**Compte Rendu – Semaines 2 et 3**

**Introduction du projet :**

Nous sommes dans le domaine de la microscopie de fluorescence pour la super résolution et utilisons un microscope STORM. Des fluorophores attachés à des cellules permettent d’en reconstituer l’image 3D, pour cela ils doivent être présents de façon suffisamment dense pour pouvoir résoudre la molécule, et doivent clignoter à tour de rôle.

On nous fournit donc énormément d’images prises au même focus avec des tâches d’Airy à différentes positions correspondant aux fluorophores qui émettent, et éventuellement des images sans tâche ou certaines images avec deux tâches d’Airy proches.

De ces images, nous devons déterminer le positionnement transverse (x,y) des émetteurs et leur positionnement axial (z ou focus), il nous serait possible de l’obtenir à partir de franges issues d’un réseau de diffraction.

**Etape initiale :**

Une première étape consistant en un traitement préliminaire des images afin d’améliorer leur qualité (bruit de fond, pondération du centre, déconvolution…) a été traitée pendant la première semaine et est quasiment terminée.

Quelques outils :

-Le masque hypergaussien permet d’atténuer l’influence du bruit du fond tout en n’ajoutant pas de fréquence aberrante dans l’espace de Fourier car les bords du masque sont lisses.

-Pour le filtre de Wiener on déconvolue par l’image de calibration filtrée, est ce pertinent ? C’est une sorte de tâche d’Airy donc il y a une certaine logique, mais cette tâche est trop grosse (boule 80µm), on pourrait aussi la réduire ou créer une tâche d’Airy correspondant à la réalité : 1.22\*λ/ONimage où ONim=ONobj/72=0.020 et λ=680nm

On a finalement une tâche d’Airy de 41.48µm.

L’image de référence utilisée est aussi une image de calibration, on compare une image de mesure filtrée avec sa calibration filtrée, au focus correspondant.

-Pour les traitements d’image on effectue souvent la soustraction calibration- img\_mesure filtrée, est-ce pertinent ? Le bruit de fond entre calibration et mesure est surement différent de toute façon. Pourquoi ne pas faire img\_mesure\_filtrée –img\_mesure ?

Il y a deux fonctions masque pour le traitement préliminaire des images :

**Masque rephase2** :

1. on enlève le fond (ou la médiane), on masque par une hypergaussienne
2. Puis fftshift fft2

3- On applique un masque hypergaussien dans l’espace de Fourier, largeur 23.064 : img.\*(1-HG) = PASSE HAUT

23.064 est la plus haute fréquence correspondant au microscope.

( Raisonnement pour faire le lien fréquences spatiales – Fourier :

Exemple : fréquence maximale collectée par un microscope

On veut virer toutes les fréquences supérieures car on sait que c’est forcément du bruit

NAobj : ouverture numérique objet du µscope

λ : longueur d’onde de travail

gy: grandissement transversal

On a la période associée à cette fréquence, en mètres.

D est la période associée en pixels, Avec c la taille d’un pixel (c=3,24675µm)

Enfin, l’image faisant NxN pixels, on a l’emplacement de la fréquence d’intérêt par rapport à l’origine des fréquences dans l’espace de Fourier : (N=120)

Ainsi, on peut appliquer un masque circulaire de rayon pf sur la FFT de l’image (avec origine des fréquences au centre) afin de ne conserver que les fréquences provenant du microscope. )

1. Ensuite ifft2
2. Traitement de recentrage. (suppression dephasage)
3. Wiener : utilisation d’une gaussienne centrée en 0 pour enlever la porteuse ? Déjà effectué par le filtre passe haut, deux stratégies valables ? À tester et comparer après avoir créé une métrique
4. Point fort : on peut enlever efficacement la porteuse

Point à éclaircir : Pourquoi utiliser Wiener ?

**Masque rephase 1** :

1. on enlève le fond
2. masquage par hypergauss
3. Traitement de recentrage
4. Wiener déconvolution par calibration

Semble plus efficace que Masque rephase2

Il reste à faire un algorithme de tri des images, permettant d’en extraire des taches d’Airy isolées, tout en conservant la partie de l’image où se trouve l’information sur les franges.

**Détermination de la position :**

**Transverse :**

La position (x,y) se trouve grâce à la fonction fit\_gauss 2D (qui appelle approxgauss2D) qui renvoie les paramètres d’une gaussienne excentrée et avec offset.

**Axiale :**

* ***Par la mesure des interfranges :***

***Traitement préliminaire:***

On utilise find\_the\_gauss qui renvoie les paramètres de la gaussienne du fond (qui nous gêne) .

