littérature beaucoup de pénalités différentes, les plus connues étant les critères AIC ($f(\hat{y}_t^k, n) = 2k$) et BIC ($f(\hat{y}_t^k, n) = 2kln(n)$), qui pénalisent le modèle selon son nombre de paramètres. D'autres pénalités existent pour le

problème de segmentation, faisant par exemple intervenir la taille de chaque segments, etc. Ici, le package `Segmentor3IsBack` utilise une pénalité proposé par Lebarbier [8] :

$$f(\hat{y}_t^k, n) = k(2ln(n/k) + 5)$$

Cette pénalité a principalement été mise au point et essayée sur des données génomiques de type "RNA-seq". Les données que nous traitons ont des propriétés différentes : les hauteurs de segments sont décroissantes, il n'y en a pas plus de 3, survenant souvent très tôt dans le temps, etc.

C'est pourquoi une idée proposé par Hocking et al. [7] est de fournir la pénalité en se basant sur des signaux pour lesquels on a annoté le nombre de segments attendus. Par la suite, on cherche une fonction de la forme $f(\hat{y}_t^k, n) = k\lambda$, $\lambda > 0$.

Pour ce faire, les étapes sont les suivantes :

1. Pour chaque signal annotés, calcul du modèle à k segments pour toutes les valeurs de k possibles, en utilisant `Segmentor3IsBack`.
2. Pour chaque modèle de chaque signal, calcul du nombre d'erreurs effectués par rapport aux prédictions (l'objectif de cet algorithme étant de minimiser ce nombre d'erreurs).
3. Calcul des intervalles de λ correspondant au choix de chaque modèle. Pour ce faire, on calcul la vraisemblance $\mathcal{L}_k = ||y_t - \hat{y}_t^k||_2^2$ de chaque modèle, puis on observe pour quel λ a-t-on $\underset{k}{\text{argmin}}\{\mathcal{L}_k + k\lambda\} = 1, 2, ..., K$.

   On obtient ainsi K intervalles de λ disjoints et dont l'union vaut $\mathbb{R}^+$. En notant $([\lambda_i; \lambda_{i+1}[)_{0 \leq i \leq K}$, avec $\lambda_0 = 0$ et $\lambda_K = +\infty$, on a

$$[\lambda_i; \lambda_{i+1}[= \left\{ \lambda \in \mathbb{R}^+; \underset{k}{\text{argmin}}\{\mathcal{L}_k + k\lambda\} = i \right\}$$

4. Avec les annotations des signaux, on en déduit pour chaque courbe l'intervalle de λ pour lequel l'erreur est minimisée (pour chaque signal).
5. Choix d'une (ou plusieurs) caractéristique x_i commune à chaque signal i afin d'établir la régression. Celle-ci n'est pas faite sur λ mais sur $ln(\lambda)$ (afin de la faire sur $\mathbb{R}$ entier) : $ln(\lambda) = f(x_i)$. Par exemple, si l'on souhaite améliorer le critère BIC ($\lambda = ln(n)$), on choisi $x_i = ln(ln(n_i))$, et on remplace $f(x_i) = x_i$, qui est la pénalité classique, par $f(x_i) = \beta x_i + w$, avec (β, w) choisi de tel sorte à minimiser l'erreur sur les signaux annotés. Ici, on utilise la fonction `featureMatrix` de `penaltyLearning` qui permet de calculer 144 de caractéristiques (tel que la moyenne empirique, la somme, les quantiles, la racine de ces valeurs, etc), puis ne conserver que les plus pertinentes pour notre problème.
6. Calcul de la régression qui minimise le nombre d'erreurs faites sur les signaux.

Ce traitement va avoir pour effet de "lisser" la segmentation, et donc d'avoir bien moins de cas comme celui affiché Fig.3.

5 Synthèse des deux modèles et création d'un indice de confiance

5.1 Comparaison des deux modèles

Afin de comparer le modèle utilisant la pénalité de Lebarbier avec celui utilisant une pénalité apprise avec le package `penaltyLearning`, nous avons considéré des vidéos sur laquelle nous avons déterminer le nombre d'erreurs effectués lors de segmentation avec les deux modèles. Nous affichons le résultats sous la forme de courbe ROC (*Receiver Operating Characteristic*), qui permettent de visualiser efficacement la performance des deux modèles. Cette courbe donne le taux de vrais positifs (marches détectés qui le sont effectivement) en fonction du taux de faux positifs (marches détéctés qui ne l'est pas), lorsque l'on fait varier λ. Ainsi, à (0,0), le modèle ne détecte aucune marches, à (1,1), le modèle détecte le nombre maximale de marches sur chaque courbes, et à (0,1), le modèle est parfait.

