M1 Imageurs, Poitiers 2023

Introduction aux macros ImageJ

Fabrice P. Cordelières

15 mars 2023

Matériel de cours
disponible sur GitHub

Corrigés en pièces jointes au pdf

Table des matières

Chapitre 1: Les bases du langage macro ImageJ

Ce premier chapitre est destiné à vous montrer les points communs, mais surtout les différences entre les langages de programmation avancés que vous pouvez connaître et le langage de scripting propre à ImageJ. Le langage macro, quoique moins puissant et moins formel qu'un vrai langage de programmation, permet une automatisation rapide de tâches, soit pour une utilisation directe, soit pour le prototypage de tâches complexes qui pourront faire ultérieurement l'objet de modules spécifiques, codés en Java.

L'outil de base

Le Macro Recorder

Le `Macro Recorder` permet d'enregistrer (presque) toutes les opérations réalisées par l'utilisateur. Il suffit de le lancer pour voir se créer une série d'instructions permettant de reproduire le protocole de traitement et d'analyse sur une autre image. Il se trouve dans le menu `Plugins>Macros>Record` et fonctionne de la même manière sous ImageJ et Fiji.

Éditer une séquence et la sauvegarder : .txt ou .ijm ?

Bien que permettant l'édition des instructions, le `Macro Recorder` n'est pas l'éditeur à proprement parler des macros ImageJ. Pour le faire apparaître, il suffit de cliquer sur le bouton `Create`.

Un éditeur plus complet s'affiche, différent suivant qu'on utilise ImageJ ou Fiji. Sous ImageJ, l'éditeur est un simple éditeur de texte, sans coloration syntaxique. En revanche, il embarque quelques fonctionnalités intéressantes :

- Un outil de déboggage, accessible via le menu `Debug`. Il permet notamment l'exécution pas-à-pas de la macro.
- Le `Function Finder`, accessible via le menu `Macros>Functions Finder...` : il permet de visualiser et de réaliser une recherche dans l'ensemble des fonctions macros. Le menu `Help>Macro Functions...` renvoie vers la **page de référence du site d'ImageJ.**

Sous Fiji, l'éditeur de macros offre une coloration syntaxique. En revanche, le `Function Finder` n'est pas disponible : il est remplacé par l'auto-complétion et une aide contextuelle. Les outils de deboggage/d'exécution pas-à-pas ne sont pas disponibles. Une astuce permet néanmoins de faire apparaître l'éditeur de macros ImageJ sous Fiji : il suffit pour d'ouvrir le fichier macro après en avoir ôté l'extension.

Justement, quelle extension pour nos fichiers macros ? Deux conventions sont possibles : au final, le fichier est un simple fichier texte. Peu importe, on pourra utiliser au choix txt ou ijm (Image J Macro).

Exécuter une macro ou la stopper

Sous ImageJ, pour exécuter une macro, depuis la fenêtre de l'éditeur, aller dans le menu `Macros>Run Macro` (raccourci clavier `CTRL` ou `CMD+R`). Pour en stopper l'exécution, presser la touche `Esc`.

Pour exécuter une macro sous Fiji, utiliser le bouton `Run` en bas de la fenêtre de l'éditeur ou le menu `Macros>Run Macro` (raccourci clavier `CTRL` ou `CMD+R`). Pour en stopper l'exécution, presser la touche `Esc` ou le bouton `Kill` en bas de la fenêtre de l'éditeur.

Les fonctions intégrées

Structure des fonctions

Plusieurs structures de commandes co-existent :

- `run("Nom_De_Commande", "Arguments")` : mot-clé run prenant 2 arguments 1-le nom de la fonction, 2-une chaîne de caractères contenant les arguments, séparés par des espaces ;
- `nomDeCommande("Nom_De_Commande")` : nom de la fonction, suivi d'une chaîne de caractères contenant les arguments.

Ce ne sont que 2 des formes des commandes macros : il n'y a pas de consensus sur la structure des fonctions. La raison est historique : les fonctions en "run" correspondent aux premières fonctions implémentées à l'origine du logiciel, les fonctions plus récentes reprenant une syntaxe plus proche du Java.

Trois logiques différentes

Les instructions peuvent adopter trois types de fonctionnement différents qu'il est important de distinguer :

- **Les méthodes :** ce sont des fonctions qui exécutent une opération ex : redimensionner une image, ajouter une coupe à une pile etc ;
- **Les fonctions :** elles renvoient un résultat qui peut être stocké dans une variable ;
- **Les méthodes de collecte :** c'est le cas par exemple de la fonction `getStatistics(area, mean, min, max, std, histogram)` dont l'appel stocke dans les variables fournies en arguments les informations demandées.

NB : Tout comme en Java, le ; est obligatoire, utilisé pour marquer la fin d'une instruction.