**1ère solution** : Afin de mesurer l’interfrange, et comme le réseau est tourné de 11.2° nous avons réalisé une fonction calculant la moyenne de plusieurs lignes orientées, on fera ensuite un « watershed » 1D :

**moydeg**(image,d,n,centrex,centrey,pasx,pasy,plot)

% Renvoie les moyennes selon un segment orienté de d degré (sens trigo) d'une image,

% image

% d : angle en DEGRE par rapport à la verticale (sens trigo). -90°<d<90°

% n: nombre de valeurs utilisées dans le calcul de chaque moyenne

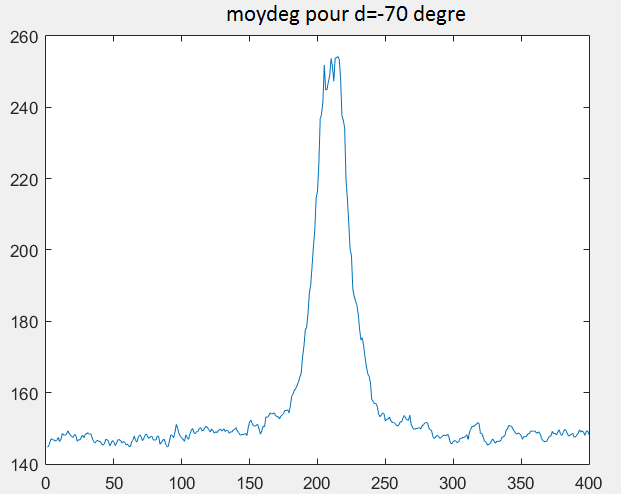
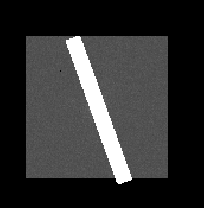
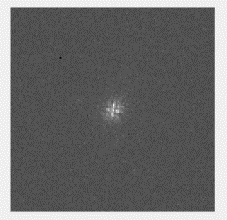
%centre x et centrey : centre du parcours de la moyenne

%pasx: pas de sampling des echantillons des moyennes, vaut 1 si non renseigné

%pasy: pas entre chaque colonne correspondant à une moyenne, vaut 1 si non renseigné

% plot: si plot a une valeur non nulle ou non NULL, la fonction affichera le chemin parcouru

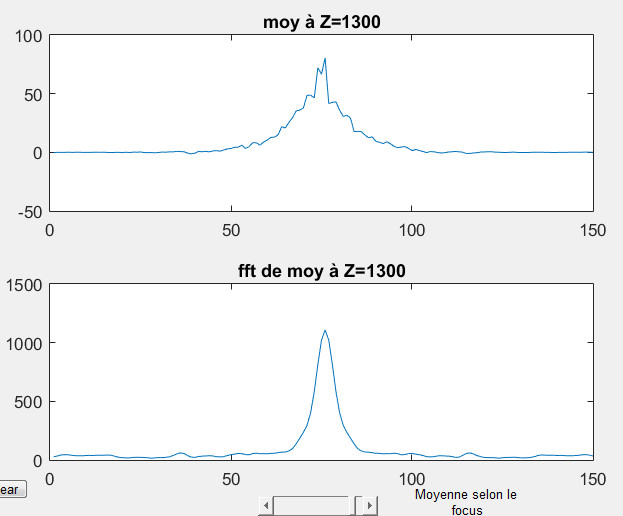
x

Actuellement le parcours effectué (en blanc) dépasse l’image donnée, on rajoute donc un bord, il faudra encore peaufiner cette fonction pour calculer à l’avance les bonnes coordonnées de départ, et la bonne taille du vecteur moy.

+vérifier qu’on passe bien toujours par le centre

La fonction plotmoy avec d=-11.2 et n=4 montre les différentes moyennes pour les focus :



On voit l’existence de l’interfrange, mais celle-ci est dure à obtenir…

**2ème solution** : essayer un algorithme ‘watershed’ de détection de contour directement en 2D.

**3ème solution** : trouver directement la fréquence de l’interfrange dans l’espace de Fourier :

Méthode pour trouver le graphe freq=f(z) :

* Traitement de l’image (masque\_rephase)
* Passage de l’image en fourier (frange\_detect)
* Application du donut (dont les paramètres sont à modifier pour plus de précision/fiabilité, ou simplement à confirmer) (frange\_detect le fait aussi)
* Utilisation de moydeg (résultat renvoyé par frange\_detect)
* Séparation du résultat de moydeg en deux

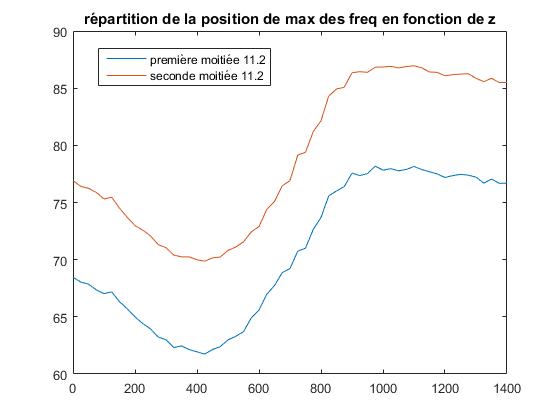
Deux possibilités à partir de là, à vérifier laquelle est la bonne

1. Utilisation de chaque moitié de ce que donne moydeg et moyennage
2. Utilisation du centroid de chaque moitié pour obtenir la valeur du centre de l’image