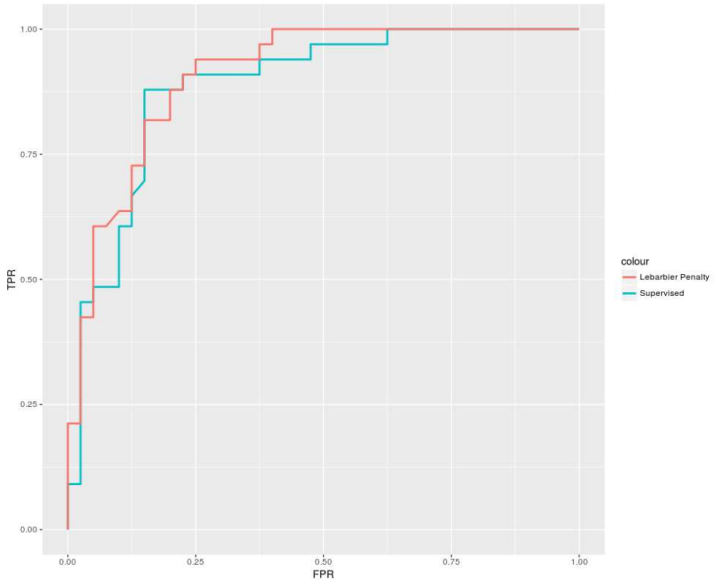

Fig. 4. Un exemple de courbes ROC pour les deux modèles de pénalité

En affichant plusieurs de ces courbes (dont un exemple est donné Fig.4), nous observons qu'aucuns de ces deux modèles ne se distingue en terme de performance. Cependant, même si l'on constate que les deux méthodes font souvent le même nombre d'erreurs, celles-ci peuvent être commise sur des signaux différents : par exemple, dans la vidéo analysé correspondant à la courbe ROC de la Fig.4, parmi les 33 points qui ont été détectés, on relève 12 erreurs pour le modèle utilisant la pénalité de Lebarbier, et 11 dans la pénalité apprise. Néanmoins, 7

erreurs sont communes aux deux modèles, et pour les autres, au moins l'un des deux modèles est correcte. Un moyen d'amélioration est donc de sélectionner le meilleur modèle pour chacun des signaux.

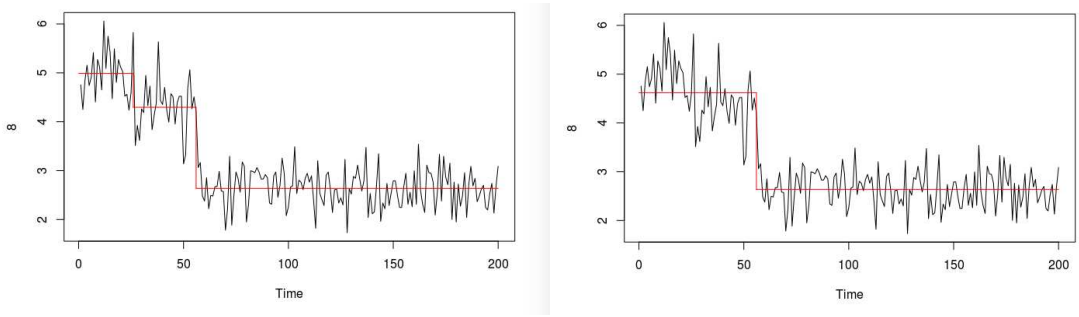

Fig. 5. Exemples de segmentations différentes selon le choix de la pénalité (à gauche, la pénalité de Lebarbier, et à droite celle apprise avec `penaltyLearning`)

5.2 Création d'un indice de confiance

Afin de pouvoir choisir lorsque deux modèles de segmentation donnent des résultats différents et afin de repérer les signaux incertains, il est intéressant de créer un indice de confiance p_i, compris entre 0 et 1, et qui permettrait de mesurer la qualité de la segmentation. Ainsi, p_i proche de 0 signifierait une segmentation très douteuse pour le signal i, tandis que p_i proche de 1 signifierait que l'on peut se fier aux calculs qui ont été établis.

Pour ce faire, nous allons prendre en compte 3 critères :

1. La taille des marches calculés : si la marche trouvé n'est pas plus grande que la variance globale du signal, cela signifie très probablement qu'elle est artificielle et que l'on ne peut pas s'y fier.
2. Le *fit* aux données : si l'on ne considérerait que le critère précédent, alors l'on préférerait toujours les signaux avec le moins de marches (car elles sont plus grandes). C'est pourquoi l'on contre-balance en prenant en compte la vraisemblance du modèle aux données, ce qui peut se faire en comparant les pertes de chaque signaux afin d'en sortir les moins fiables. Cela permet également de repérer les signaux non analysables, qui ne présente pas de réelles chutes d'intensité.
3. Enfin, au vue de notre contexte biologique, il est assez rare qu'une marche apparaissent à la fin de l'acquisition, et la détection de marches à la fin du signal est souvent une erreur, ce que l'on peut également prendre en compte ici.