Éléments de syntaxe

Trouver ses marques avec ce nouveau langage

A l'évidence, pour des tâches simples, le `Macro Recorder` peut suffire. En revanche, dès lors qu'il faudra modifier dynamiquement un paramètre, boucler un processus sur un ensemble d'éléments ou adapter le comportement de la macro en fonction d'une entrée, il deviendra nécessaire de connaître la syntaxe d'un ensemble d'éléments de base que sont les variables, les boucles et les structures de choix. Comme dans tout langage de programmation, il sera également important de documenter son code : il est nécessaire de connaître les balises à utiliser.

Commenter son code : quelles balises ?

Les balises de commentaires sont identiques aux balises utilisées en Java :

- **Les commentaires courts :** ils sont délimités par une seule balise, le `//` ;
- **Les commentaires longs :** la balise ouvrante est `/*`, la balise fermante est `*/`.

Les variables : types, opérateurs et portée

Types de variables

Dans le langage Macros ImageJ, les variables ne sont pas typées. En revanche, trois catégories de variables sont utilisables :

- **Les variables simples :** elles sont déclarées par leur nom. L'attribution d'une valeur se fait au moyen du signe égal. Les guillemets permettent de définir la nature "chaîne de caractère" du contenu. ex : `maVariable=3;` ou `monAutreVariable="chaîne de caractères";;`
- **Les variables de type tableau :** elles sont déclarées par leur nom et l'attribution (=) suivi du mot-clé `newArray()`. Cette instruction peut prendre trois types d'arguments (voir ci-après). *Attention : le langage Macros ImageJ ne permet pas de créer de tableaux multi-dimensionnels.*
  - `monTableau=newArray();` : le tableau est initialisé mais n'a pas de dimension ;
  - `monTableau=newArray(n);` : le tableau est initialisé sans contenu mais à une dimension de "n" cases ;
  - `monTableau=newArray(1, 2, 3);` : le tableau est initialisé et dimensionné en fonction du contenu. Ses cases sont initialisée telles que `monTableau[0]` contient la valeur 1, `monTableau[1]` contient la valeur 2 etc. On peut déterminer la dimension du tableau en ayant recours à l'instruction `monTableau.length`.
- **La liste de couples clé-valeur :** On peut utiliser *une unique liste de clés-valeurs*. Voici une liste des fonctions les plus utiles :
  - `List.clear()` : efface le contenu de la liste actuelle ;
  - `List.size` : renvoie le nombre d'éléments dans la liste actuelle ;
  - `List.set(cle, valeur)` : ajoute ou modifie la paire clé-valeur dans la liste ;
  - `List.get(cle)` : renvoie la valeur associée à la clé en tant que chaîne de caractères, ou une chaîne vide si la clé n'existe pas dans la liste ;
  - `List.getValue(cle)` : renvoie la valeur numérique associée à la clé. Si elle est absente de la liste ou n'est pas une valeur numérique, l'exécution de la macro est stoppée.

Opérateurs

Les opérateurs applicables aux variables sont résumés dans le tableau pages suivante. Certaines opérations entre variables de nature différentes aboutissent à un transtypage implicite en chaînes de caractères au moment où l'opération est réalisée.

```
myVariable1 = 1
myVariable2 = " variable de type chaîne"
myVariable3 = myVariable1 + myVariable2

print(myVariable3)
```

L'exécution de la macro renvoie "1 variable de type chaîne" dans la fenêtre `Log`

Opérateur	Priorité	Description
++	1	Pré ou post incrément
−−	1	Pré ou post décrément
−	1	Soustraction bit à bit
!	1	Complémentaire du booléen
~	1	Bit complémentaire
*	2	Multiplication
/	2	Division
%	2	Reste entier de la division
|	2	Ou logique bitwise (OR, opération sur les bits)
^	2	Ou exclusif logique (XOR, opération sur les bits)
<<, >>	2	Rotation de bit (vers la gauche ou la droite)
+	3	Addition arithmétique ou concaténation de chaînes de caractères
−	3	Soustraction arithmétique
<, <=	4	Comparaisons : plus petit que, plus petit ou égal à
>, >=	4	Comparaisons : plus grand que, plus grand ou égal à
==, !=	4	Comparaisons : égal à, différent de
&&	5	Et logique (AND, opération sur les booléens)
||	5	Ou logique bitwise (OR, opération sur les booléens)
=	6	Attribution
+=, -=, *=, /=	6	Attribution combinée à une opération

Portée des variables

Le langage Macros ImageJ permet de créer ses propres groupes fonctionnels (voir le Chapitre 3 - Groupes fonctionnels). Cette possibilité permet de mieux organiser son code et d'en réutiliser certaines parties dans plusieurs macros sans pour autant avoir à tout re-coder. Ces groupes peuvent nécessiter la déclaration de variables en leur coeur : leur contenu ne sera alors pas disponible à l'extérieur de la fonction à moins qu'un retour explicite soit implémenté. De même, une variable déclarée en dehors de la fonction définie par l'utilisateur n'y sera pas directement utilisable : il faudra alors passer son contenu en argument, au moment de l'appel.

