



# The NEMO Dots Assembly: Single-Particle Tracking and Analysis

*Jean-Yves Tinevez and Sébastien Herbert*

- 4.1 Introduction – 68**
- 4.2 Datasets – 69**
- 4.3 Tools and Prerequisites – 70**
- 4.4 Workflow – 70**
- 4.5 Single-Particle Tracking with TrackMate – 71**
  - 4.5.1 Step 1: Loading Image Data and Launching TrackMate – 71
  - 4.5.2 Step 2: Detection – 72
  - 4.5.3 Step 3: Filtering – 73
  - 4.5.4 Step 4: Particle-Linking – 76
  - 4.5.5 Step 5: Filtering Tracks – 79
  - 4.5.6 Step 6: Export Results – 81
- 4.6 Motility Analysis with Mean-Square Displacement – 82**
  - 4.6.1 Step 1: Importing Tracks into MATLAB – 82
  - 4.6.2 Step 2: Create and Add Data to the MSD Analyzer – 85
  - 4.6.3 Interlude: A Short Word About Mean-Square Displacement Analysis – 87
  - 4.6.4 Step 3: Compute the Mean-Square Displacement – 88
  - 4.6.5 Step 4: Log-Log Fit of the Mean-Square Displacement – 90
  - 4.6.6 Step 5: Analysis of the Log-Log Fit – 91
- 4.7 Results and Conclusion – 94**
- Bibliography – 95**

## What You Learn from This Chapter

The aim of this chapter is to learn the principles and pitfalls of single-particle tracking (SPT). Tracking in general is very important for dynamic studies, as it is about propagating object identities over time, permitting the calculation of dynamic quantities such as object velocities. Tracking is often the first step in analyzing dynamics.

The output of tracking is simply tracks, and later steps involve computing relevant quantities from these tracks. In the case of the applications we use in this module, we wanted to learn something about the particles we track, which are unknown organelles (at the time of the publication) appearing transiently in cells upon stimulation by an interleukin. Namely we want to determine whether they are bound to a structure, transported or freely diffusing. To do so, the analysis is completed by performing a Mean Squared-Displacement (MSD) analysis.

### 4.1 Introduction

---

The data and analysis we will perform in this module is taken from the following paper: *TNF and IL-1 exhibit distinct ubiquitin requirements for inducing NEMO-IKK supramolecular structures*, Tarantino et al. (2014). In particular, we will reproduce the analysis of the paragraph “NEMO-containing punctae are slow-moving anchored structures located close to the cell surface”.

Nuclear factor KB (NF-KB) essential modulator (NEMO) is a regulatory component of the IKK kinase (IKK) complex and controls NF-KB activation through its interaction with ubiquitin chains. The work of Emmanuel Laplantine focuses on the mechanics of NF-KB regulation by ubiquitination. Recently, his lab showed that NEMO, a component of the IKK complex, is crucial for NF-KB activation and the linear ubiquitination by K64. Patients with a deficiency in the linear ubiquitination machinery enabled to correlate their symptoms with a defect in NF-KB activation by cytokines. This project aims at investigating the details of the NF-KB activation initiated by stimulation by IL-1 and TNF.

We engineered cells that were expressing constitutively NEMO-eGFP from a NEMO-deficient human fibroblast cell line. They allowed us to follow NEMO dynamics using high-resolution microscopy. The most marking result of the project is that stimulation with interleukin-1 (IL-1) and TNF induces a rapid and transient recruitment of NEMO into punctate structures. These structures appear briskly, probably assembling from a pool of NEMO soluble in the cytoplasm, and roughly constant in quantity in the cell over time. They disassemble in, on average, 15–45 min depending on the stimulus (TNF or IL-1).

Our part of the project revolved around the characterization of these previously unknown structures via imaging and image analysis, completing data obtained by biochemistry. These structures and their dynamics proved to be extremely sensitive to light, and their study required careful imaging with a dedicated protocol. The part we will cover in this module focuses on a single question: Are these punctate structures bound to the membrane, freely diffusing or transported?

This simple question was important to backup the biochemistry data. Since the punctate structures were not described before and their function unknown, their motion type could give us clues about their function. For instance, if they are transported, they may be internalized in some vesicles and transported from membrane to nucleus to convey the cell activation signal. If they are freely diffusing, they may be supramolecular structures polymerizing upon some signal.

Our first imaging protocol falsely led us to think that they were transported towards the nucleus after assembly. However, these movements proved to be artifacts and caused by a phototoxic effect.<sup>1</sup> A second imaging protocol involved the use of TIRF microscopy with a low illumination power, which diminished the phototoxic effects. We filmed the dots for long times and process the acquired structures to analyze their motility.

We tracked the dots in Fiji using TrackMate (Tinevez et al. 2017) and, because the dots are well separated, the tracking proved relatively easy. We then analyzed the tracks using MSD analysis, to conclude on their motility with certainty. The MSD analysis is also the subject of this module, and we will then go from Fiji to MATLAB to perform it.

This particular analysis proved that the NEMO dots are anchored, both when stimulated by IL-1 and TNF. We concluded further that they are not anchored to actin filaments or microtubules (MTs), as repeating the analysis with drugs that depolymerize the cytoskeleton did not show any change. Additional analysis showed that they were most likely anchored to the cell membrane, and that NEMO molecules were under a rapid turnover in these punctate structures. So they probably play the role of phosphorylation factories, assembled and anchored at the cell membrane, that would process quickly a large amount of the otherwise soluble NEMO proteins.

## 4.2 Datasets

The data for this module consists in a subset of the data only from the paper. It only features 5 movies over 2 conditions:

- Ctrl: A couple of dots can be seen wandering in the cells, even if there is not stimulation. They are permanent instead of transient, and probably non-specific. They give a control of how spurious particles would perturb our measurements.
- IL1: Following IL-1 stimulation. In the study, this was the “easy” case, for the dots were bright and large compared to *e.g.* the dots we see after TNF stimulation.

You can find it on Zenodo:

► <https://doi.org/10.5281/zenodo.1341987>. The dataset (download size about 800 MB) is organized as follow:

```
Tracking-NEMO-movies_subset
NEMO-Ctrl
  Cell_01.tif           35.2 MB
  Cell_01.xml           5.9 MB
  Cell_01_Tracks.xml   16.6 kB
  Cell_02.tif          175.5 MB
  Cell_02.xml           5.6 MB
  Cell_02_Tracks.xml   170.4 kB
NEMO-IL1
  Cell_02.tif           91.9 MB
  Cell_02.xml           6.4 MB
  Cell_03.tif          444.8 MB
  Cell_03.xml           53.0 MB
```

1 We documented this phototoxic effect later in Tinevez et al. (2017).

Cell_03_Tracks.xml	1.2 MB
Cell_04.tif	251.6 MB
Cell_04.xml	50.8 MB
Cell_04_Tracks.xml	4.6 MB

The movies themselves are not very pretty. Bright dots can be seen over a cell background caused by the soluble pool of NEMO. They bleach over time. The temporal resolution is not very high (0.5 s) and the SNR is not high either since we had to compromise on laser power to avoid cell death.

Files are `.tif` movies, made for ImageJ, with the right spatial and temporal calibration. They are already split cell by cell, and have a ROI that encloses the cell. There also are `.xml` files from TrackMate and `_Tracks.xml` files generated from TrackMate, ready to be imported in MATLAB. But we will do the tracking ourselves.

### 4.3 Tools and Prerequisites

- Fiji
  - Download URL: ► <https://imagej.net/Fiji/Downloads>
  - It does not require any extra, as TrackMate is included in the core of Fiji.
- MATLAB
  - We rely on MATLAB for the MSD analysis part, with the Curve Fitting toolbox.
  - You need to know at least a little bit about MATLAB features, like logical indexing and structures. We will not be introducing them here.
  - Because we will install specialized functions and classes in MATLAB, you also need to know at least a little bit about the MATLAB path. ► [https://mathworks.com/help/matlab/matlab\\_env/what-is-the-matlab-search-path.html](https://mathworks.com/help/matlab/matlab_env/what-is-the-matlab-search-path.html)
  - We built a special class to perform the analysis that you can download here:
    - <https://github.com/tinevez/msdanalyzer/zipball/master>

`@msdanalyzer` is a MATLAB *class*, so you have to put the `@msdanalyzer` *folder* in the MATLAB path, but not its content.<sup>2</sup> For instance on my MATLAB installation, I have a folder called `/Users/tinevez/Development/Matlab/msdanalyzer` that is on the path and that contains the `@msdanalyzer` folder. But the `@msdanalyzer` folder is not in the path.

### 4.4 Workflow

We will deal separately with single-particle tracking in Fiji using TrackMate, and track motility analysis in MATLAB using `@msdanalyzer`. The two following sections are largely independent and present different concepts. To perform the MSD analysis, please use the dataset linked above that include them.