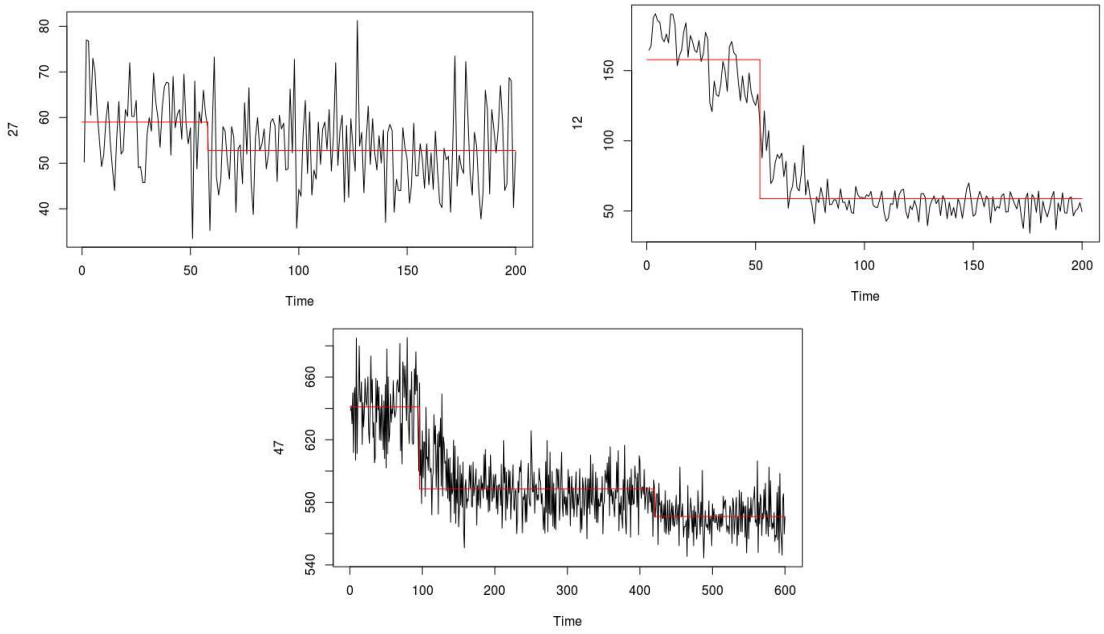

Fig. 6. Exemples de graphes incertains, détectables avec les 3 critères cités : rapport hauteur de marche sur bruit pour le 1^{er}, valeur de la fonction de perte pour le 2^{e}, et marche qui apparaît à la fin pour le 3^{e}

Pour ces trois critères, nous créons trois indices $p_{i,1}$, $p_{i,2}$ et $p_{i,3}$, et p_i sera la moyenne géométrique de ces trois critères :

$$p_i = p_{i,1}^{\alpha_1} p_{i,2}^{\alpha_2} p_{i,3}^{\alpha_3} \text{ , avec } \alpha_1 + \alpha_2 + \alpha_3 = 1$$

On utilise une moyenne géométrique et non arithmétique car on souhaite que p_i soit proche de 0 même si un seul des $p_{i,j}$ est proche de 0 et les autres proches de 1.

Calcul de l'indice $p_{i,1}$ pour le critère 1 :

Pour chaque signal i, on souhaite comparer la/les hauteurs de marches détectés $h_{i,1}$ (et $h_{i,2}$ quand il y en a 2) à la variance du signal σ. Si une hauteur h mesuré est inférieur à σ, il est probable de penser que l'on est face à un faux positif.

On choisit alors de caractériser la condition suivante :

— Si le signal a une marche, alors nous ne pouvons pas avoir de doutes en considérant la taille de la marche si $h > 2\sigma$

— Si le signal a deux marches, alors nous ne pouvons pas avoir de doutes en considérant les tailles des deux marches si $h_1 > \sigma$ et $h_2 > \sigma$

On considère ainsi la fonction de densité suivante :

$$\begin{aligned} f_1 : \mathbb{R}^+ &\to [0;1] \\ x &\mapsto \frac{1}{2}\left(1 + th(x - \frac{3}{2})\right) \end{aligned}$$