Une autre stratégie est envisageable : modifier la portée des variables. L'utilisation du mot-clé `var` devant une variable placée dans le corps de la macro (plutôt en en-tête, pour plus de lisibilité) permet de la transformer en **variable globale**, accessible depuis n'importe quel point du code.

Les boucles

Boucle définie à priori : for

Sa structure est identique à celle utilisée en Java, à la différence qu'il n'est pas possible d'utiliser un itérateur sur une liste :

```
for(initialisation; condition de poursuite; itération) {
    //Instructions
}
```

Boucle indéfinie à priori : while

Sa structure est identique à celle utilisée en Java :

```
while(condition) {
    //Instructions
}
```

Boucle indéfinie à posteriori : do/while

Cette boucle est exécutée au moins une fois, l'évaluation de la condition s'effectuant à la fin du bloc. Il se déclare comme suit :

```
do {
    //Instructions
} while (condition);
```

Structure de choix

Une seule structure de choix est disponible : la structure if/then/else - else if :

```
if(condition1) {
    //Instructions 1
}else{
    //Instructions 2
}else if(condition2){
    //Instructions 3
}
```

NB : Inutile d'essayer, la structure `switch/case` n'est malheureusement pas implémentée...

Chapitre 2: Premiers pas avec le langage macro ImageJ

Au cours de cette première séquence nous allons explorer le langage macro et découvrir les outils à disposition pour résoudre la problématique posée (voir l'encart "problématique").

Par où commencer ?

Ayant pris connaissance de la problématique, deux étapes devront être réalisées :

1. Identifier les étapes du protocole de traitement et d'analyse ;
2. Identifier les instructions clés à utiliser.

Les étapes du protocole de traitement et d'analyse

L'utilisateur a fourni un protocole détaillé. Le plus simple est donc de reprendre chaque étape proposée et d'initier la macro en utilisant chaque item comme un commentaire. L'espace laissé libre entre chaque commentaire sera progressivement complété avec les instructions adéquates.

Étape 01 : À vous de jouer !

A faire :

- Créer une nouvelle macro : `Plugins>New>Macro`
- En utilisant la syntaxe décrite au "Chapitre 1 - Commenter son code", créer un commentaire par étape de traitement

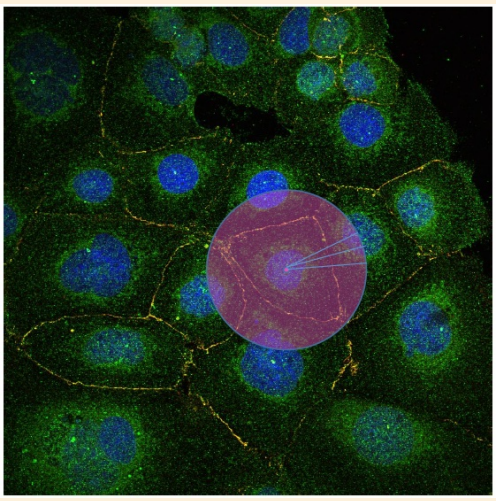
Identifier les instructions clés à utiliser

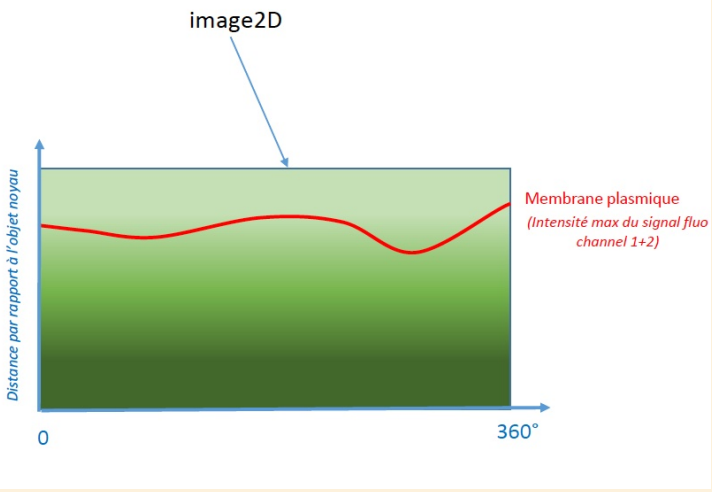
ImageJ dispose d'une fonction permettant d'enregistrement des opérations réalisées sous la forme d'un script : c'est le `Macro Recorder`, activé au moyen du menu `Plugins>Macros>Record...`.