→ il faut réfléchir à ce souci

* On lisse la courbe donnée par moydeg par la spline (smooth\_max → CreateFitSpl)
* On fit la spline par une gaussienne (smooth\_max → fit\_gauss)
* On récupère la moyenne de la gaussienne et ça nous donne une bonne approximation du centroid de la patate

A ce moment-là, on a donc la fréquence en fourier du centroid de la « patate » (faudrait peut-être lui donner un nom plus sympa que patate. Genre Garry) d’intérêt du spectre de l’image. Comme on fait ça pour chaque image de calibration dont la profondeur de mise au point est connue, on a bien freq=f(z)



**4ème solution :**

* **Seuillage des images de calibration** : plusieurs méthodes déjà testées (**essai\_seuillage.m)**
* Seuillage simple (on fait le seuil par rapport à une valeur fixe) -> pas de surprise, quand la valeur est assez haute, on n’a plus du tout de bruit, et les images semblent exploitables ; MAIS probabilité très forte de perdre des informations en fixant arbitrairement un seuil.
* Combinaison de seuillage dynamique et de détection de contour (on seuille un pixel par rapport à 2 pixels déjà traités de la ligne précédente) -> on distingue la tache centrale, mais l’image n’est pas du tout exploitable
* Seuillage adaptatif (on fait la moyenne avec la fonction **moydeg** de chaque ligne de l’image, on approxime cette moyenne à une gaussienne, puis on compare chaque pixel de la ligne à cette gaussienne ; puis la même chose en alliant les moyennes des 2 lignes perpendiculaires) -> même genre d’image que pour la méthode précédente, pas exploitable
* Seuillage adaptatif en comparant à une approximation polynomiale de l’image (à l’aide de la fonction matlab **fit**) -> le résultat est similaire aux cas précédents, non exploitable dans notre cas
* Seuillage adaptatif en comparant à une approximation gaussienne en 2D de l’image (on utilise la fonction **approxgauss2D**) -> l’image obtenue correspond à ce que l’on attendait : une tache centrale composée de points plus ou moins gros pour les pics. On gardera donc cette méthode de seuillage.
* Avant tout seuillage, on applique un masque hypergaussien (fonction **masque\_rephase**).
* Récupération d’une image ne contenant que les centres des points (pour déterminer les distances entre les points de manière plus précise) à l’aide de la fonction matlab **bwmorph.**
* **Idées pour la prochaine semaine de projet** :
* Retrouver les fonctions matlab permettant de calculer précisément la distance entre les points (=l’interfrange), puis appliquer à cette distance la fonction **calc\_rapport**; ou revenir à la méthode de Killian.

Sinon on dispose d’une fonction approxpol qui approxime des données par un polynôme de degré voulu.

On pourra utiliser la répétition d’image fournie au même focus pour vérifier la répétabilité au même focus de nos idées, et vérifier sa répétabilité sur d’autres focus, ainsi que la cohérence au niveau du résultat. (trouver un écart vers 200 pour Z=700 et Z=900)

Il faudra aussi établir une métrique (critère de qualité)

On pourra envisager plus tard une reconstitution en 3D de la cellule sur une image 3D, qu’on pourrait visualiser en défilant dans les couches.

Reste notamment à faire :

Finir **myui2**, **forui** => éclaircir le début ; faire en sorte de faciliter le changement de l’image à afficher

**Visualisation\_moydeg** : prend un stack et regarde à la période 3px/franges en Fourier pour voir qui est où, trace le graphe de Val (ν) = f(Z). [Après avoir traité l’image, déconvoluée par une gaussienne]

**Création d’une interface homme-machine :**

*Prendre franges en x et y, comme les deux bruits sont indépendants on divise par racine(2)*

*Offset dans la calibration f\_frange(z)*

**program.m :**

UI qui permet de tester nos algorithmes, et peut être utilisé comme programme final :

Quels sont les Besoins ?

Besoin final : trouver pour chaque tâche d’Airy sur une image de mesure la position (x,y,z) associée

Processus :

0-> Traitement des images pour les améliorer, enlever du bruit

1-> dissection des images pour isoler les tâches d’Airy (image résultante toujours 120\*120 ?)

2-> fit gauss de la tâche d’Airy pour trouver (x,y)

3-> on peut trouver z à l’aide à la fois de la variance de la tâche variance (obtenue avec fitgauss), et des interfranges présentent dues au réseau.