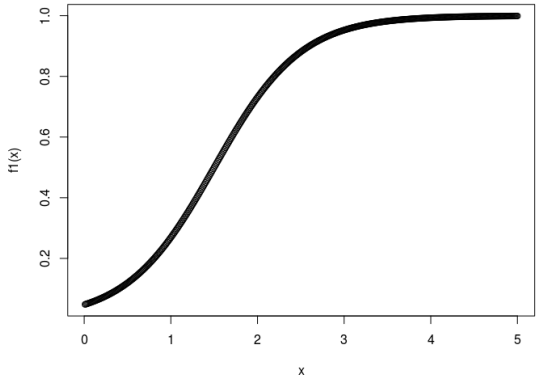

Fig. 7. Graphe de la fonction f_1

Et on choisit $p_{i,1} = \begin{cases} f_1(\frac{h_{i,1}}{\sigma}) & \text{si } h_{i,2} = 0 \\ \sqrt{f_1\left(2\frac{h_{i,1}}{\sigma}\right) f_1\left(2\frac{h_{i,2}}{\sigma}\right)} & \text{sinon} \end{cases}$

Calcul de l'indice $p_{i,2}$ pour le critère 2 :

Pour ce critère, on souhaite prendre en compte les valeurs des pertes $\mathcal{L}_i = ||y_i - \hat{y}_i||_2^2$, mais contrairement au critère précédent, nous ne disposons pas de valeurs de référence pour établir une comparaison, nous comparons simplement les valeurs entre elles. La solution intuitive est de translater les valeurs $\mathcal{L}_i$ dans $[\beta; 1]$, avec β l'indice que l'on souhaite donner au pire signal (c'est-à-dire avec le plus grand $\mathcal{L}_i$).

On choisit donc

$$p_{i,2} = 1 - \alpha \frac{\mathcal{L}_i - \min\limits_i \mathcal{L}_i}{\max\limits_i \mathcal{L}_i - \min\limits_i \mathcal{L}_i}$$

Calcul de l'indice $p_{i,3}$ pour le critère 3 :

Pour le dernier critère, on considère le temps de la dernière rupture t_i pour chaque signal, et on considère la fonction :

$$\begin{array}{rccll} f_2 : [0;1] & \to & [0;1] & & \\ x & \mapsto & 1 & \text{si } 0 \leq x \leq 0.5 \\ x & \mapsto & 2-2x & \text{si } 0.5 \leq x \leq 1 \end{array}$$

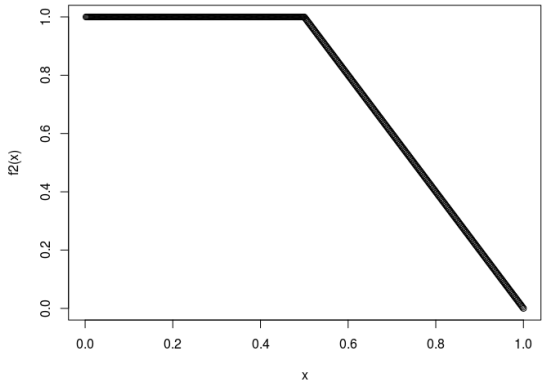

Fig. 8. Graphe de la fonction f_2

On applique cette fonction au rapport $\frac{t_i}{n}$ pour obtenir $p_{i,3}$.

On en choisit ensuite $(\alpha_1, \alpha_2, \alpha_3)$ (par exemple égale à $(0.5, 0.2, 0.3)$), puis on en déduit la valeur de p_i. On effectue alors la segementation de chaque signal avec les deux méthodes, puis lorsque les résultats diffèrent, on conserve le modèle avec le meilleur indice de confiance. On établit également un seuil de confiance p_{min}, pour lequel on considère que tout les signaux vérifiant $p_i < p_{min}$ sont très probablement mal analysés.

Néanmoins, malgré le fait que p_i soit compris entre 0 et 1, considérer par exemple que p_i est la probabilité que l'analyse du signal soit correct est complètement faux.

6 Estimation de la taille moyenne d'une marche

A cette étape, nous disposons de l'analyse temporel de chaque protéine, avec le nombre de segments, leur tailles, les hauteurs de marches, etc. Néanmoins, estimer le niveau d'oligomérisation ainsi que l'intensité moyenne d'une émission n'est pas immédiat. En effet, il se peut que deux émissions surviennent au même moment, et cela se traduit par une marche avec une taille doublée à la place deux marches normales, et à l'inverse il se peut que l'on trouve deux marches proches dans le temps avec l'algorithme précédent, mais dont il est clair à la vue de leur taille qu'il ne s'agit que d'une seule marche (voir Fig.9).