Certaines fonctions ne sont pas enregistrables : en complément au `Macro Recorder` il existe une *page de référence* listant l'ensemble des fonctions accessibles en scripting. En outre, nombre de fonctions documentées incluent des exemples d'utilisation.

D'ores et déjà, au moment de l'enregistrement des instructions, il est important d'anticiper deux points en se posant deux questions. *De quelle situation partons-nous ?* En d'autres termes, une analyse précédente aura-t-elle une incidence sur la nouvelle analyse, en laissant, par exemple, des

La problématique





Besoin : L'utilisatrice dispose d'images 3 canaux :DAPI/TRP/ZO1. Elle souhaite

- Identifier les noyaux individuels et en définir le centre
- Pour chaque noyau, tracer des rayons partants du centre, vers la périphérie (distance à définir), à angle variable (de $0°$ à $360°$, par pas de $1°$).
- Déterminer le profil d'intensité le long de chaque rayon et créer pour tous les angles une image du profil en fonction de l'angle.
- Moyenner les images "dépliées" pour l'ensemble des cellules.
- Créer un graphique de l'intensité moyenne$\pm$deviation standard pour toutes les cellules en fonction de la distance au noyau.

éléments pouvant compromettre l'analyse courante. *L'analyse actuelle est-elle généralisable ?* Les dénominations des éléments manipulés sont-elles fixes ? Que se passe-t-il si le nom de l'image est différent ? Il est important de prêter attention aux arguments des fonctions utilisées et à la manière dont on pourra les manipuler ultérieurement.

Étape 02 : à vous de jouer !

A faire :

- Activer le `Macro Recorder` `Plugins>Macros>Record...`
- Réaliser manuellement les étapes de traitement de l'image, pour le moment en 2D :
  1. Ouvrir une image exemple : `File>Open...`
  2. Dupliquer uniquement le canal 1, correspondant aux noyaux : `Image>Duplicate...` en donnante pour titre au duplicata "Noyaux" et en décochant "Duplicate hyperstack".
  3. Utiliser l'outil de seuillage `Image>Adjust>Threshold...` pour obtenir une segmentation automatique des noyaux, basée sur l'intensité. La méthode "Triangle" semble bien fonctionner. Attention, les noyaux doivent apparaître en rouge sur fond noir : si ce n'est pas le cas, cocher la case "Dark Background".
  4. Pour le moment, les pixels ont été partitionné en deux catégories : positifs/négatifs. Il faut à présent grouper les pixels positifs en objets dont on va déterminer le centre. Utiliser `Analyze>Set measurements...` pour déterminer les paramètres à déterminer. Cocher la case "Centroids" afin de déterminer les coordonnées du centre géométrique de chaque noyau.
  5. Utiliser `Analyze>Analyze articles...` pour grouper les pixels positifs en objets et en déterminer le centre : mettre pour filtre de taille "50-Infinity" (en microns), cocher les cases "Display Results" pour afficher un tableau de résultats, "Clear Results" pour effacer un éventuel tableau précédent et "Exclude on edges" pour ne pas prendre en compte les noyaux coupés sur les bords de l'image.

Tester la macro

La première version de la macro étant prête, il est possible de la tester en la lançant depuis l'éditeur, soit au moyen du bouton `Run` soit du menu `Run>Run`. A noter, il est possible de ne sélectionner qu'une partie du code et de n'exécuter que les lignes concernées au moyen du menu `Run>Run selected code`.

En cas de problème, quelques points à vérifier :

- Manque-t-il un point-virgule pour terminer la ligne précédent la ligne donnée en erreur ?
- Le nom des éléments à manipuler correspondent-ils aux éléments effectivement présents à l'écran ? Vérifier, par exemple, le nom des images à l'écran et les confronter aux noms des images attendus par la macro
- Pour les tables de résultats et le ROI Manager, en cas d'éléments surnuméraires : dans quel état était la ResultsTable/ROI Manager au démarrage de la macro ?
- L'éditeur de macro permet de vérifier les parenthèses, accolades et crochets : chaque balise ouvrante est-elle accompagnée d'une balise fermante ?
- L'instruction qui pose problème est-elle bien orthographiée ? On peut vérifier en enregistrant l'instruction ou en se référant à *cette page*.
- La syntaxe de l'instruction qui pose problème est-elle correcte ? Manque-t-il des arguments ? La nature des arguments est-elle la bonne ? On peut vérifier en enregistrant l'instruction ou en se référant à *cette page*.
- De quelle couleur apparaît l'instruction ? Les arguments ? L'éditeur utilise une coloration basée sur la syntaxe : les commentaires apparaissent en vert, les instructions en orange, les chaînes de caractères en rose et les nombres en violet. Si la couleur n'est pas la couleur attendue, il y a sans doute un problème.