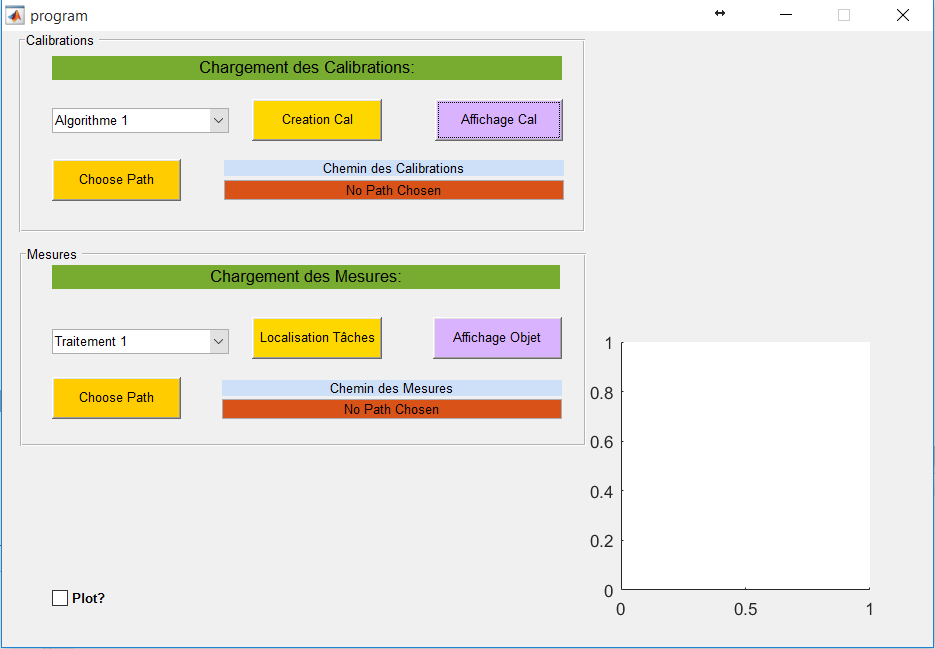
* Nécessité d’une courbe de calibration pour les deux, et de courbes pour deux axes orthogonaux pour limiter le bruit (division par racine2). On va mesurer l’interfrange sur les images de Mesure. Avec la variance et l’interfrange, et par recherche d’antécédent avec les calibrations on va retrouver z.

4-> Positionnement du point sur une image, qu’on peut afficher ou non. Enregistrement dans un fichier.

**Solution de l’ui :**

* Chargement des calibrations : choix du dossier, bouton pour charger
* Création des images de calibration : choix d’un algorithme, bouton pour le lancer, bouton pour afficher les courbes résultantes
* Chargement des mesures : choix du dossier, du nombre de mesures qui nous intéressent, bouton pour charger
* Recherche de la position : choix d’un traitement (ils comprendront chacun quelque chose comme: traitement préliminaire + calcul interfrange variance + utilisation calibrations pour trouver z) , bouton pour le lancer, bouton pour afficher les positions résultantes.
* Sauvegarde : Enregistrement du résultat dans un fichier image ou texte.

Résultat super beau :