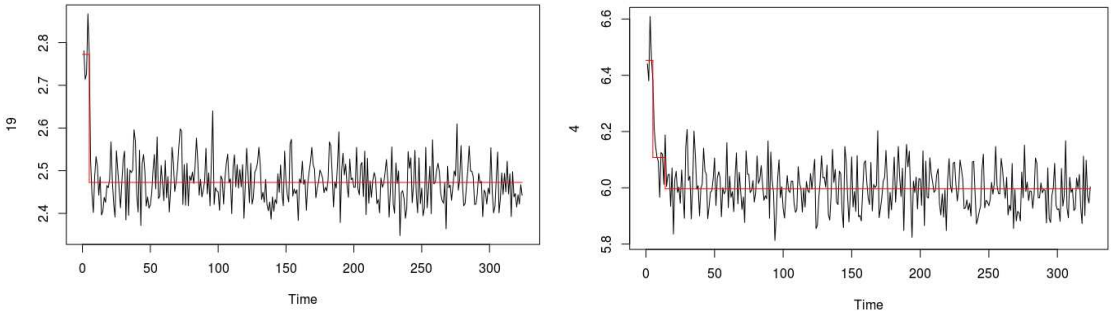

Fig. 9. À gauche, un exemple où une marche contient très probablement deux émissions au vue de sa taille. À droite, c'est l'inverse : deux marches ont été détectées mais il semblerait qu'il n'y en a en vérité qu'une seule.

6.1 Modélisation du problème

Objectif : Calculer la proportion $\pi \in [0;1]$ de signaux à 2 marches (la proportion de signaux à une marche serait donc 1-π), ainsi que la valeur moyenne d'une marche $m > 0$.

On dispose de deux jeux de données, $X^1 = \{x_1^1; \ldots; x_n^1\}$ correspondant aux hauteurs de première marche des signaux, et $X^2 = \{x_1^2; \ldots; x_n^2\}$ les hauteurs de deuxième marche lorsqu'il y en a une, et 0 s'il le signal n'a qu'une marche.

Intuitivement, X^1 doit comporter deux groupes de valeurs : celles autour de la valeur moyenne d'une marche m (la valeur qu'on souhaite calculer), et celles autour de $2m$ (cas de deux marches simultanées). Si l'on considère l'ensemble $X = \{x_1^1 + x_1^2; \ldots; x_n^1 + x_n^2\} = \{x_1; \ldots; x_n\}$, nous aurons également ces deux groupes de valeurs, mais avec une proportion de $(1 - \pi)$ pour le premier, et de π pour le second, ce qui correspond à la deuxième valeur que nous souhaitons estimer.

La distribution de notre échantillon est donc de la forme :

$$f(x) = (1 - \pi)f(x; m) + \pi f(x; 2m)$$

On suppose que que ces deux distributions sont gaussiennes, centrées sur m et $2m$:

$$f(x) = (1 - \pi)\mathcal{N}(x|m, \sigma_1) + \pi\mathcal{N}(x|2m, \sigma_2)$$

Il semble difficile de faire une supposition sur le lien entre les variances σ_1 et σ_2, car certaines valeurs sont la somme de deux acquisitions, d'autres non, et de plus, pour des raisons expérimentales, l'acquisition d'une deuxième marche (qui arrive plus tard dans le temps) est moins précise que la première.

6.2 Application de l'algorithme EM

Notre modèle est donc un mélange Gaussien à deux composantes avec une contrainte sur les paramètres ($m_2 = 2m_1$). Pour l'estimer, nous pouvons utiliser l'algorithme EM en indiquant la contrainte, ce qui est possible de faire avec la fonction `normalmixEM` du package `mixtools`. Nous obtenons ainsi toutes les valeurs souhaités (ici m et π)

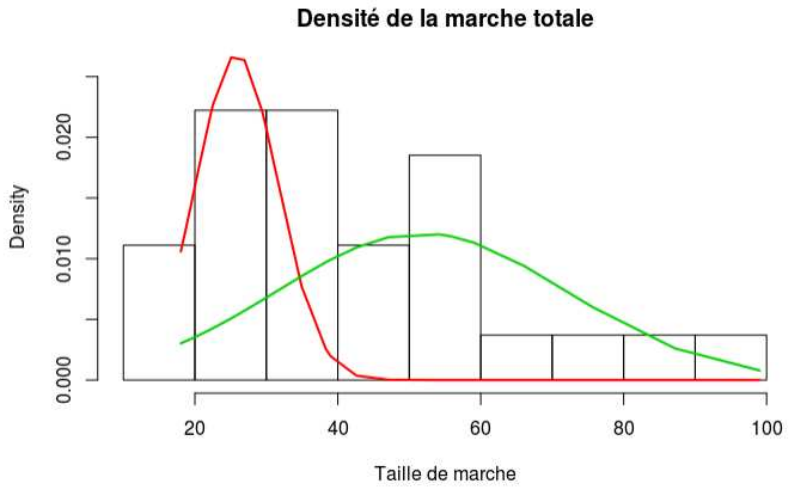

Fig. 10. Exemple d'une densité trouvée à partir des valeurs de la taille des marches. On observe bel et bien une somme de deux gaussiennes, avec une moyenne 2 fois plus élevée pour la deuxième.