<sup>2</sup> This is explained on The Mathworks website: ► [https://mathworks.com/help/matlab/matlab\\_oop/organizing-classes-in-folders.html](https://mathworks.com/help/matlab/matlab_oop/organizing-classes-in-folders.html).

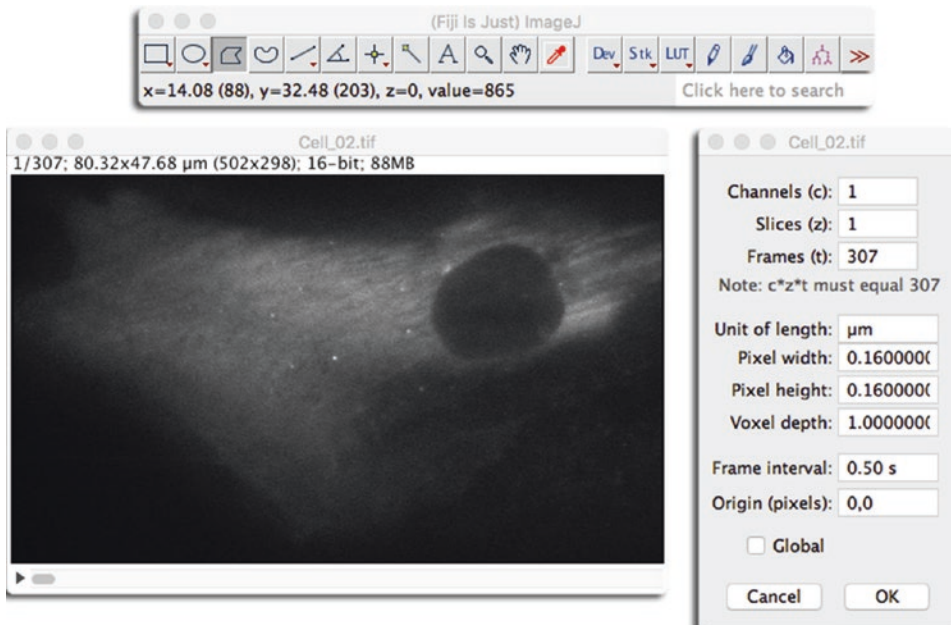
## 4.5 Single-Particle Tracking with TrackMate

TrackMate is a Fiji plugin dedicated to tracking. It can do cell-lineaging (and was initially developed for this very purpose, see Tinevez et al. (2012)) but also has automated analysis tools to perform single-particle tracking of sub-cellular structures. It ships a user-friendly graphical user interface that allows to inspect tracking results and curate them. The following part describes how to use TrackMate to generate the tracks over one of our movies. An extended documentation for this plugin can be found here: ► <https://imagej.net/TrackMate>, along with supplementary material of the associated publication.

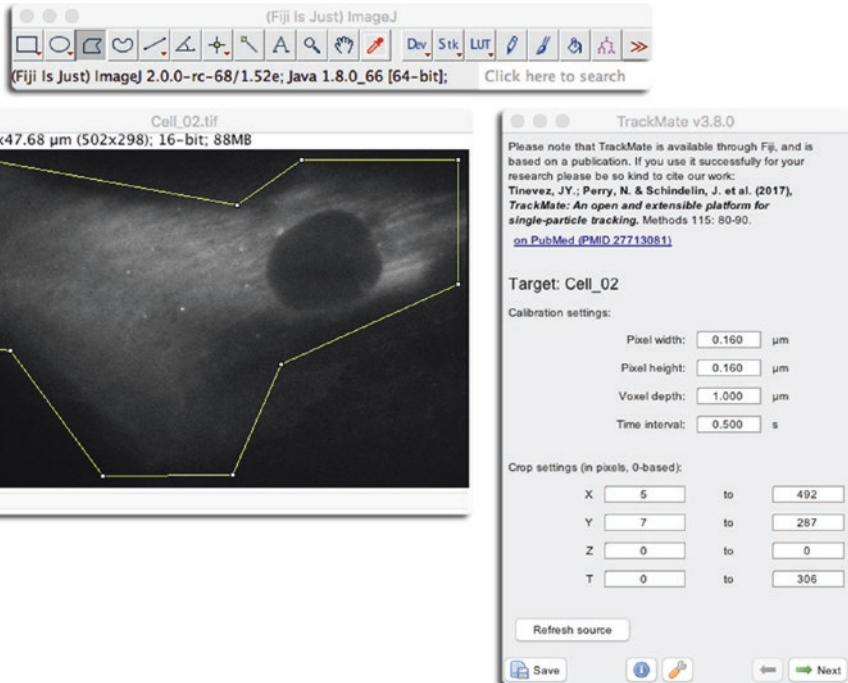
### 4.5.1 Step 1: Loading Image Data and Launching TrackMate

For the example below, we will use the `Cell_02.tif` in the `NEMO-IL1` folder. You can display a `.tif` file by performing a simple drag and drop on the Fiji toolbar. This movie does not have many dots, which will simplify getting familiar with the workflow.

It is very important that you check the dimensionality of the image at this point of the analysis and correct it if required. To do so, check the image properties in Fiji (`Image >> Properties...` menu item or `⬆+P`, ► Fig. 4.1). In our case we have a 2D over time acquisition, so make sure the metadata reports 1 z-slice and 307 frames. Also, TrackMate reports any quantities (space and time) in physical units, so the pixel size



► Fig. 4.1 Image properties



■ Fig. 4.2 The first panel in TrackMate UI




and frame interval must be correct since you will not be able to change them further in the analysis. For these movies, the pixel size is  $0.160 \mu\text{m}$  per pixel and the frame interval is  $0.5 \text{ s}$ .

When this is done, close the properties window, make sure the active image is the one with our NEMO-labelled cell, and launch TrackMate. The plugin can be found in the `Plugins >> Tracking >> TrackMate` menu. The GUI will show up and the first panel will display a recapitulation of the image metadata. At this step, you can define a ROI that will be used for the analysis. In our case it does not matter, since the images are cropped around a cell. If you want to use a ROI in TrackMate, draw a ROI over the active image, and press the `Refresh source` button on the first panel. You should see the bounds changing on the panel (■ Fig. 4.2).

## 4.5.2 Step 2: Detection

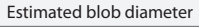


In our case, the objects we want to track are NEMO dots; since they are smaller than the resolution limit of the microscope, we cannot resolve their shape, hence *segmenting* them would not bring any information that would allow us to discriminate them. They all look the same in the eye of our microscope. We need a simple *detection* algorithms that will yield their position nothing more, which is exactly what TrackMate ships.

The TrackMate user interface is inspired by the Bitplane Imaris wizard. You will find at the bottom right of the panel a `Next` button that will bring you to the next panel when you are done with the current one. Typically, you deal with one group of parameters or


choice per panel and you can navigate backward if you want to try another one. Click on the  button and you will now see a panel where you can choose the detector we will use. Pick the , which is the default, then click again on the  button. You are now presented with a panel that lets you configure the LoG detector.

The LoG detector is performing remarkably well for its simplicity. It excels at finding bright, blob-like, roundish objects, that we will call spots or detections and only requires two parameters: the approximate diameter (in physical units) of the objects we want to detect, and a threshold value on a quality metrics, below which detections will be rejected. The LoG detector works by filtering the image with the Laplacian of Gaussian filter (also dubbed Mexican hat filter) tuned to the specified diameter. In the filtered image, the spots will appear as bright and sharp peaks, and they are detected by looking for local maxima. The quality of a detection is the value at the local maxima location in the filtered image. Due to this image filtering step, spots smaller or larger than the estimated diameter will have a lower quality value than spots of the same intensity but of the right size. Consequently, the quality of detection is highest when the spot is bright and of the right size.

### Exercise 4.1

Play with the  and  parameters to find settings that would detect all NEMO dots and a limited number of spurious detections. The  button will show you what the chosen parameters do on the current frame only. The other parameters can be ignored.




### Solution


Using a diameter of 0.5  $\mu\text{m}$  and a threshold of 1000 seems OK ( Fig. 4.3). We still have many spurious spots but we can filter them out later.

Click on the `Next` button to run the analysis on the whole movie. It should not take too long (a few seconds) and you should have about 9000 spots in total.

## 4.5.3 Step 3: Filtering

---

It is very important to filter out as many spurious spots as we can because detection might very well yield a large amount of them. By setting the quality threshold to a non-zero value we already filtered them a first time. When clicking  after the detection is completed, you are presented with the initial filtering panel ( Fig. 4.4). It shows the quality histogram and allows for discarding spots with a quality lower than what we set here. In simple cases, we expect a bi-modal histogram, with two peaks well separated between spurious spots with low quality and spots with a high quality. This is not the case here, so we keep the value unchanged and move to the next panel by clicking .