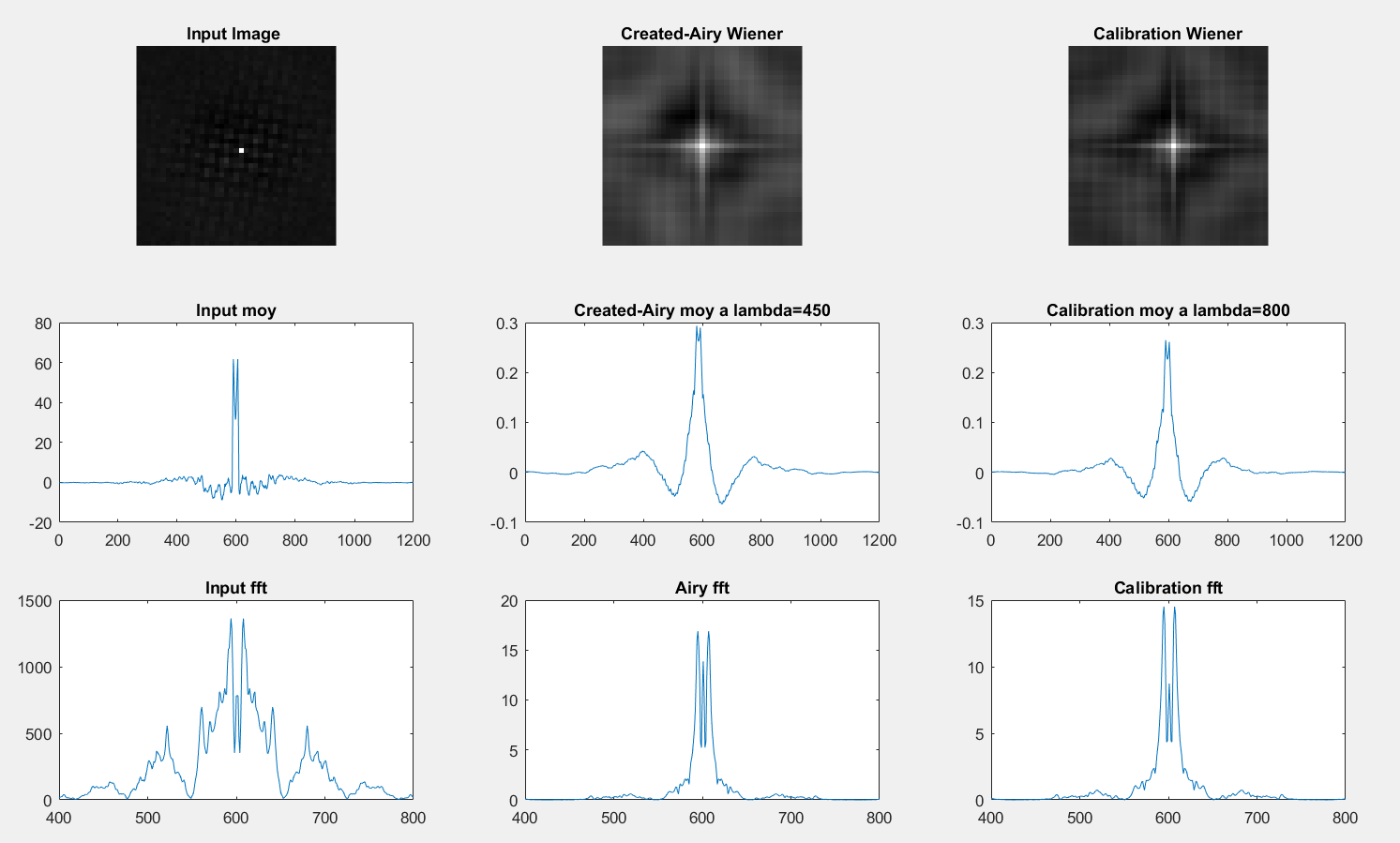


**Test de Wiener :**

Essai de sauvetage de Wiener, le problème étant qu’on ne connait pas l’objet final donc n’a pas de référence pour utiliser minEQM. On choisit soit une référence arbitraire, soit un paramètre lambda arbitraire.

plotWiener permet de faire varier lambda ou le focus, et d’observer les changements.

testWiener affiche ceci :

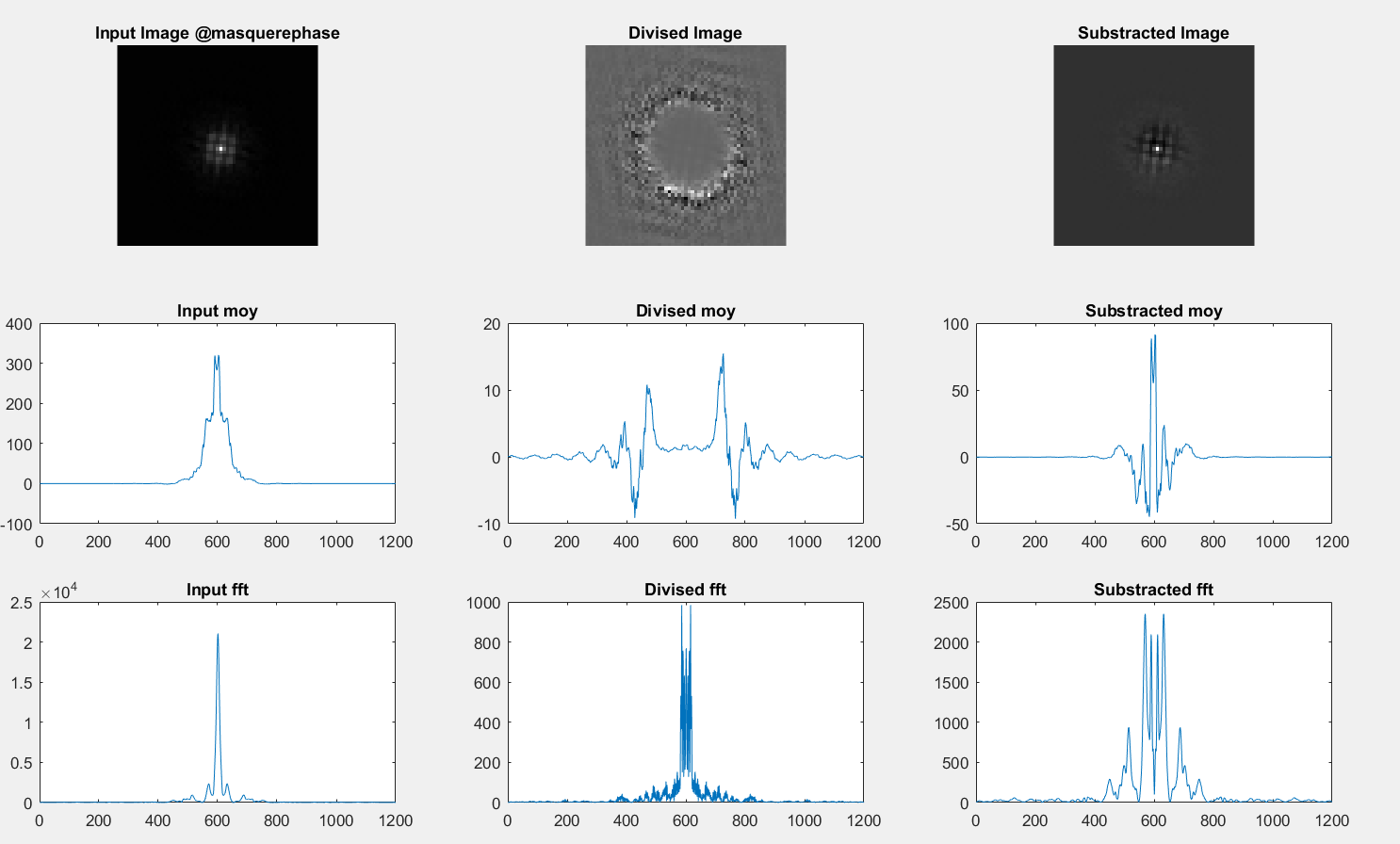


**Removegauss et divisegauss :**

Détecte une gaussienne et l’enlève à l’image. Peut-être pratique pour détecter les interfranges (alternative : méthode du donuts gaussien qui marche bien)

Divisegauss donne un truc étrange… removegauss marche bien !