7 Tutoriel : Comment effectuer l'analyse d'une vidéo

7.1 Utilisation de la macro sur `ImageJ`

L'ensemble des opérations effectués sur `ImageJ` ont été rassemblées dans une macro afin d'automatiser toutes les étapes de traitement.

Son fonctionnement est le suivant :

- Ouvrir ImageJ, puis ouvrir la vidéo que l'on souhaite traiter
- Ouvrir la macro sur *ImageJ*, puis la lancer en cliquant sur `Macros`, `Run Macro`
- Sélectionner le dossier de destination pour le résultat
- Une boîte de dialogue au milieu de l'exécution propose de modifier les constantes qui sont indiquées par défaut
- Attendre la fin d'exécution du programme

En sortie sera situé (dans le dossier choisi) un fichier `csv` contenant les valeurs des points détectés sur chaque *frame*, ainsi que deux images provenant de la vidéo qui permettent de faire le lien entre le numéro de la colonne du fichier contenant le résultat et la vidéo. **Attention** : le format du fichier de sortie est un csv avec des **virgules** pour séparateur.

Si jamais un message d'erreur intervient durant l'exécution de la macro ou durant la partie suivante, un moyen d'identifier le problème est de vérifier si le fichier *"results.csv"* a bel et bien été crée. Si ce n'est pas le cas, cela peut être due à un chemin particulièrement long, ou un problème dans l'analyse des données avec `THUNDERSTORM` (voir par exemple si les filtres et le merging ont bien été appliqués). Si le fichier a été crée et que le problème intervient dans la deuxième partie, on peut alors vérifier que le fichier csv contient une seule colonne de label, et que l'ensemble des autres colonnes ne sont composés que de nombres.

7.2 Organisation des dossiers

Pour la partie sur R, l'ensemble des fichiers sont divisés dans 3 dossiers :

1. Le premier dossier `Code` comporte les fichiers .R et .rmd qui contiennent l'ensemble du code, ainsi que les pages html générés.
2. Le deuxième dossier `Fit` comporte le fichier `"data.csv"` avec les données qui servent pour l'entraînement du modèle avec `penaltyLearning`, ainsi que le fichier `"ann.csv"` avec les annotations correspondantes. Il contient également le modèle `"Training.rda"` généré à partir de ces données.
3. Enfin, le dernier dossier `results` est le dossier dans lequel doit se trouver le fichier de sortie `"results.csv"` provenant de la macro sur `ImageJ`, ainsi que les deux images générées par cette même macro.

7.3 Entraîner le modèle de `penaltyLearning` avec de nouvelles données

Pour générer un modèle avec `penaltyLearning` qui servira à prédire la pénalité d'un signal, plusieurs étapes sont nécéssaires :

1. Collecter les données : la première étape est de récupérer les données temporelles des signaux que l'on souhaite annoter (cela peut se faire avec la macro sur ImageJ). Il faut ensuite placer le fichier résultant dans le dossier `fit`, et le renommer `"data.csv"`. Si l'on souhaite entraîner le modèle à partir de plusieurs vidéos, il suffit de copier-coller les colonnes supplémentaires au bout du fichier (comme si ce n'était qu'une vidéo mais avec plus de points)
2. Annoter les données : remplir un fichier csv que l'on nommera `"ann.csv"`, et qui contient 4 colonnes :
   - *protein*, qui est le numéro de chaque signal (il suffit de numéroter de 1 à N)
   - *min* et *max*, qui donne la "région" du signal dans laquelle on indique le nombre de segments observés. Par exemple, si un signal est composé de 200 points et que l'on observe une rupture à $t = 30$, on pourrait ajouter trois lignes, indiquant qu'entre 0 et 28 on n'observe aucune rupture, qu'entre 29 et 31, on en observe 1, puis 0 entre 32 et 200. Mais cette précision, en plus d'alourdir la tâche d'annotation à la main, ne semble pas donner de meilleurs résultats. Par conséquent, il est recommandé de ne mettre qu'une ligne par protéine, en mettant toujours 0 pour *min*, et la taille du signal pour *max*.
   - *annotation*, qui indique le nombre de ruptures observés. Seule trois possibilités sont données : soit l'on indique qu'il n'y a aucune marche en indiquant **0changes**, soit on indique qu'il y en a exactement une en indiquant **1change**, soit on indique qu'il y en une ou plus, en indiquant **>0changes**.
3. Executer le programme : si le fichier `"training.rda"` existe déjà dans le dossier `/Fit`, il est recommander de le déplacer, le renommer ou le supprimer, afin de constater que le nouveau modèle a bien été créer. Ensuite, il suffit d'appeler la fonction `Training` sur R avec les arguments adéquat (voir la partie suivante).