TrackMate lets you pick the view to display the detection and tracking results. As of today, there is only one working consistently, the HyperStack Displayer. It simply displays the results on the ImageJ hyperstack, so select this one. The spots are now displayed on the image as magenta circles ( Fig. 4.5). A quick inspection reveals that we still have spurious spots. You can see them appearing and disappearing as you move in time, while true spots tend to persist over several frames.

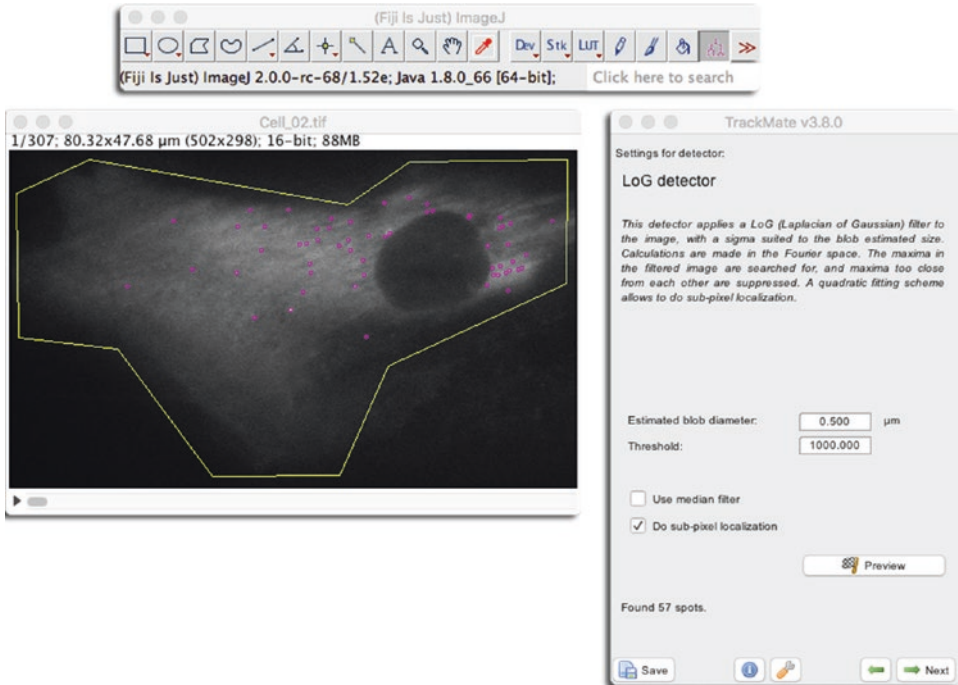
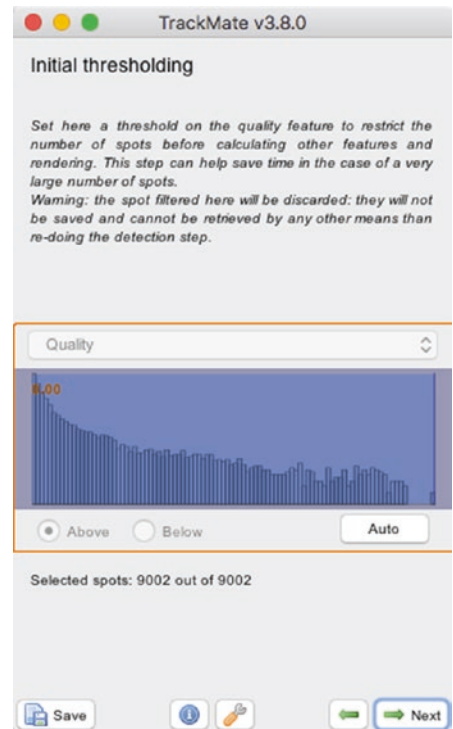
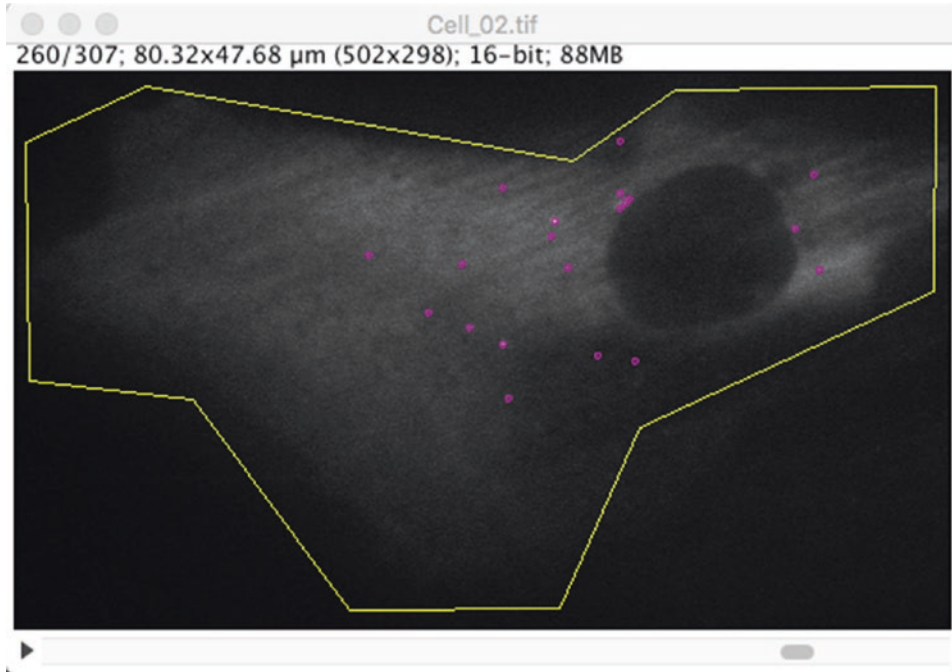


Fig. 4.3 LoG detector parameters

Fig. 4.4 Initial filtering panel







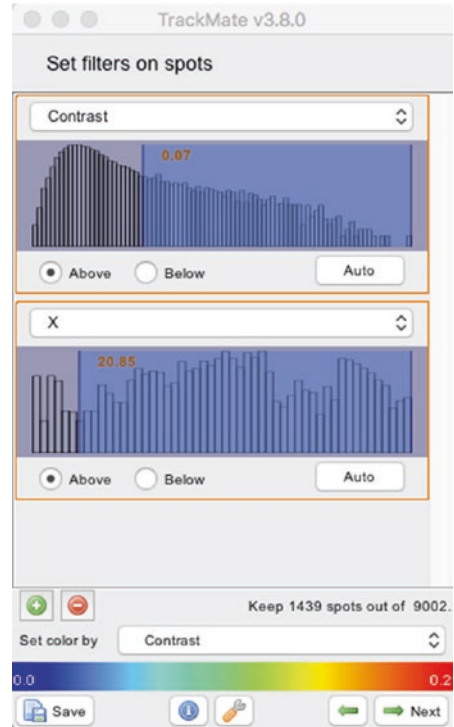
■ Fig. 4.5 The HyperStack Displayer with detection results


Move on to the next panel by clicking . We are now presented with another filtering panel. As we already had one before (the initial filtering panel), it is worth mentioning the differences.

1. Spots have numerical features attached to them. A feature is a numerical scalar value that reports a quantitative information on the object it represents. For instance, the mean intensity around a spot location is a numerical feature. Features are calculated *after* the initial thresholding step. The filters you set in this panel are based on features that are not available before this step and the initial filtering can only be done on the quality value.
2. These filters are *reversible*. The spots are not deleted from the data, but hidden. So when you remove a filter, the spots it discarded reappear. This is useful if you realize later that the filters were inadequate and too stringent and preventing proper linking. To adjust the filters later, you can navigate back to this panel by pressing the left green arrow on the GUI. In contrary, when using the initial thresholding, the spots are deleted, which is useful to save memory but is irreversible. If you want to retrieve the spots you discarded in the initial thresholding step, you must re-run the detection step.

Filters are added by clicking on the green  button. A small panel appears that lets you choose the feature you want to set the filter on, with what value and whether we should retain spots with feature values above or below the threshold. The panel also displays the histogram of the feature values collected for all spots, and has an `Auto` button that

**Fig. 4.6** Filtering panel with two filters on contrast and X position, additionally setting the spot color by contrast value



automatically determines a threshold value using Otsu method (Fig. 4.6). The red  button removes the last filter. Also note that you can use the feature values to color the spots, using the drop-down list at the bottom of the filtering panel. Filters are combined by taking the intersection of the spots they yield. On Fig. 4.6 for instance, we retain the spots that have contrast values above 0.07 **and** whose X position are above 20.85  $\mu\text{m}$ .

#### Exercise 4.2

Find a combination of filters that remove all spurious spots and keep the true ones.

#### Solution

Such a combination is hard if not impossible to find in our case. Thankfully, we do not mind a few spurious spots as we will be analyzing tracks. We will later filter out tracks made of spurious spots.