Ajouter un peu de souplesse à la macro : utilisation de variables

En regardant le code de la macro, on peut identifier plusieurs paramètres pouvant varier d'une exécution à l'autre :

- Le recorder a enregistré l'ouverture de l'image : on pourrait effacer cette ligne, supposant que l'utilisateur aura ouvert une image en préalable. Il faudra gérer une exception : l'absence d'ouverture d'image. Il existe sûrement une fonction renvoyant le nombre d'images ouvertes, et une fonction arrêtant la macro si la condition n'est pas remplie.
- L'instruction `selectWindow` est utile puisqu'elle permet d'activer l'image sur laquelle travailler. Encore faut-il savoir sur quelle image le traitement doit s'appliquer. Est-elle utile ici, alors que l'on suppose que l'image ouverte est justement active ?
- Si l'on regarde la ligne `run("Analyze Particles...", "size=50.00-Infinity` on se rend compte qu'un seul argument est présent, permettant de régler les paramètres de détection. Si la taille des noyaux devait varier d'une image à l'autre, il serait utile de pouvoir remplacer la valeur limite de $50 \mu^2 par une valeur modifiable par l'utilisateur, sans avoir à modifier$

Étape 03 : à vous de jouer !

A faire :

1. Supprimer les instructions d'ouverture et de sélection de l'image.
2. En vous aidant de la *documentation en ligne*, gérer l'exception liée à l'absence d'image.
3. Créer la variable `minSize` contenant la taille minimale de noyau.
4. Modifier l'argument du bloc `Analyze Particles` afin de prendre en compte le contenu de la variable.
5. Tester la macro.

Chapitre 3: Fonctionnalisation de la macro et création d'images dépliées

Le mot-clé de la seconde séquence sera "`function`". Nous allons tirer profit des possibilités de fonctionnalisation qu'offre le code macro. Au-delà du gain en lisibilité du code, la création de fonctions personnalisées permettra une réutilisation simplifiée de pans entiers d'instructions entre macros.

Groupes fonctionnels

Utilité des fonctions

Les groupes fonctionnels sont ce que les chapitres sont à un livre : une manière d'organiser les instructions par thématique. Le corps du code apparaît alors comme une table des matières : elle référence l'ensemble des chapitres et permet un accès direct aux sections d'intérêt.

Pour un langage aussi simple que le langage macro ImageJ, outre leur avantage structurant, les fonctions personnalisées généralistes sont utiles dans le cadre d'une réutilisation de blocs, diminuant le temps nécessaire au développement, pourvu qu'elles aient été bien réfléchies en amont.

Syntaxe des fonctions

Une fonction est déclarée par le mot-clé `function` suivi du nom de la fonction. Ce nom doit être suffisamment explicite. Par convention, on utilise la notation chameau employée en Java (les mots sont collés les uns aux autres, leur première lettre est capitalisée à l'exception de la lettre initiale). Il doit obligatoirement commencé par une lettre mais peut comporter des chiffres.

Le nom est suivi de deux parenthèses, ouvrante et fermante. Entre les parenthèses peuvent être déclarés des **arguments**, facultatifs : une ou plusieurs variables sont alors fournies entre parenthèses. La portée de ces variables est locale : elles ne sont utilisables qu'à l'intérieur de la fonction.

Les **instructions** définissant les opérations réalisées par la fonction sont placées entre **accolades**. Une fonction particulière, facultative, peut également y être présente : le mot-clé `return` permet de renvoyer le contenu d'une unique variable à l'extérieur de la fonction suite à son appel.

L'appel à la fonction se fait simplement en utilisant son nom et fournissant les arguments nécessaires. Si la fonction renvoie un résultat, il peut être attribué à une variable pour utilisation ultérieure.

```
//Appel à la fonction
resultat=maFonction(1, "arg2");

//Déclaration de la fonction
function maFonction(arg1, arg2, ...){
   //Instruction 1
   //Instruction 2
   ...
   return valeurDeSortie;
}
```

Différents types de fonctions

Les fonctions disposent de deux lots d'éléments facultatifs : les arguments et le retour. Sur la base de ce dernier, on distinguera deux grands types de fonctions : les **méthodes** qui ne disposent pas de l'élément "`return`" (le retour est `null`) et les **fonctions** qui l'utilisent pour renvoyer une unique variable.

Attention : le fait que le mot-clé ne renvoie le contenu que d'une variable ne signifie pas pour autant qu'on ne puisse renvoyer qu'une valeur. On peut, par exemple utiliser une variable de type tableau pour renvoyer un ensemble d'éléments.

Fonctionnalisation de la macro

Le code de l'**étape 03** génère un tableau de résultats à partir d'une image et d'une valeur limite de taille d'objet. Il peut être empaqueté au sein d'un groupe fonctionnel simple, ne prenant qu'un seul argument et opérant l'analyse.