7.4 Modifier les chemins de dossier et générer les pages *html*

Deux fichiers *.rmd* servent à illustrer le résultat de l'analyse :

- `Taunique.rmd`, qui est un compte rendu contenant les estimations finales et les graphes incertains.
- `graphes.rmd`, qui contient l'ensemble des graphes analysés.

Pour générer les pages *html* associés à chacun des deux fichiers, il faut modifier le chemin indiquant les 3 dossiers (`/Fit`, `/Code` et `/Results`) dans le début de code, puis cliquer sur **knit** sur Rstudio.

8 Explication du code écrit en R

Au sein du code, les fichiers .rmd servent à générer les pages *html*, et appellent les fonctions définis dans les fichiers R et qui permettent l'analyse des signaux.

8.1 Description des fonctions principales du code :

Trois fichiers regroupent l'ensemble des fonctions implémentées :

Fichier `"fusionsegmentor3.R"` :

Ce fichier regroupe ce qui concerne la segmentation à l'aide du package `Segmentor3IsBack`. On y trouve deux fonctions :

- Une fonction `Fusion`, qui calcule la segmentation d'un signal avec la pénalité de Lebarbier. Elle prend en argument un vecteur contenant l'intensité d'un signal au cours de temps, le nombre maximal de possibles possibles, le rapport minimum entre l'amplitude de signal et la taille des marches calculées (voir 3.1), ainsi que le numéro du graphe (pour la présentation du graphe). On obtient en sortie la liste des hauteurs des segments (c'est-à-dire l'ensemble des $(\mu_k)_k$), le nombre de segments, la liste des abscisses des instants de rupture, ainsi que la valeur de la vraisemblance du modèle choisie.
- Une fonction `FusionALL` qui prend en argument le nom d'un fichier csv contenant en colonne l'ensemble des valeurs temporelles des signaux, le nombre maximal de segments possibles et le ratio minimum pour la taille d'une marche (identiques à celui de `Fusion`). Cette fonction va appelé séquentiellement la fonction `Fusion` pour l'ensemble des signaux, et va renvoyer la liste du nombre de segments pour chaque signal, un tableau contenant la liste des hauteurs de segments, un tableau contenant les instants de rupture, ainsi que 3 listes contenant la variance, la taille, et la vraisemblance au modèle pour chaque signal.

Fichier `"penaltylearning.R"` :

Ce fichier concerne la segmentation avec `penaltyLearning`, et contient 4 fonctions :

- Une fonction `Training` qui permet de générer un modèle de prédiction à partir des données qu'on lui fournit. Il prend en argument le nom du fichier contenant les intensités des signaux qui permettent l'entraînement, le nom du fichier contenant les annotations associés, et le nombre maximal de segments possibles. La fonction ne renvoie rien, mais génère un fichier `"Training.rda"` avec le modèle entraîné.

- Une fonction `Choix_k` qui prend en argument le nom d'un fichier contenant des intensités, le nom d'un modèle de prédiction, et le nombre maximal de segments, et renvoie le nombre de segments prédis pour chaque signaux.
- Deux fonctions `Calcul_Segments` et `Calcul_SegementsALL`, analogues à `Fusion` et `FusionALL` de la partie concernant `Segmentor3IsBack`.

Fichier `"analyse.R"` :

Ce fichier contient des fonctions annexes qui servent entre autre pour l'analyse après les segmentations. Il contient 5 fonctions :

- Une fonction `Marches_coord`, qui à partir de ce que renvoie `FusionALL` ou `Calcul_SegementsALL`, procède au tri sur les signaux mal analysés, et renvoie le nombre final de signaux exploités, deux listes contenant les hauteurs des premières et deuxièmes marches (calculés à partir des hauteurs de segments renvoyés par `FusionALL` ou `Calcul_SegementsALL`), les abscisses des premières et deuxième marches, et les numéros des graphes conservés.
- Les fonctions f_1 et f_2 définis dans la partie 5.2.
- Une fonction `Indice_confiance` qui prend en entrée ce que renvoie `FusionALL` ou `Calcul_SegementsALL` et ce que renvoie `Marches_coord`, et retourne la liste des indices de confiance pour chaque signal.
- Une fonction `affiche_marche` qui permet d'afficher le graphe d'un signal ainsi que sa segmentation associé (la valeur "IC = ..." correspond à l'indice de confiance du modèle)

8.2 Exemple d'analyse

L'ensemble de ces étapes sont reprises dans le fichier `Code/Taunique.rmd`.