Pour visualiser le résultat : testenlevegauss :



Méthode avec Fourier : en deux parties

* Fourier

-On utilise une image traitée, on passe en Fourier, on sélectionne un donut autour de la fréquence d’intérêt. On utilise moydeg => on obtient un vecteur contenant les fréquences intéressantes ré-échantillonné (x2).

-On récupère le vecteur qu’on smooth à l’aide d’une spline (createFitSpl) puis fit gaussien pour extraire facilement la valeur du max (fit\_gauss) . (tout ça est fait dans smooth\_max/smooth\_max2)

-On effectue cela pour chaque valeur de z. On obtient donc une courbe de la position de la fréquence d’intérêt en fonction de z.

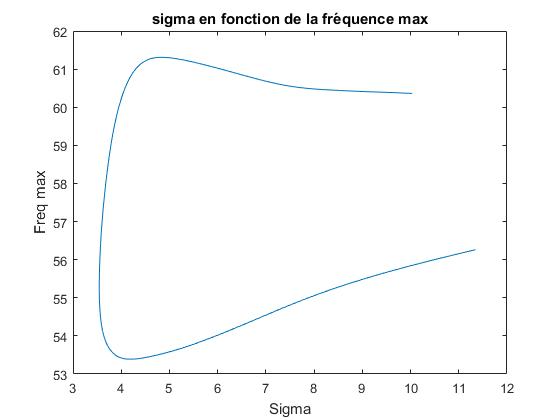
* Airy

-Calcul de la largeur de la tâche d’Airy à l’aide de find\_the\_gauss (qui utilise fit\_gauss2D).

-Construction du graphe de la taille de la tâche en fonction de z.

* Synthèse

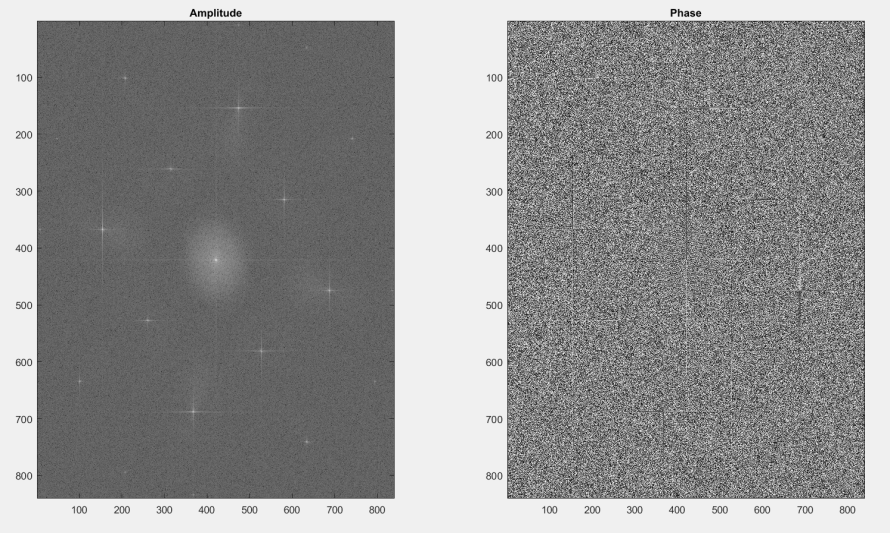
-On smooth les deux courbes obtenues à l’aide de splines, ce qui permettra un ré-échantillonnage

-On trace l’une en fonction de l’autre, c’est ce graphe qui permet de lever l’indétermination : Etant donné les coordonnées d’un point (la taille de sa tâche ainsi que sa fréquence max dans la zone d’intérêt) on trouve le point le plus proche sur la courbe qui lui correspond [ce qui reste à faire].

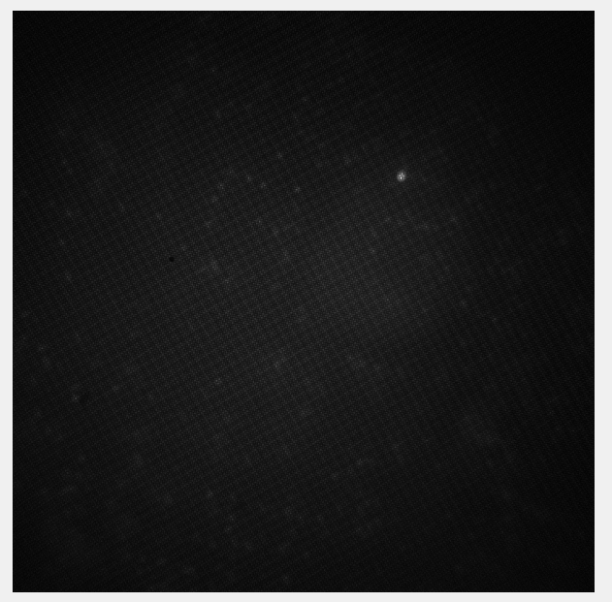
Contribution d’Adrien – semaine 4:

Modification partielle du GUI pour le rendre plus esthétique

Ajout de fonctions utiles telles que imshowf, pour visualiser les images en Fourier :

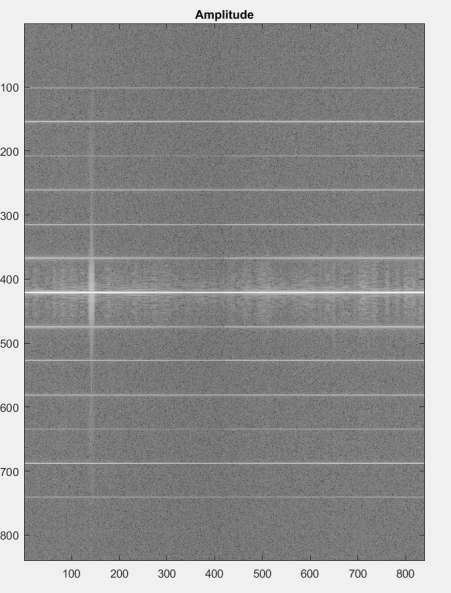
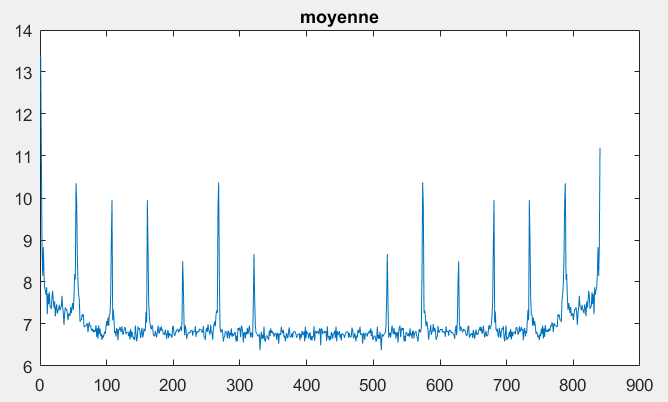
 *(fft2 de nos images expérimentales)*

Ajout aussi de meansquare (renvoie l’ecart quad. moyen entre deux images).  
  
Le reste de la problématique porte sur le traitement d’images expérimentales de mesures, avec plusieurs ou aucune tâche d’Airy.  
 On a tout d’abord un fort fond lumineux qui créé un interferogramme dont on veut se débarasser, tout en ne dégradant pas l’information sur la fréquence entourant les tâches des fluorophores.

(Note : La tâche blanche n’est pas due à un fluorophore, et certaines tâches noires sont dûes à des poussières sur la caméra. )

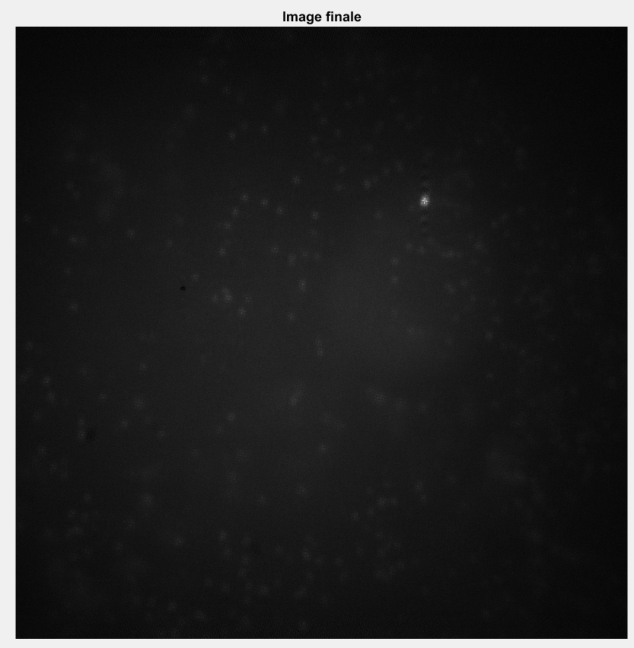
L’idée est de passer par la transformée de M.Fourier pour détecter les pics de fréquence correspondants, on peut le faire en fft2 mais ces pics se détectent mieux en fft (question : fft induit-elle une erreur ?)