Étape 04 : à vous de jouer !

A faire : Adapter le code de manière à fonctionnaliser l'extraction des coordonnées des noyaux :

- Créer la fonction `detectNuclei`.
- Ne pas oublier de déclarer son argument.
- Testez la macro
- NB : pour qu'une fonction réalise l'opération souhaitée, il faut l'appeler...

Créer les images dépliées des noyaux

Coder la brique élémentaire : gérer un noyau

Une fois les coordonnées des noyaux connues, il faut s'attaquer au problème de la création des images dépliées. Plutôt que de se lancer directement sur l'ensemble des noyaux, nous allons coder l'opération séparément, pour un noyau.

Le plus simple est de créer manuellement une image qui servira de destination pour les opérations que nous allons réaliser. Cette image aura pour largeur 360 pixels, pour représenter les profils obtenus entre 0 et 359° autour du noyau, sa hauteur sera d'un nombre de pixels équivalent à la longueur que l'utilisateur souhaite pour chaque profil.

Il faudra trouver un moyen de tracer une ligne sur l'image, depuis le centre du noyau vers la périphérie, sous un certain angle, et de rayon défini par l'utilisateur. Si le point de départ est connu, un peu de trigonométrie sera nécessaire pour calculer le point d'arrivée.

La *documentation en ligne* permet de prendre connaissance d'une instruction permettant de rapatrier les valeurs d'intensité le long d'une région d'intérêt : elle sera utile pour connaître les intensités à placer sur l'image de destination. Attention, le contenu renvoyé par la-dite fonction est un tableau : il faudra le stocker dans une fonction et se référer au chapitre 1 pour savoir comment l'utiliser. Une fois ces valeurs obtenues, il faudra inscrire un à un les pixels sur l'image des destination.

Bien évidemment, créer un bloc fonctionnel pour l'ensemble de ces instructions serait un plus !

Étape 05 : à vous de jouer !

A faire : créer une nouvelle macro permettant de réaliser une image dépliée à partir des coordonnées d'un noyau et du nom de l'image à analyser.

1. Créer la fonction `unwrapImage` : elle prend les arguments suivants :
   - L'image sur laquelle réaliser l'analyse, `img`.
   - Les coordonnées du noyau, `xCent` et `yCent`.
   - Le rayon d'analyse, `dMax`.
2. Sélectionner l'image `img`.
3. Attention : les coordonnées déterminées par le module d'analyse sont exprimées en microns ! Pour le tracé de ligne, on s'attend à des pixels : une conversion s'impose... Il faudra trouver un moyen de déterminer et de stocker la taille du pixel ! La *documentation en ligne* peut aider...
4. Créer une nouvelle image nommée `Out` de taille 360x`dMax` pixels.
5. Pour une série d'angles de 0° à 360° par pas de 1° :
   - Sélectionner l'image `img`.
   - Calculer les coordonnées x1, y1 de départ par simple conversion microns>pixels, les coordonnées x2, y2 d'arrivée par trigonométrie (attention, il y a une translation à prendre en compte !).
   - Créer une ligne en utilisant les 2 couples de coordonnées précédent.
   - Stocker les valeurs d'intensités relevées le long du profil d'intensité dans une variable de type tableau, nommée `values`.
   - Sélectionner l'image `Out`.
   - Pour chaque valeur stocker dans `values`, ajuster l'intensité du pixel pertinent sur l'image `Out`.
6. Ajuster l'affichage de l'image `Out` en procédant à un réglage automatique du contraste.
7. Tester la macro.
8. NB : pour qu'une fonction réalise l'opération souhaitée, il faut l'appeler...
9. NB2 : l'exécution prend du temps du fait de la nécessité de mettre à jour l'affichage. Il existe sans doute des moyens d'éviter cela...

Généraliser à l'ensemble des noyaux

La routine de pré-traitement fonctionne, celle de génération de la vue dépliée pour un noyau aussi. Il nous faut à présent connecter les deux et appliquer la fonction `unwrapImage` à l'ensemble des noyaux. Les coordonnées d'intérêt sont stockées dans un tableau de résultats : il faudra en extraire les contenus des deux colonnes d'intérêt, nommées X et Y. Si l'image de base comporte de nombreux noyaux, il sera important de veiller à définir une convention de nommage, permettant de naviguer rapidement entre les images. Il existe d'ailleurs une fonction, dans ImageJ/Fiji, permettant de compiler l'ensemble des images ouvertes sous la forme d'une pile : `Image/Stacks/Images to Stack`.

Étape 06 : à vous de jouer !

A faire : créer une nouvelle macro assemblant les fonctions `detectNuclei` et `unwrapImage`, implémenter la fonction `unwrapAllNuclei` appliquant les fonctions nécessaires à l'ensemble des noyaux.