Remove all filters, set the `color-by` option to `Uniform color` and click the `Next` button.

### 4.5.4 Step 4: Particle-Linking

Now that we have spots, we want to link them over time and build tracks. The tracks will be what we will analyze in the section dedicated to motion analysis, and we will do it in MATLAB. But for now we need to generate these tracks. In TrackMate, particle-linking happens similarly to the detection step. You are now presented with a

panel that lets you select the particle-linking algorithm (or “tracker”) to be used for the next step.

TrackMate ships several trackers, but the most useful ones fall in two main categories:

- The LAP-based trackers. LAP stands for Linear Assignment Problems. There are two trackers named `Simple LAP tracker` and `LAP tracker` that implement a stripped version of an algorithm published by Jaqaman et al. (2008). They are configured to deal well with objects that diffuse or move randomly.
- The Kalman-filter based trackers. We have only one, called `Linear motion LAP tracker`. It is based on what is called a Kalman-filter<sup>3</sup> introduced in the 1960s by R.E. Kalman (Kalman (1960)). Our implementation is well suited to particles that have a nearly-constant velocity vector. That is: particles that move by roughly the same amount between each frame and do not change direction too fast. Of course it can accommodate some changes of velocity provided they are modest.

Choosing the right tracker is critical. In Chenouard et al. (2014), the performance of 14 different single-particle tracking methods were assessed. One of the main conclusions of this work is that there is not a universally good tracker for all bio-imaging problems. A tracking algorithm has to be chosen depending on the motion model of the objects to be tracked. For instance, the LAP trackers of TrackMate are well suited for objects that are freely diffusing or bound. The Linear motion tracker is well suited for objects that are transported.

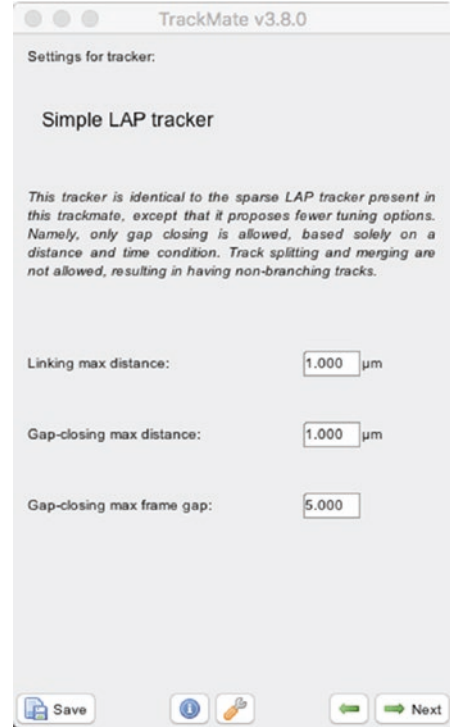
This causes a chicken-and-egg problem in our case, since we actually want to determine what is the motion model of the NEMO dots by analyzing tracks. Practically, we carefully looked at the movies and assessed whether it was plausible for the dots to be transported. Their motion seemed erratic, and so we started with the LAP tracker. As we will see later, we found that the dots have a motion type for which the LAP trackers are well suited, so our choice appears valid *a posteriori*. This is close to having a circular reasoning fallacy. However, we must temper this criticism. The choice of the right tracker is important to yield accurate tracks that faithfully follow the true particles over time. The analysis of tracks is a subsequent step. So first, an inadequate choice of a tracker can be detected by checking the tracks manually, following a dozen of them and looking for jumps to another particles or early breaks. And second, we can be in a situation where the particle density and the detection quality is such that the choice of the tracker will not matter. This is the subject of an exercise below.

In TrackMate, the trackers suited for non-transported motion are the LAP trackers. They are based on minimizing the total cost to link a set of spots in one frame to the spot in the next frame, or the cost to link track segments together. We have the `LAP tracker` and its `Simple LAP tracker` version. They actually wrap the same algorithm, but the latter offers fewer configuration options. The `LAP tracker` can be configured to generate tracks that are splitting (as for cell division) or merging. The cost to link one spot to another one can be altered by differences in spots features values, such as intensity, radius, ... The `Simple LAP tracker` only offers to bridge gaps in tracks caused by missed detections, and the linking costs are simply based on distance. It results in tracks being *linear*, that is without merges or splits, and at most one spot per frame. We observe that the NEMO dots do not merge or split, so this is the right tracker to pick.

---

3 Kalman filter on Wikipedia: ► [https://en.wikipedia.org/wiki/Kalman\\_filter](https://en.wikipedia.org/wiki/Kalman_filter).

Fig. 4.7 Configuring the Simple LAP tracker



### Exercise 4.3

The `Simple LAP tracker` has 3 parameters to configure it, that set

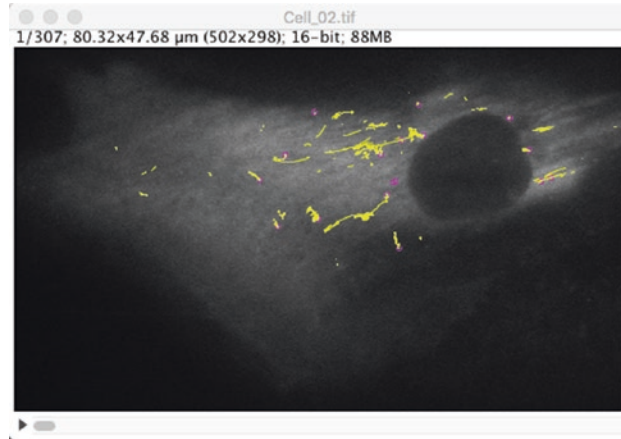
- the maximal distance to link from one frame to the next;
- the maximal frame gap to bridge over missing detections;
- the maximal distance to bridge over missing detections.

Try and find a suitable parameter set that yields acceptable results, based on you checking the tracks.

### Solution

As explained in Jaqaman et al. (2008), this tracker performs spatially global optimization, and therefore is rather robust against a lot of variation in parameter values. You should find acceptable results for a wide range of parameters, provided they are not aberrant. Try some of them and click the `Next` button to get the tracking results displayed on the image. Click on the `Previous` button to change parameters and start again. Check Fig. 4.7 for values that work.

■ Fig. 4.8 Raw tracking results



### 4.5.5 Step 5: Filtering Tracks

The image should be updated with tracks, as shown in ■ Fig. 4.8. We note that there are about 3 tracks that seem to display a large excursion in the cell. Also, there are many tracks that are very short and are probably made of spurious spots. We want to filter them out. Move on to the next panel.

#### ❓ Exercise 4.4

The track filtering panel works like for the spot filtering panel. Find a combination of filters that can remove tracks originating from spurious spots.

#### ✓ Solution

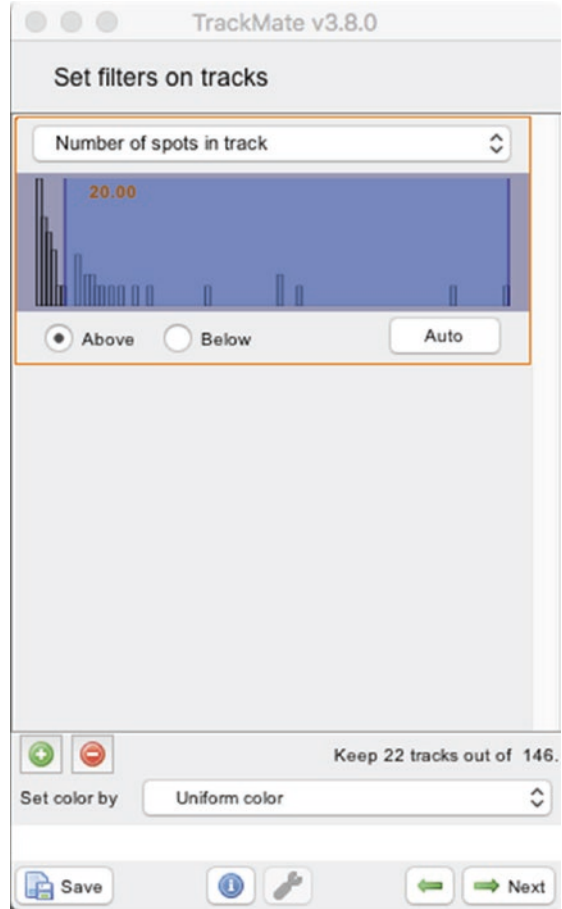
Spurious spots arise from noise in the image. As long as there are only a few of them, it is therefore very unlikely that spurious spots appear many times consecutively in the vicinity of the same location. If they are ever linked in a track, it will be short compared to tracks originating from true spots, that can be followed over many frames. A good strategy is therefore to filter tracks based on the number of spots they contain. It has a side benefit: the MSD analysis we will perform later in this chapter requires the tracks to be long for accuracy. Since the histogram for the number of spots in tracks have a large peak at low values that precede a gap at  $N=20$ , we can use this value as a threshold (■ Fig. 4.9).

