

M1 Imageurs, Poitiers 2024

Introduction aux macros ImageJ

Fabrice P. Cordelières

10 avril 2024

*Matériel de cours
disponible sur GitHub*




Corrigés en pièces jointes au pdf

TABLE DES MATIÈRES

| | | | |
|---|----------|---|----------|
| 1 Les bases du langage macro ImageJ | 1 | | |
| L'outil de base | 1 | Identifier les instructions clés à | |
| Le Macro Recorder | 1 | utiliser | 4 |
| Éditer une séquence et la sauvegarder : | | Tester la macro | 5 |
| .txt ou .ijm? | 1 | Gérer les exceptions utilisateur . . . | 5 |
| Exécuter une macro ou la stopper . | 1 | | |
| Les fonctions intégrées | 1 | 3 Fonctionnalisation de la macro | 6 |
| Structure des fonctions | 1 | Groupes fonctionnels | 6 |
| Trois logiques différentes | 2 | Utilité des fonctions | 6 |
| Éléments de syntaxe | 2 | Syntaxe des fonctions | 6 |
| Trouver ses marques avec ce | | Différents types de fonctions | 6 |
| nouveau langage | 2 | Fonctionnalisation de la macro . . | 6 |
| Commenter son code : quelles | | Créer la routine d'analyse | 7 |
| balises? | 2 | Coder la brique élémentaire 1 : | |
| Les variables : types, opérateurs | | pré-traiter l'image d'un canal | 7 |
| et portée | 2 | Coder la brique élémentaire 2 : | |
| Les boucles | 3 | mesurer la colocalisation à | |
| Structure de choix | 3 | partir des ROIs du canal 1 | |
| | | et des détections du canal 2 | 8 |
| 2 Premiers pas avec le langage macro | | | |
| ImageJ | 4 | 4 Amélioration de l'expérience utilisateur | 9 |
| Par où commencer? | 4 | Créer une interface utilisateur | 9 |
| Les étapes du protocole de traitement | | Obtenir les paramètres : portée des | |
| et d'analyse | 4 | variables | 9 |

CHAPITRE 1: LES BASES DU LANGAGE MACRO IMAGEJ

 E 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 débogage, 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 débogage/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 initialisées 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



U 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

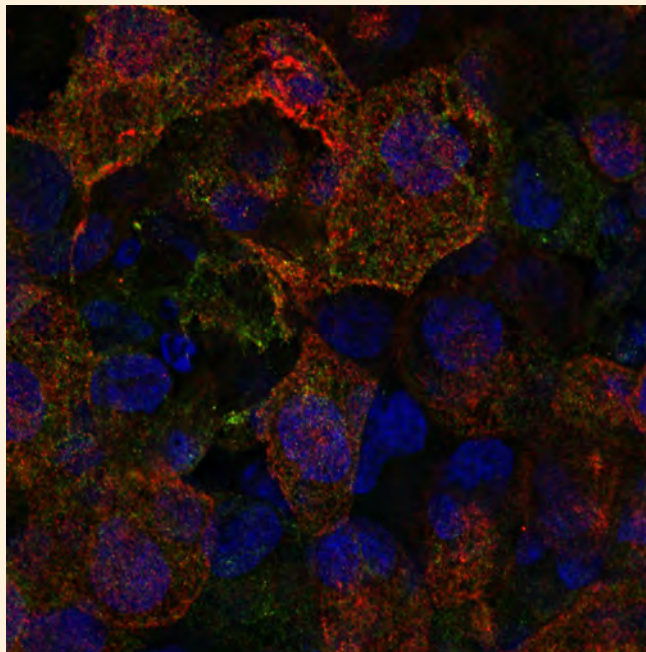
ImageJ dispose d'une fonction permettant d'enregistrement des opérations réalisées sous la forme d'un script : c'est le Macro Recorder, activé au moyen du menu

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

LA PROBLÉMATIQUE



Besoin : L'utilisatrice dispose d'images 3 canaux :DAPI/TRP/ZO1. Elle souhaite, sur un fichier d'images 3D :

- sélectionner une ROI (sur une coupe Z, on analyse en 2D)
- les signaux (localisation : find maxima ??)
- comparer les localisations entre deux canaux : colocalisation si dans le même cercle, proximité si dans le deuxième cercle, isolé si à l'extérieur
- gérer la construction d'un tableau résultat et la sauvegarde des résultats

termes, une analyse précédente aura-t-elle une incidence sur la nouvelle analyse, en laissant, par exemple, des 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 suivantes :
 1. Ouvrir une image exemple : File>Open...
 2. Demander à l'utilisateur de dessiner une région d'intérêt sur l'image. Attention, cette instruction n'est pas enregistrable : il faudra identifier l'instruction à l'aide de la page de référence.
 3. Ajouter la région au ROI Manager
Edit>Selection>Add to Manager. Peut-être qu'il sera nécessaire de veiller à ce que le ROI Manager soit vide au moment de l'ajout
 4. Dupliquer la portion d'image contenant la région et la nommer "Slice" : Image>Duplicate...u.
 5. Séparer les canaux en images individuelles :
Image>Color>Split Channels. Observer comment les canaux individuels sont nommés. Notez quels canaux seront d'intérêt pour l'analyse.