1. Créer la fonction `unwrapAllNuclei` : elle prend les arguments suivants :
   - Le nom de l'image à analyser, `img`.
   - La taille minimale d'un noyau, `minSize`.
   - Le numéro du canal sur lequel procéder à l'analyse, `channel`.
   - Le rayon d'analyse, `dMax`.
2. Sélectionner l'image `img`.
3. Détecter les noyaux en utilisant la fonction `detectNuclei`.
4. Créer deux variables, x et y. Ces variables, de types tableau, stockeront les coordonnées des noyaux. Il existe certaines fonctions s'appliquant au tableau de résultats permettant d'extraire le contenu d'une colonne, sur la base de son nom.
5. Pour chaque jeu de coordonnées (x, y) :
   - Sélectionner l'image `img`.

. S'assurer que le canal d'intérêt est actif : il existe une fonction permettant d'activer un canal sur une pile d'images.
. Appliquer la fonction `unwrapImage` aux coordonnées (x, y).
. L'image dépliée étant générée, la renommer sou la forme `Nucleus_XX`.

6. Utiliser la fonction `Image/Stacks/Images to Stack` poru assembler l'ensemble des images dépliées sous la forme d'une pile.
7. Tester la macro.
8. NB : pour qu'une fonction réalise l'opération souhaitée, il faut l'appeler...
9. NB2 : plutôt que de coder en dur le nom de l'image, on pourrait trouver une instruction permettant de garder une trace de l'image active lors de l'exécution de la macro.

Chapitre 4: Implémentation de l'analyse

Nallons finaliser la partie analyse de la macro. Nous disposons d'une image dépliée pour chaque noyau : l'idéal serait de moyenner les résultats obtenus sous la forme d'une unique image. L'utilisatrice souhaite également disposer d'un graph de la moyenne d'intensité en fonction de la distance au noyau : nous allons implémenter ces deux sorties.

Créer une image moyennée

C'est l'opération la plus simple à réaliser : nous disposons d'une pile d'images. Certaines opérations sont proposées directement par ImageJ/Fiji, notamment des fonctions de reconstruction 3D : on parle de projection. L'entrée de menu `Image/Stacks/Z Project...` permet pour une série de pixel (x, y), pour toutes les coupes de la pile, de ne conserver/de calculer une seule valeur à placer sur une image de destination. Parmi les options, on peut utiliser l'option `Average Intensity` pour obtenir l'image souhaitée. D'un point de vue opérationnel, il serait souhaitable de renommer cette image pour la retrouver plus facilement.

Créer un graph de l'intensité moyenne en fonction de la distance au noyau

La projection réalisée précédemment présente en abscisses l'angle d'observation, en ordonnées la distance aux noyaux. Le but de cette nouvelle étape est d'extraire pour une distance donnée, pour tous les angles, les intensités, de les moyenner, d'en extraire la déviation standard pour placer un point sur un graph. Certaines fonctions seront utiles : `getProfile`, que nous avons déjà vu, `Array.getStatistics(array, min, max, mean,` pour extraire des statistiques à partir d'une variable de type tableau.

Créer un graph *(source: doc. en ligne)* :

Plot.create("Title", "X-axis Label", "Y-axis Label", xValues, yValues) - Generates a plot using the specified title, axis labels and X and Y coordinate arrays. If only one array is specified it is assumed to contain the Y values and a 0..n-1 sequence is used as the X values. It is also permissible to specify no arrays and use Plot.setLimits() and Plot.add() to generate the plot. Use Plot.show() to display the plot in a window, or it will be displayed automatically when the macro exits.

Plot.add(type, xValues, yValues) - Adds a curve, set of points or error bars to a plot created using Plot.create(). If only one array is specified it is assumed to contain the Y values and a 0..n-1 sequence is used as the X values. The first argument (type) can be "line", "connected circle", "filled", "bar", "separated bar", "circle", "box", "triangle", "diamond", "cross", "x", "dot", "error bars" (in y direction) or "xerror bars".

Plot.show() - Displays the plot.

Étape 07 : à vous de jouer !

A faire : créer une nouvelle macro permettant de créer la projection, et de tracer les données.

- Créer la projection d'intensité moyenne à partir de la pile, la renommer `Moyenne`.
- Créer trois variables de type tableaux destinées à accueillir les données suivantes : `dist`, la distance au noyau en pixels ; `mean`, l'intensité moyenne pour une distance donnée ; `stdDev`, la déviation standard correspondante. Bien réfléchir à la taille des tableaux : on doit pouvoir soit la déduire de l'image active, soir la connaître (il s'agit d'un des paramètres utilisés précédemment).