So we are now left with a small number of long tracks (■ Fig. 4.10).

#### ❓ Exercise 4.5

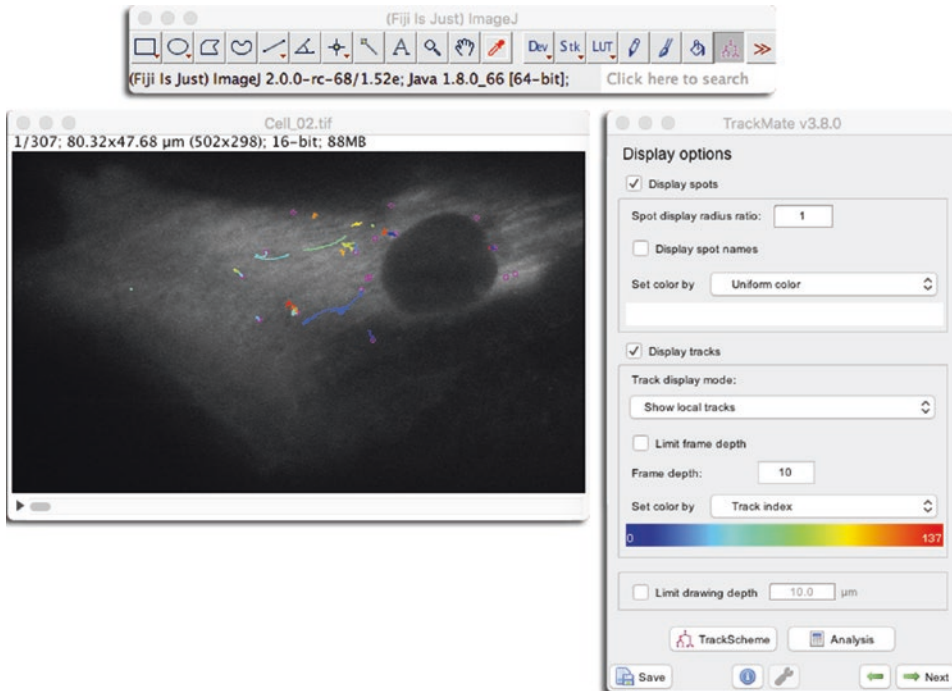
Go back 3 steps and try to perform tracking with the `Linear motion LAP tracker`. As we said, it is not the optimal tracker for the motion type we suspect the NEMO dots have. Check whether the tracking results differ from those of the `LAP tracker`.

Fig. 4.9 Filtering tracks based on the number of spots they contain



### ✓ Solution

Even though we do not have many tracks in our case, visual inspection is not enough. A good approach for a first comparison of tracking results is to compare tracks based on the feature values that are calculated for them. To do this, once the TrackMate GUI shows the `Display options` panel (Fig. 4.10), click on the `Analysis` button. Three tables should appear, and we just want to retain the `Track statistics` one. Duplicate it (`File` > `Duplicate`) when the table is active) and give it a name like `Track statistics-LAP`. Then go back 3 steps, select the `Linear motion LAP tracker` and perform tracking as before. Regenerate the `Track statistics` table and compare with the previous one. You will find that we have identical tracks. The track labels will be different because they are regenerated every-time we perform tracking, but you will see they are made of the same spots.



■ Fig. 4.10 The final tracking results

This worked in our case only because we have an easy case for tracking: the spots that remain after filtering are few and well spaced. The density is so low that the motion type of the tracker does almost not matter. As shown in Chenouard et al. (2014), at high density the difference of performance among trackers is exacerbated.

#### 4.5.6 Step 6: Export Results

We want now to export the track results in a format that can easily be re-imported in MATLAB. The XML file that is generated when you press the **Save** button in the GUI contains all the information to restore a tracking session: settings, parameter values, path to images, etc. It is probably not well suited to the simple track import we want to perform.

Move to the last panel of the TrackMate GUI called **Select an action**. It offers a selection of miscellaneous actions that do not fit in other panels. In the list, one action called **Export tracks to XML** will generate the format we want. It is a simple format derived from the one used in the ISBI single-particle tracking challenge (whose results are the subject of Chenouard et al. (2014)) and suited for tracks that do not have split nor fusion events. Execute the action and a new file called `Cell_02_Tracks.xml` should be generated. Its content looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Tracks nTracks="22" spaceUnits="um" frameInterval="0.5" timeUnits="s"
generationDateTime="Wed, 5 Sep 2018 14:11:51" from="TrackMate v3.8.0">
<particle nSpots="112">
<detection t="0" x="53.66873043851335" y="11.384524705860331" z="0.0" />
<detection t="1" x="53.6447201035091" y="11.417907762121915" z="0.0" />
<detection t="3" x="53.565164363562936" y="11.45756457406076" z="0.0" />
...
```

This is what we will use in MATLAB later. You can see that each track is represented by a `particle` section, containing several `detection` items, with `t`, `x`, `y` and `z`. In our case, `z` is always 0 since we have 2D time-series.

### **?** Exercise 4.6

Perform tracking and exports for all the other movies included in the dataset. Then move on to the next section.

## 4.6 Motility Analysis with Mean-Square Displacement

Tracking is almost never the last step of an analysis workflow. Tracking tools such as TrackMate produce tracks and their role stops there. But tracks are just an intermediate data structure in the workflow. Their subsequent analysis will produce the numbers upon which we will draw a scientific conclusion. Because this track analysis is specific to the scientific question to be addressed, tracking tools remain generic and seldom include specialized analysis modules. Another toolset is required for track analysis, and in this module we will focus on using MATLAB. The main reason for this choice is that there exist ready-to-use functions to import the XML files we produced in the previous section, which underlies the importance of interoperability.

There are other alternatives however. For instance, KNIME provides excellent tools to read XML, and things such as MSD of coordinates of any dimensionality can easily be computed. See for instance Hauer et al. (2017).

### 4.6.1 Step 1: Importing Tracks into MATLAB

Close Fiji and launch MATLAB. We want to import the tracks generated above into the MATLAB workspace. Rather than writing our own XML importer, we can use one that was made specifically for TrackMate, and that is distributed with Fiji. It is called `importTrackMateTracks` and you can find it in the `scripts` folder of Fiji:



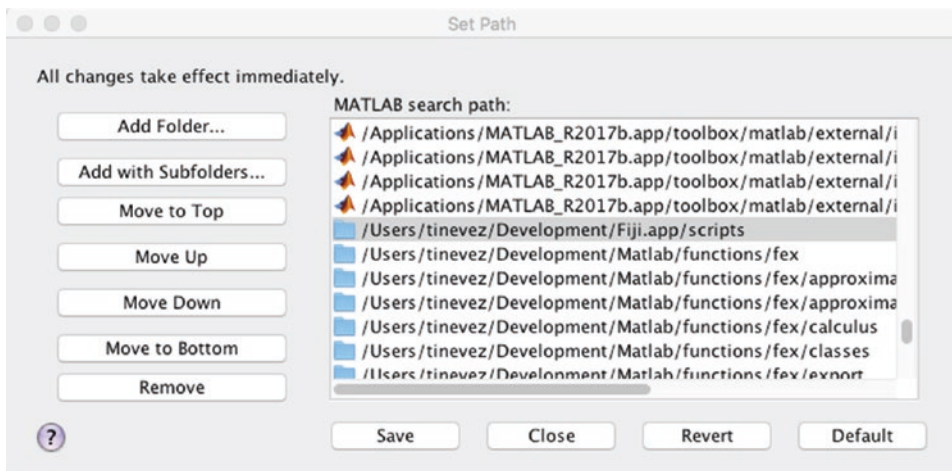
```
tinevez@lilium:~/Development/Fiji.app/scripts$ ls
ImageJ.m                copytoImg.m
InstallJava3D.m         copytoImgPlus.m
IsJava3DInstalled.m     copytoMatlab.m
Matlab3DViewerDemo_1.m  importTrackMateTracks.m
Matlab3DViewerDemo_2.m  trackmateEdges.m
Matlab3DViewerDemo_3.m  trackmateFeatureDeclarations.m
Matlab3DViewerIntroduction.m trackmateGraph.m
Miji.m                  trackmateImageCalibration.m
Miji_Test.m             trackmateSpots.m
bfopen.m
```

To make these scripts usable from MATLAB, open the path editor, and add the scripts folder to the path (■ Fig. 4.11).

This can also be achieved using `addpath('./path/to/your/Fiji.app/scripts')`; in the MATLAB prompt.

Once this is done, the functions in this folder are visible and can be called from MATLAB. For instance, we can now get the help of the function we want to use in MATLAB:

```
>> help importTrackMateTracks
| importTrackMateTracks | Import linear tracks from TrackMate
This function reads a XML file that contains linear tracks generated by TrackMate (http://fiji.sc/TrackMate). Careful: it does not open the XML TrackMate session file, but the track file exported in TrackMate using the action 'Export tracks to XML file'. This file format contains less information than the whole session file, but is enough for linear tracks
(tracks that do not branch nor fuse).
...
```



■ Fig. 4.11 Add the Fiji script folder to the MATLAB path

**? Exercise 4.7**

Read the help section of the function and try to find the correct syntax to import the tracks in a desirable way. For instance, we do not need the Z coordinates, since we dealt with a 2D dataset, and we do not need the time to be scaled by a physical units.

**✓ Solution**

The proper syntax is something along the lines of:

```
>> track_file = '/Users/tinevez/Desktop/Tracking-NEMO-movies_subset/
    NEMO-IL1/Cell_02_Tracks.xml';
>> tracks = importTrackMateTracks(track_file, true, false);
```

You need of course to specify the path to the XML file we saved in the previous section. The first flag `true` is used to specify that we do not need to import the Z coordinates, and the second flag `false` is used to specify that we want a time interval in integer units of frames.

The imported content is made of a cell list of several  $N \times 3$  arrays:

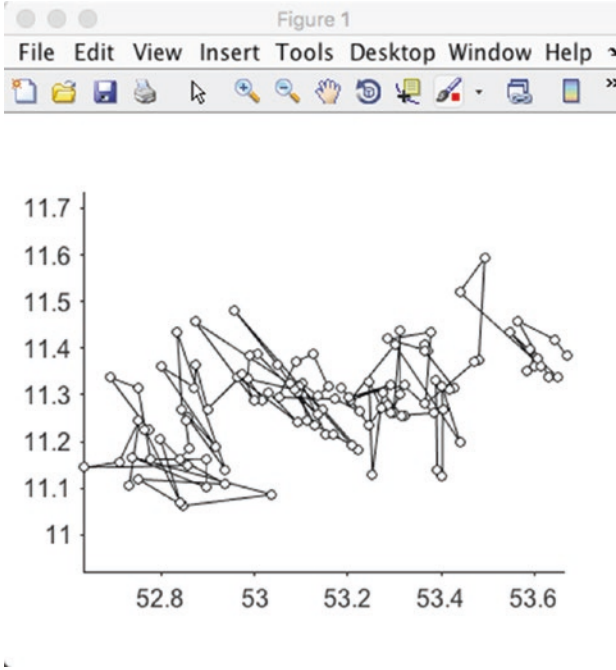
```
>> tracks
tracks =
    22x1 cell array
    112x3 double
    268x3 double
    159x3 double
    ...
```

And each array contains 3 columns with the frame, X and Y coordinates, one line per time point of a complete track:

```
>> tracks1
ans =
      0   53.6687   11.3845
  1.0000   53.6447   11.4179
  3.0000   53.5652   11.4576
  4.0000   53.6317   11.3376
  5.0000   53.6501   11.3377
  6.0000   53.5482   11.4344
    ...
```

From it, we can plot an example trajectory:

```
>> x = tracks1(:,2);
>> y = tracks1(:,3);
>> plot(x,y, 'ko-', 'MarkerFaceColor', 'w'), axis equal, box off
```



## 4.6.2 Step 2: Create and Add Data to the MSD Analyzer

As stated above, `@msdanalyzer` is a MATLAB class. If you do not know what is a class, you can think of it roughly as a collection of functions organized around a common and clearly defined data structure. The functions of a class are called methods and we will use this denomination in the following. If you followed the instruction of ► Sect. 4.3, the `@msdanalyzer` should be on the MATLAB path. You should be able to access the help for the class and the help for the constructor of the class:

```
>> help msdanalyzer
>> help msdanalyzer.msdanalyzer
```

The first instruction gives help about the class itself and the second syntax gives you help about the syntax to use when creating an analyzer. You can retrieve the list of methods defined for this class with

```
>> methods('msdanalyzer')
```

and the help for a method called `addAll` is obtained via:

```
>> help msdanalyzer.addAll
```

We need to create an analyzer first, before giving it data. This is done like this:

```
>> ma = msd_analyzer(2, 'um', 'frames')
```

Now `ma` is an empty `msd_analyzer` object, set to operate for 2D data (this is the meaning of the '2' as first argument), using  $\mu\text{m}$  as spatial units ('um' because MATLAB does not handle UTF8 characters very well) and frames as time units.

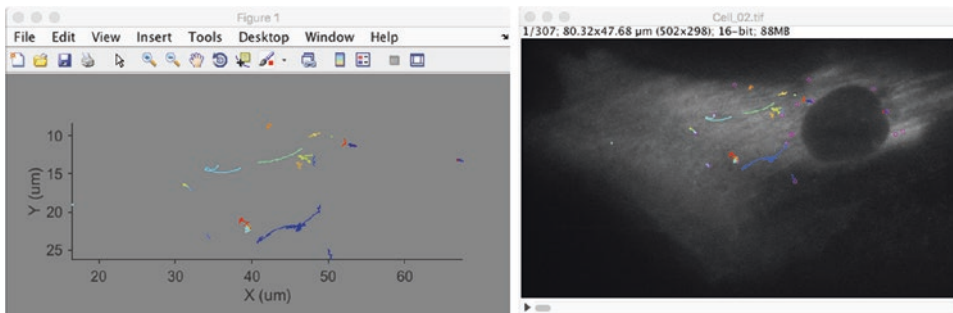
As stated above, this object is empty, and we have to feed it the tracks with the `addAll()` method. Luckily for us, as you can read in the help of the `addAll` method, it expects the tracks to be formatted exactly in the shape we have. So we can run directly:

```
>> ma = ma.addAll(tracks)
ma =
msd_analyzer with properties:
    TOLERANCE: 12
    tracks: 22x1 cell
    n_dim: 2
    space_units: 'um'
    time_units: 'frames'
    ...
```

Note that for now, it just has the 22 tracks of the first movie we analyzed. We want to add the tracks coming from the other movies *in the same category*. For instance, we will later add to the same `msd_analyzer` object all the tracks coming from all the movies of the NEMO-IL1 folder. But for now, we can use some of the methods of the `msd_analyzer` to have a nice track overview:

```
>> ma.plotTracks % Plot the tracks.
>> ma.labelPlotTracks % Add labels to the axis.
>> set(gca, 'YDir', 'reverse')
>> set(gca, 'Color', [0.5 0.5 0.5])
>> set(gcf, 'Color', [0.5 0.5 0.5])
```

In [Fig. 4.12](#), the results of these commands are displayed next to the TrackMate results. The track colors happen to be the same, but this is by chance. As a side note, look at the line 3 in the above snippet. When displaying images, the Y axis runs from top to bottom.



**Fig. 4.12** TrackMate tracks displayed in MATLAB

But MATLAB displays data in plots where it runs in the other direction, so we had to invert it here to make the tracks look like their Fiji counterparts.

### Exercise 4.8

Repeat this procedure to add all the tracks to the same `msdanalyzer` object. Be careful to use the `ma = ma.addAll( )` syntax each time.

### Solution

Supposing we continue with the `msdanalyzer` object we created above (and using the XML files that are already distributed with the dataset...):