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.

GÉRER LES EXCEPTIONS UTILISATEUR

Et si l'utilisateur avait omis de dessiner une ROI au moment où celà lui est demandé ? La macro planterait au moment d'ajouter la ROI absente au ROI Manager. Comment gérer cette exception ? L'un des moyen serait de vérifier la présence d'une ROI avant de l'ajouter. On peut également envisager d'afficher le message d'invite tant qu'aucune ROI n'est présente sur l'image.

ÉTAPE 03 : À VOUS DE JOUER !

A faire :

- En vous référant au chapitre 1, trouver l'instruction qui permettra d'afficher le message tant qu'aucune ROI n'est dessinée.
- En vous aidant de en se référant à *cette page*, trouver l'instruction renvoyant la nature de la ROI (ou -1 si aucune n'est dessinée).
- En amont de l'invite, on peut également faire en sorte d'effacer toute ROI déjà présente sur l'image. L'instruction est enregistrable au travers du menu Edit>Selection>Select None.

CHAPITRE 3: FONCTIONNALISATION DE LA MACRO



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.

GROUPE 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 commencer 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** réalise la préparation des images pour l'analyse. Il peut être empaqueté au sein d'un groupe fonctionnel simple, ne prenant aucun argument et ne renvoyant aucun résultat.

ÉTAPE 04 : À VOUS DE JOUER!

A faire : Adapter le code de manière à fonctionnaliser la préparation de l'image :

- Créer la fonction `prepareImages()`.
- Testez la macro
- NB : pour qu'une fonction réalise l'opération souhaitée, il faut l'appeler...

CRÉER LA ROUTINE D'ANALYSE

CODER LA BRIQUE ÉLÉMENTAIRE 1 : PRÉ-TRAITER L'IMAGE D'UN CANAL

L'utilisatrice demande à repérer des structures ponctuelles sur ses images. Les structures étant d'une taille proche de la résolution, elles apparaissent sous forme de points assimilables à des maxima locaux : c'est cette stratégie que nous allons mettre en oeuvre. ImageJ embarque une telle fonctionnalité dans le menu

Process>Find Maxima. Cette fonction est enregistrable dans les macros. Elle demande à l'utilisateur d'entrer deux paramètres : la prominence (de combien un pixel doit dépasser de l'intensité de ses voisins pour être considéré comme maximum local) et un type de sortie (une sélection, une image etc).

En terme de prominence, il faudra adapter la valeur en fonction de l'image. Concernant la sortie, il nous faudra à la fois une ROI contenant tous les points détectés (on pourra réutiliser les coordonnées de chaque point pour placer des zones de recherche autour), ainsi que l'image des points détectés (elle nous servira pour le comptage de points colocalisant/situés à proximité d'un point de référence).

En testant rapidement la fonction Find Maxima, on se rend compte que l'image de l'utilisatrice nécessite un peu de pré-traitement pour que la détection soit optimisée. Plusieurs outils sont à disposition, comme les filtres de rang

(Process>Filters>Median, mais également l'algorithme rolling-ball qui permet d'éliminer le bruit de fond évalué localement

(Process>Subtract background). Anoter, ces deux outils nécessitent d'entrer chacun un paramètre, à affiner en fonction de l'image : le rayon de filtrage/d'analyse du fond.

ÉTAPE 05 : À VOUS DE JOUER !

A faire : créer une nouvelle macro permettant de réaliser le pré-traitement et les détections des structures ponctuelles d'intérêt. Au préalable, on aura pris soin de tester les différentes étapes possibles et de relever les paramètres optimaux.

1. Créer la fonction `preprocessImage` : elle prend les arguments suivants :
 - . L'image sur laquelle réaliser l'analyse, `image`.
 - . Le rayon du filtre median, `median`.
 - . Le rayon de recherche du fond local, `bkgd`.
 - . La prominence utilisée pour la détection des maxima locaux, `prominence`.
2. Sélectionner l'image `image`.
3. Dupliquer l'image : on conserve l'image originale et on pré-traite une copie dont le nom est constitué à partir de `image` et du suffixe `"-Processed"`.
4. Réaliser le filtre médian de rayon `median`.
5. Réaliser la soustraction de fond de rayon `bkgd`.
6. Rappeler la ROI dessinée par l'utilisateur : même si le référentiel image est différent, elle sera rappelée relativement au coin supérieur gauche de l'image.
7. Réaliser la détection des maxima locaux avec le paramètre `prominence` et une sortie de type `"Single points"` : une nouvelle image est créée.
8. Donner un nom standardisé à l'image de sortie, du type `image` suivi de `"-points"`.
9. Sélectionner de nouveau l'image pré-traitée : ImageJ travaille sur l'image active, et la dernière image activée est celle contenant les points.
10. Réaliser la détection des maxima locaux avec le paramètre `prominence` et une sortie de type `"Points selection"`.
11. Ajouter cette région au ROI Manager pour utilisation ultérieure, lors de la phase d'analyse.
12. Ne pas oublier d'appeler la fonction !
13. NB : La fonction devra être appliquée aux deux canaux à analyser.