1. Pour chaque distance au noyau :
   - Créer une région d'intérêt de type ligne.
   - Relever les intensités le long de cette ligne et les stocker dans une variable de type tableau.
   - Générer les statistiques sur ce tableau : relever la moyenne et la déviation standard.
   - Alimenter le tableau `dist` avec la distance courante au noyau.
- Créer et afficher le graph de l'intensité moyenne en fonction de la distance en pixels, en y ajoutant la déviation standard.
- Tester la macro
- Convertir ces instructions en une fonction que l'on nommera `analyzeAndPlot`. Bien évidemment, il faudra déclarer les paramètres utiles, le cas échéant, et l'appeler pour la tester.
- Ajouter cette fonction à la macro globale.

Chapitre 5: Amélioration de l'expérience utilisateur

Pour cette dernière séquence, nous allons améliorer l'expérience utilisateur. Nous avons implémenté une analyse qui requiert un certain nombre de paramètres pour être réalisée : une boite de dialogue pour les entrer plutôt que d'avoir à éditer directement le code. Nous verrons également qu'il est possible de stocker et rappeler les préférences utilisateur.

Créer une interface utilisateur

La création d'interface (**Graphical User Interface** ou GUI en anglais), quoique rudimentaire en langage macro, est relativement simple et puissante. Certaines fonctionnalités ont été récemment implémentées sous ImageJ pour augmenter les capacités du logiciel. Notamment, une nouvelle instruction permet d'afficher une boite de dialogue afin que l'utilisateur choisisse un chemin : c'est la version intégrée du `getDirectory` que nous avons utilisé et qu'il faudra remplacer. **Attention :** pour pouvoir utiliser les fonctions les plus récentes des boites de dialogue, il est impératif de procéder à la mise à jour du coeur ImageJ utilisé par Fiji. Pour cela, aller dans le menu `Help>Update ImageJ` puis redémarrer le logiciel.

Les paramètres à collecter sont les suivants :

- Taille minimale d'un noyau.
- Canal sur lequel faire l'analyse des noyaux.
- Canal sur lequel faire l'analyse.
- Distance maximale au noyau le long de laquelle faire l'analyse.

Une fois la boite de dialogue affichée, il faudra rapatrier l'ensemble des entrées utilisateur. Les éléments de la GUI ne sont pas stockés dans des variables indépendantes : on ne les adresse pas directement. Le mécanisme de collection se fait dans l'ordre où les éléments ont été ajoutés à l'affichage, au moyen de getters (fonctions permettant d'obtenir un résultat) spécifiques.

Créer une GUI *(source: doc. en ligne)* :

Dialog.create("Title")
Creates a modal dialog box with the specified title, or use Dialog.createNonBlocking("Title") to create a non-modal dialog. Call Dialog.addString(), Dialog.addNumber(), etc. to add components to the dialog. Call Dialog.show() to display the dialog and Dialog.getString(), Dialog.getNumber(), etc. to retrieve the values entered by the user. Refer to the DialogDemo macro for an example.
Dialog.addNumber(label, default) - Adds a numeric field to the dialog, using the specified label and default value.
Dialog.show() - Displays the dialog and waits until the user clicks "OK" or "Cancel". The macro terminates if the user clicks "Cancel".
Dialog.getNumber() - Returns the contents of the next numeric field.

Bien évidemment, afin de structurer au mieux le code, l'idéal serait de créer une **fonction GUI** en charge de l'affichage et du rapatriement des paramètres.

Obtenir les paramètres : portée des variables

La fonction GUI dispose d'un certain nombre de variables pour stocker les différents paramètres. Comme indiqué au "Chapitre 1 - Portée des variables", les variables à l'intérieur des fonctions sont locales et ne sont pas accessibles depuis l'extérieur.

L'utilisation de variables globales est généralement déconseillé. En effet, si l'on veut réutiliser un bloc fonctionnel, un simple copier/coller entre macros devrait suffire. Si une fonction utilise des variables globales, en plus du bloc fonctionnel, il faudra penser à créer les variables globales qui s'y rapportent, ce qui peut compliquer la tâche. Dans le cas de la fonction GUI, non seulement elle est spécifique de cette macro, mais en plus, elle comporte de nombreuses sorties qu'il ne sera pas pratique de stocker, par exemple, dans une unique sortie de type tableau. Nous allons donc employer des variables globales.

Étape 8 : à vous de jouer !

A faire : sur la macro intégrée précédente

- Basculer les variables contenant les paramètres en variables globales.
- Modifier la fonction `unwrapAllNuclei` de sorte à ce qu'elle prenne en compte à la fois le numéro du canal où détecter les noyaux et celui où réaliser l'analyse.
- Créer l'interface graphique, définir ses champs, l'afficher et rapatrier les entrées utilisateurs.
- Empaqueter l'interface graphique sous la forme d'une fonction nommée `GUI`.
- Tester la nouvelle macro