```
>> tracks2 = importTrackMateTracks('/Users/tinevez/Desktop/Track-
    ing-NEMO-movies_subset/NEMO-IL1/ Cell_03_Tracks.xml', true, false);
>> ma = ma.addAll(tracks2);
>> tracks3 = importTrackMateTracks('/Users/tinevez/Desktop/Tracking-
    NEMO-movies_subset/NEMO-IL1/ Cell_04_Tracks.xml', true, false);
>> ma = ma.addAll(tracks3);
>> ma
    ma =
        msdanalyzer with properties:
            TOLERANCE: 12
            tracks: 614x1 cell
            n_dim: 2
            space_units: 'um'
            time_units: 'frames'
    ...
```

## 4.6.3 Interlude: A Short Word About Mean-Square Displacement Analysis

Let's consider particles undergoing Brownian motion. Let's suppose that all the particles were released from a single point at  $t=0$ , that  $r$  is the distance to this point, and that  $D$  is the diffusion coefficient of all these particles in the medium they diffuse in. We can find the equation for their density for instance in one of Einstein's historical papers (Einstein (1905)):

$$\rho(r,t) = \rho_0 \exp\left(-\frac{r^2}{4Dt}\right)$$

Using this formula, one can derive the mean square displacement (MSD) for such particles. After a delay  $\tau$ , the mean-square displacement of the particle ensemble is:

$$\text{MSD}(\tau) = \langle r^2 \rangle = 2dD\tau \quad (4.1)$$

We see that the plot of the MSD value as a function of time delay  $\tau$  should be a straight line in the case of simple freely diffusing movement. We therefore have a way to check what is the motion type of the particles. If the MSD is a line, then it is diffusing, and the slope gives us the diffusion coefficient. If the MSD saturates and has a concave curvature, then its movement is impeded: it cannot freely diffuse away from its starting point. On the

contrary, if the MSD increases faster than at linear rate, then it must be transported, because Brownian motion could not take it away that fast. See Qian et al. (1991) for a first application to biological data.

This is great, because to decide whether the erratic movement of a particle that you are observing is freely diffusive, impeded, or transported, you would only have to follow the particle for a finite amount of time. This equation can be evaluated to check what the particle movement type is. So we just need a way to evaluate it practically.

Experimentally, the MSD for a single particle is also taken as a mean. If the process is stationary (that is: the “situation”, experimental conditions, etc... do not change over time) and spatially homogeneous, the ensemble average can be taken as a time average for a single trajectory, and MSD for a single particle  $i$  can be calculated as

$$r_i^2(t, \tau) = (r_i(t + \tau) - r_i(t))^2$$

We then average over overall possible  $t$  for a given delay  $\tau$  to yield  $\text{MSD}_i(\tau)$ , and then average the resulting  $\text{MSD}_i$  over all particles. This is exactly what the `@msdanalyzer` class was built for, as we will see now.

We note that for finite trajectories, the smaller delays  $\tau$  will be more represented in the average than longer delays. For instance, if a trajectory has  $N$  points in it, the delay corresponding to one frame will have  $N - 1$  points in the average, and the delay corresponding to  $N$  frames will only have one. This has major consequences on measurement certainty, see Michalet (2010). This is one of the reason why we insisted above on having tracks that were not too short. Additionally, one has to keep in mind that processes are rarely stationary over long period of times and anomalous diffusion (any case when the MSD is not a line) processes are families of various origins which can have more specific effects on MSD.

#### 4.6.4 Step 3: Compute the Mean-Square Displacement

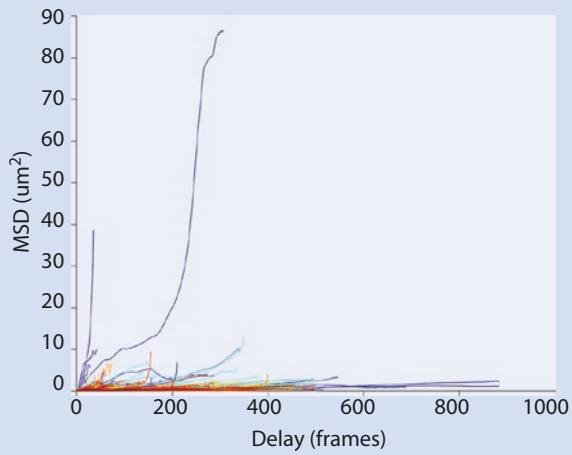
The `@msdanalyzer` automates the calculation. Using the object we prepared in step 2, calculating MSD is as simple as:

```
>> ma = ma.computeMSD
Computing MSD of 614 tracks... Done.
ma =
  msdanalyzer with properties:
    TOLERANCE: 12
    tracks: 614x1 cell
    n_dim: 2
    space_units: 'um'
    time_units: 'frames'
    msd: 614x1 cell
...

```

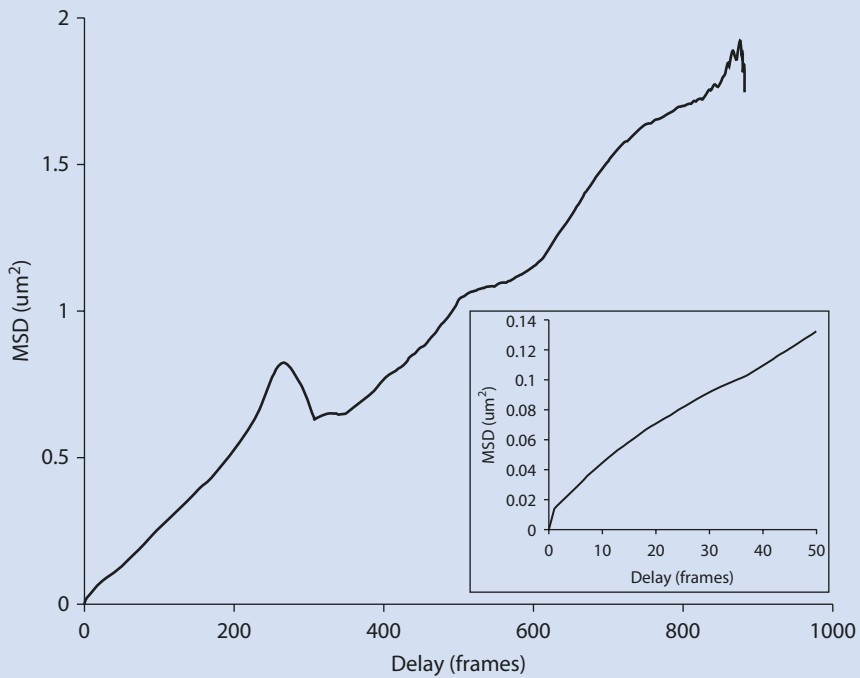
Notice that now the `msd` field of the object has some content. However interpreting it is not trivial. The plot of the individual MSD curves look like this:

```
>> ma.plotMSD
```



We can plot the ensemble-mean MSD, averaged over all particles:

```
>> ma.plotMeanMSD
```



Apart from the bumps beyond a delay of 200 frames, the ensemble curve looks like a straight line. However, looking at smaller delays, we see that this curve displays a slight concavity.

#### 4.6.5 Step 4: Log-Log Fit of the Mean-Square Displacement

4

This is not enough for us to conclude. A cell is a complex environment and each particle might have different properties that are confused in the ensemble mean plotted above. We therefore turn to another strategy.

Let's consider a single particle that diffuses freely. In that case Eq. 4.1 holds. The MSD as a function of delay  $\tau$  is a straight line. If we compute the logarithm of Eq. 4.1 we get:

$$\begin{aligned} \text{MSD}_{\text{diff}}(\tau) &= 2dD_i\tau \\ \log(\text{MSD}_{\text{diff}}(\tau)) &= \log(\tau) + \mathcal{C} \end{aligned} \quad (4.2)$$

that we can write  $y = 1 \times x + \mathcal{C}$  if  $y$  is  $(\log \text{MSD})$  and  $x$  is  $(\log \tau)$ .

Let us now consider a particle that moves with a nearly constant velocity vector. In that case,  $r$  varies linearly with  $\tau$  and MSD varies with the square of  $\tau$ . We then can write:

$$\begin{aligned} \text{MSD}_{\text{trans}}(\tau) &\propto \tau^2 \\ \log(\text{MSD}_{\text{trans}}(\tau)) &= 2 \times \log(\tau) + \mathcal{C} \end{aligned} \quad (4.3)$$

or  $y = 2 \times x + \mathcal{C}$ .

So in a log-log plot, the MSD curves can be approximated by straight lines of slope 1 for diffusion motion, 2 for transported motion, and less than 1 for constrained motion. We can therefore turn this into a test to determine the motion type of our dots. We will fit the log-log plot of the MSD curve by a line for each particle, and measure its slope *alpha*. The distribution of all slopes for a given condition will yield the motion type. We can also use the fitting approach to add an automated quality check. For instance, we can decide not to include slope values for fits with an  $R^2$  lower than 0.5.

Again, there is a method that does all of this for us in the `@msdalyzer` class:

```
% Get the description of the log-log fit function.
>> help ma.fitLogLogMSD
% Perform the fit:
>> ma = ma.fitLogLogMSD
Fitting 614 curves of log(MSD) = f(log(t)), taking only the first 25
% of each curve... Done.
% Note that now the loglogfit field of the analyzer is not empty any-
more:
>> ma.loglogfit
ans =
    struct with fields:
        alpha: [614x1 double]
        gamma: [614x1 double]
```



```

r2fit : [614x1 double]
alpha_lci: [614x1 double]
alpha_uci: [614x1 double]

```

We are interested in the slope  $\alpha$ , but first we want to remove all fits that had an  $R^2$  value lower than 0.5. The  $R^2$  values are stored in the `ma.loglogfit.r2fit` field.

```

% Logical indexing:
>> valid = ma.loglogfit.r2fit > 0.5;
>> fprintf('Retained %d fits over %d.\n', sum(valid), numel(valid))
Retained 461 fits over 614.