CODER LA BRIQUE ÉLÉMENTAIRE 2 : MESURER LA COLOCALISATION À PARTIR DES ROIS DU CANAL 1 ET DES DÉTECTIONS DU CANAL 2

Pour mesurer colocalisation et proximité entre éléments des deux canaux, on utilisera tour à tour les ROIs déterminées sur un canal et l'image des points obtenue à partir de l'autre canal.

A partir de la ROI, on pourra extraire les coordonnées des points individuels : il existe une fonction pour cela, non enregistrable. Chaque point pourra servir de centre pour la création d'une région circulaire, de rayon correspondant au rayon de recherche de colocalisation ou de proximité.

Placée sur l'image des points, cette ROI circulaire définira le territoire où trouver des détection sur le second canal. On pourra réaliser une mesure d'intensité dans cette ROI. Chaque point a pour intensité 255 : l'intensité sommée de la région, divisée par 255, permet de remonter au nombre de points par zone, sans avoir à réaliser de détection de maxima locaux.

En s'aidant de la page des *fonctions macro ImageJ*, on pourra bâtir un tableau de résultats, nommée d'après l'image, contenant pour chaque point ses coordonnées, le nombre de points de l'autre canal présents dans la zone de colocalisation, et dans la zone dite de proximité.

Petit raffinement : on pourrait garder une trace des ROIs circulaires en les ajoutant à l'Overlay des images de détections. L'overlay est comme un unique calque qui peut stocker des ROIs directement dans l'image.

ÉTAPE 06 : À VOUS DE JOUER !

A faire : créer une nouvelle macro permettant ajoutant aux fonctionnalités précédentes celle d'analyse de colocalisation/proximité.

1. Créer la fonction `colocProx` : elle prend les arguments suivants :
 - . Le numéro de la ROI qui, dans le ROI Manager contient les détections du canal, `roiNb`.
 - . L'image sur laquelle contenant les détections (points) de l'autre canal, `image`.
 - . Le rayon de colocalisation, `coloc`.
 - . Le rayon de proximité, `prox`.
2. Créer le tableau de résultat nommée `Roi_roiNb-Image_image`.
3. Sélectionner l'image `image`.
4. Si nécessaire, bien veiller à la cohérence des unités : certaines fonctions fonctionnent sur la base d'un nombre de pixels, d'autres sur une taille exprimée en unité de calibration de l'image (microns). On pourrait vouloir stocker la calibration actuelle de l'image et "traduire" les rayons de colocalisation/proximité de pixels à unités ou inversement.
5. Dans le ROI Manager, sélectionner la ROI `roiNb`.
6. Extraire toutes les coordonnées x,y de chaque point de la ROI et les stocker dans des variables de type tableau, `xpoints` et `ypoints` (il y a sans doute une fonction pour cela!).
7. Pour chaque point de coordonnées `xpoints`, `ypoints` :
 - . Créer une sélection circulaire de rayon `coloc`, centrée sur `xpoints`, `ypoints`.
 - . Optionnel : Donner une coloration verte aux contours de la ROI.
 - . Optionnel : Ajouter la ROI circulaire à l'overlay de l'image
 - . Rappatrier les statistiques brutes de la région.
 - . Créer la variable `nbColoc` qui stocke le nombre de détections dans la zone de colocalisation.
 - . Créer une sélection circulaire de rayon `prox`, centrée sur `xpoints`, `ypoints`.
 - . Optionnel : Donner une coloration rouge aux contours de la ROI.
 - . Optionnel : Ajouter la ROI circulaire à l'overlay de l'image
 - . Rappatrier les statistiques brutes de la région.
 - . Créer la variable `nbProx` qui stocke le nombre de détections dans la zone de proximité.
 - . Dans le tableau de résultat, logger sur une ligne par coordonnées, dans des colonnes individuelles, les informations suivantes :
 - Un label, du type `Spot_XX`.
 - Les coordonnées XY du point source (2 colonnes).
 - Le nombre de détections dans la zone de colocalisation.
 - . Le nombre de détections dans la zone de proximité.
8. Ne pas oublier d'appeler la fonction!
9. NB : La fonction devra être appliquée aux deux canaux à analyser en combinant les ROIs et images de détections.

CHAPITRE 4: 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 boîte de dialogue pour les entrer plutôt que d'avoir à éditer directement le code.

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 boîte 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 boîtes de dialogue, il est impératif de procéder à la mise à jour du cœur ImageJ utilisé par Fiji. Pour cela, aller dans le menu `Help>Update ImageJ` puis redémarrer le logiciel.

Une fois la boîte 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 7 : À VOUS DE JOUER !

A faire : sur la macro intégrée précédente

- Basculer les variables contenant les paramètres en variables globales.
- 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
- Optionnel : maintenant que tout marche, pourquoi ne pas créer une fonction `cleanUp` qui préparerait l'analyse en fermant toutes les images (sauf l'image active), qui viderait le ROI Manager etc ?