**

*En appliquant moyhor sur le log(fft(img)+1) on a des pics très contrastés.*

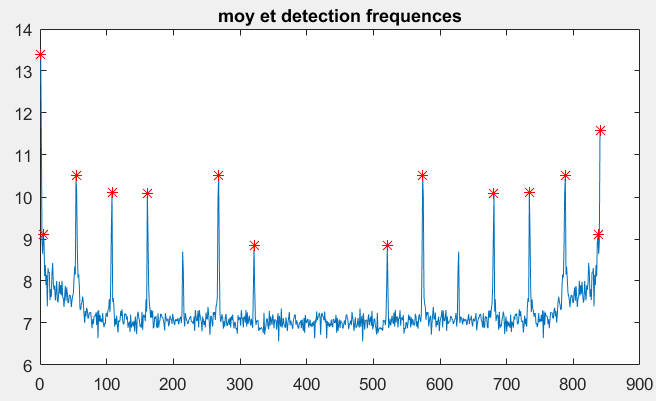
Le nombre de pics à enlever est identifiable sur l’image fft2 : 1pic central + 8 pics d’ordre 1 + 4 pics d’ordre supérieur. On les repère bien sur l’image moyenne.

L’algorithme marche pour des fréquences à éliminer déjà rentrée (54,108 …) :

On peut deviner l’interferogramme propre à chaque tâche. Pour utiliser cette algorithmique il faut donner la valeur 0 au booléen ‘auto’

J’ai essayé de rendre la détection de ces fréquences automatique, mais ça n’est pas toujours évident.  
On va utiliser une fonction externe : extrema qui délivre les extremums et leurs positions.

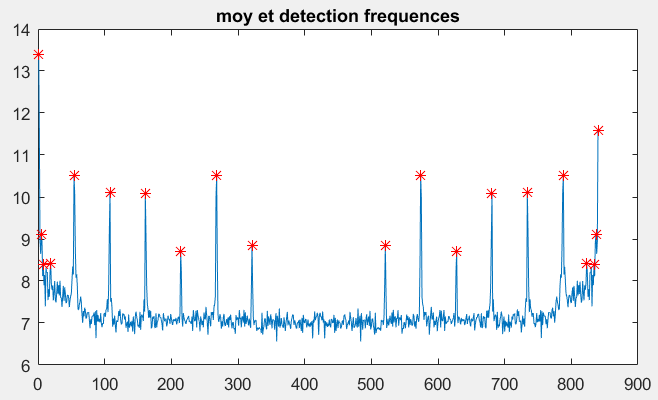
On peut rencontrer ce type de problème :  
  
  
*Ici j’ai fixé le nombre de pics à 14 et gardé les 14 plus élevés.*

On détecte **des pics parasites**, majoritairement dûs au centre et rate les pics alentours  
  
On peut augmenter légèrement le nombre de pics cherchés par rapport au nombre de pic final voulu, , et appliquer plusieurs idées (simultanément ou non):

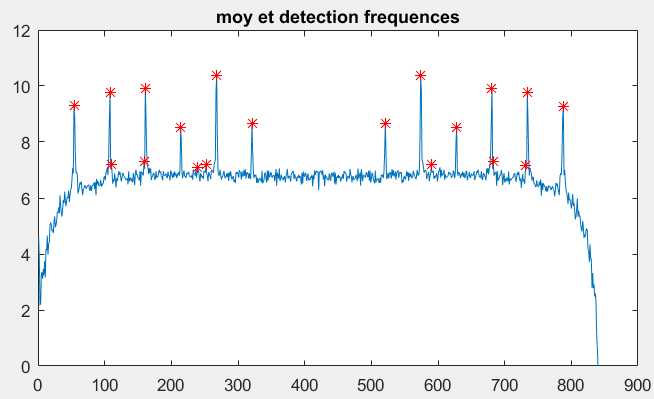
1) on garde les pics qui sont bien multiples d’une même période. (ici notre période type est 54).   
Comme chacune des fréquences à trouver est espacée du même Δf, ceci en partant de 0, on peut facilement détecter les valeurs aberrantes.  
2) on se débarrasse du pic central par exemple en lissant spécifiquement le centre  
3) Après coup, on ne garde pas les pics proches de 0 obtenus (définir une limite…)

4) Une autre idée serait de ne plus utiliser directement la fonction extrema mais d’appliquer un filtre de dérivation, en smoothant ou non.

**Idée : les pics parasites sont tous sur le pic central :**  
Ceci correspond aux idées 2) et 3), en augmentant le nombre de pics détectés, on trouve tous les pics parasites proches du centre :



On va rajouter un coefficient ‘signore’ et multiplier l’image sur laquelle on fait la moyenne (pas l’image de mesure) par (1-hypergaussienne(0,0,…,ignore…) ). C’est une hypergaussienne centrée en 0, d’ecart type signore et étalée horizontalement (pour correspondre à fft)

Pour une hypergaussienne d’ecart type 1.5 et de puissance 5, on fait bien ressortir les 12 pics qui nous intéressent :  
  *moyhor sur echelle logarithmique*

Ici on peut chercher directement le bon nombre de pics et pas plus.  
Optimisation possible : On pourrait peut-être gagner du temps (sur la création de Hgaussp et multiplication de img) avec l’idée 3 en cherchant plus de pics que besoin et en enlevant ceux sous 30.

En travaillant avec une autre transformée qui ne donne qu’un pic pour un sinus (et non deux), on peut gagner du temps de calcul (on crée deux fois moins d’hypergausiennes)