1^{re} étape : entraînement du modèle de `penaltyLearning`

Dans le dossier `/Fit`, si le fichier `"training.rda"` n'existe pas, alors il est généré à partir des fichiers `"ann.csv"` et `"data.csv"` avec la fonction `Training` :

```
dossier_fit = "Taunique/Fit/"
train_model = paste(dossier_fit,"training.rda", sep = "")

if(!file.exists(train_model)){
    train_data = paste(dossier_fit,"data.csv", sep = "")
    train_ann = paste(dossier_fit,"ann.csv", sep = "")
    Training(train_data, train_ann, kmax = 4)
}
```

2^e étape : segmentation des signaux à l'aide des deux méthodes

On récupère dans le dossier `/Results` le fichier sortie par la macro sur ImageJ qui nous donne les valeurs d'intensité de l'ensemble des points en fonction du temps, et on éxecute les fonctions `FusionALL` et `Calcul_SegmentsALL` pour obtenir le calcul des segmentations avec les deux pénalités différentes :

```
dossier_results = "Taunique/Results"
test_data = paste(dossier_results, "results.csv", sep = "")

list_k = Choix_k(test_data, train_model, kmax = 4)
resultat1 = Calcul_SegmentsALL(test_data, list_k, kmax = 4, affichage = 0)

resultat2 = FusionALL(test_data, affichage = 0, kmax = 4, ratio = 10)
```

Les variables `resultat1` et `resultat2` contiennent les résultats des deux segmentations.

3^e étape : calcul des indices de confiance et sélection de la meilleure segmentation

Pour les deux méthodes, on calcule les indices de confiances de chaque marches et on compare ces deux nombres pour chaque signaux afin de stocker dans la variable `resultat` le meilleur résultat :

```
marches1 = Marches_coord(resultat1)
marches2 = Marches_coord(resultat2)

IC1 = Indice_confiance(marches1, resultat1)
IC2 = Indice_confiance(marches2, resultat2)
IC = pmax(IC1,IC2)

resultat = resultat1
resultat$nbmarches[IC1 < IC2,] = resultat2$nbmarches[IC1<IC2,]
resultat$moymarches[IC1 < IC2,] = resultat2$moymarches[IC1<IC2,]
resultat$absmarches[IC1 < IC2,] = resultat2$absmarches[IC1<IC2,]

marches = Marches_coord(resultat)
```

Les variables `marches` et `resultat` contiennent l'ensemble des informations nécessaires, il suffit ensuite d'extraire celles que l'on souhaite.

4^e étape : affichage des courbes

Il suffit pour cela d'appeler la fonction `affiche_marche`, avec les variables `marches`, `resultat`, et `data` qui sont les données temporelles initiales.

On peut également n'afficher que les courbes qui ont un indice de confiance supérieur à un certain seuil si on le souhaite :

```
data = read.table(test_data, header=TRUE, sep = ",")
data = data[,-1] #pour enlever la colonne des labels

seuil_confiance = 0.65

for(i in marches$absgraphes){
  if(IC[i]<seuil_confiance){
    affiche_marche(resultat, data, marches, i, IC)
  }
}
```

Littérature

1. Elie et al. (2015) Tau co-organizes dynamic microtubule and actin networks. Sci Rep 5 :9964
2. M. Ovesný, P. Křížek, J. Borkovec, Z. Švindrych, G. M. Hagen. ThunderSTORM : a comprehensive ImageJ plugin for PALM and STORM data analysis and super-resolution imaging. Bioinformatics 30(16) :2389-2390, 2014
3. Izeddin, I. Boulanger, J. Racine, V. Specht, C. G. Kechkar, A. Nair, D. Triller, A. Choquet, D. Dahan, M. and Sibarita, J. B. Wavelet analysis for single molecule localization microscopy. Optics Express 20(3), 2081–95 (2012)
4. Alice Cleynen, Guillem Rigaill and Michel Koskas (2016). Segmentor3IsBack : A Fast Segmentation Algorithm. R package version 2.0.
   `https://CRAN.R-project.org/package=Segmentor3IsBack`
5. Scrucca L., Fop M., Murphy T. B. and Raftery A. E. (2016) mclust 5 : clustering, classification and density estimation using Gaussian finite mixture models The R Journal 8/1, pp. 205-233
6. Toby Dylan Hocking (2017). penaltyLearning : Penalty Learning. R package version 2017.12.08.
   `https://CRAN.R-project.org/package=penaltyLearning`
7. Hocking TD, Rigaill G, Bach F, Vert J-P. Learning sparse penalties for change-point detection using max-margin interval regression. International Conference on Machine Learning (ICML), 2013.
8. Lebarbier, E. Detecting multiple change-points in the mean of gaussian process by model selection. Signal Processing, 85 :717–736, 2005.