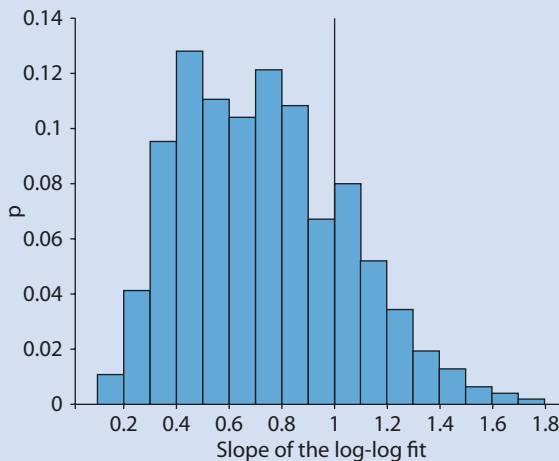
```

Now we can plot the histogram of slopes:

```

>> histogram(ma.loglogfit.alpha( valid ), 'Normalization', 'probability')
>> box off
>> xlabel('Slope of the log-log fit.')
>> ylabel('p')
>> yl = ylim;
>> line( [ 1 1 ], [ yl(1) yl(2) ], 'Color', 'k', 'LineWidth', 2)

```



#### 4.6.6 Step 5: Analysis of the Log-Log Fit

The histogram displayed above shows a peak around a slope of 1, and several other peaks below 1, around 0.4 and 0.8 judged from its shape. This suggests that there are mixed populations in our dataset, with some particles freely diffusing and others, the majority, probably constrained.

The population average behavior can be assessed by computing the mean of this distribution and checking whether it is significantly lower than 1 based on a t-test evaluation.

```
>> fprintf('Mean slope in the log-log fit: alpha = %.2f +/- %.2f
(N = %d).\n', ...
mean( ma.loglogfit.alpha(valid) ), std( ma.loglogfit.alpha(valid)),
sum(valid))
>> if (h)
    fprintf('The mean of the distribution IS significantly lower than
1 with P = %.2e.\n', p)
else
    fprintf('The mean of the distribution is NOT significantly
lower than 1. P = %.2f.\n', p)
end
Mean slope in the log-log fit: alpha = 0.73 +/- 0.31 (N = 461).
The mean of the distribution IS significantly lower than 1 with
P = 4.79e-57.
```

This ensemble analysis is not perfectly relevant however. The t-test we ran at the end gives a conclusion on the mean of the slope value, which is not exactly what we want to know. We know that there are likely to be a mixed population of particles with different motility. We expect for instance some non-specific particles to be freely diffusing or completely stuck to the substrate.

We may ask how many particles have a constrained motility and if they are the majority. A way to assess this at the single particle level is to check the confidence interval for the value of the slope in the fit. We state that if the confidence interval of the slope value is below 1, then the particles have a constrained motility. Again, things are made easy to us, as the confidence interval is also stored in the `@msdanalyzer` instance:

```
cibelow = ma.loglogfit.alpha_uci(valid) < 1;
ciin    = ma.loglogfit.alpha_uci(valid)  >= 1 & ma.loglogfit.
alpha_lci(valid) <= 1;
ciabove = ma.loglogfit.alpha_lci(valid) > 1;
fprintf('Found %3d particles over %d with a confidence interval for t
he slope value below 1.\n', ...
    sum(cibelow), numel(cibelow))
fprintf('Found %3d particles over %d with a slope of 1 inside the
confidence interval.\n', ...
    sum(ciin), numel(ciin))
fprintf('Found %3d particles over %d with a confidence interval for t
he slope value above 1.\n', ...
    sum(ciabove), numel(ciabove) )
Found 345 particles over 461 with a confidence interval for the
slope value below 1.
Found 36 particles over 461 with a slope of 1 inside the confidence
interval.
Found 80 particles over 461 with a confidence interval for the
slope value above 1.
```

This allowed us to conclude that the majority of the dots that were tracked have a constrained or sub-diffusive motility at the time-scale of their appearance. A reasonable hypothesis is that they are anchored to some static structure in the cell.

### ? Exercise 4.9

Would the conclusion have been very different if we had been much more stringent on the  $R^2$  value we used to filter out bad tracks? For instance, with  $R^2=0.8$ ?

### ✓ Solution

The distribution of alpha changes. The peak centered around 0.4 disappears, and the histogram takes the shape of a large and wide peak centered at 0.8, with a secondary, small peak around 1. The mean slope value changes accordingly, however the conclusion on the motility type is still valid.

```
>> valid = ma.loglogfit.r2fit > 0.8;
>> fprintf('Retained %d fits over %d.\n', sum(valid), numel(valid))
Retained 317 fits over 614.
>> fprintf('Mean slope in the log-log fit: alpha = %.2f +/- %.2f
(N = %d).\n', ...
mean(ma.loglogfit.alpha(valid)),      std(ma.loglogfit.alpha(valid)),
sum(valid))
>> if (h)
    fprintf('The mean of the distribution IS significantly lower than
1 with P = %.2e.\n', p)
else
    fprintf('The mean of the distribution is NOT signifi-
cantly lower than 1. P = %.2f.\n', p)
end
Mean slope in the log-log fit: alpha = 0.86 +/- 0.28 (N = 317).
The mean of the distribution IS significantly lower than 1 with
P = 2.31e-17.
```

### ? Exercise 4.10

Redo all the analysis for the control condition. In our case, the control condition corresponds to cells that were not stimulated. The dots we observed then were permanent instead of being transient when the cells were stimulated. They probably correspond to some spurious particles.

### ✓ Solution

We can re-execute the whole approach displayed above on the two movies in the Control folder:

```
clear all
close all
clc
tracks1 = importTrackMateTracks('/Users/tinevez/Desktop/Tracking-
NEMO-movies_subset/NEMO-Ctrl/ Cell_01_Tracks.xml', true, false);
tracks2 = importTrackMateTracks('/Users/tinevez/Desktop/Tracking-
NEMO-movies_subset/NEMO-Ctrl/ Cell_02_Tracks.xml', true, false);
ma = msdalyzer(2, 'um', 'frames');
```

```

ma = ma.addAll(tracks1);
ma = ma.addAll(tracks2);
ma = ma.computeMSD;
ma = ma.fitLogLogMSD;
valid = ma.loglogfit.r2fit > 0.8;
fprintf('Retained %d fits over %d.\n', sum(valid), numel(valid))
fprintf('Mean slope in the log-log fit: alpha = %.2f +/- %.2f (N = %d).\n', ...
    mean(ma.loglogfit.alpha(valid)), std(ma.loglogfit.alpha(valid)), sum(
    valid))
[h, p] = ttest( ma.loglogfit.alpha(valid), 1, 'tail', 'left');
if (h)
    fprintf('The mean of the distribution IS significantly lower than 1 with P = %.2e.\n', p)
else
    fprintf('The mean of the distribution is NOT significantly lower than 1. P = %.2f.\n', p)
end

```

And the output is:

```

Computing MSD of 19 tracks... Done.
Fitting 19 curves of log(MSD) = f(log(t)), taking only the first 25%
of each curve... Done.
Retained 13 fits over 19.
Mean slope in the log-log fit: alpha = 0.87 +/- 0.24 (N = 13).
The mean of the distribution IS significantly lower than 1 with
P = 3.41e-02.

```

The majority of non-specific particles appears to also be stuck. So what is the difference with the IL1-stimulation condition? In this case, the conclusion on motility is the same but it does not apply to the same particles. The control condition movies are made of the few cells we could find that had fluorescent dots that were visible without stimulation. What matters is that there are few of them and that their number is not enough to change the scientific conclusion on the dots that appear transiently upon stimulation, regardless of their motility.

## 4.7 Results and Conclusion

This module is one part of the work that helped us conclude on the NEMO dot motility. The MSD analysis indicated that the dots made of NEMO-eGFP proteins that appear upon stimulation by IL1 are anchored to some static structure of the cell during the little time they are visible.

We then turned to investigate what this static structure could be. So we repeated the analysis you just did on cells for which we depolymerized actin filaments and microtubules. The conclusion did not change. There was still the same proportion of NEMO dots with the same constrained motility type.

We then led other investigations, relying on biochemistry and confocal imaging, and concluded that NEMO dots are anchored at the cell membrane. The membrane is fluid and the anchor point might be diffusing, but we do not see this behavior on the time-scale of the live-imaged NEMO dots. The whole story and more can be found in the original paper Tarantino et al. (2014).

### Take Home Message

We hope this module serves as an example and shows that biophysics and image analysis can provide new approaches to a scientific question that would otherwise solely rely on biochemistry. The original paper contains quite some heavy biochemistry studies, but their results are reinforced by the orthogonal approach presented here.

We relied on mean-squared-displacement analysis to reach a conclusion on the motility type. This is the historical method and the first to have been applied on biological data Qian et al. (1991). Its main drawback is that the tracks need to be long and the detections accurate to have a decent accuracy on the quantities MSD analysis yields. Good, accurate tracks are especially difficult to obtain in many life-science cases, so several research labs have been working on developing new methods improving on MSD. We can cite for instance the work of Hansen et al. (2018) based on analyzing step distributions, or Briane et al. (2018) that relies on a statistical approach.

**Acknowledgements** We are very grateful to Emmanuel Laplantine and Nadine Tarantino, our fellow authors on the paper, that agreed to make the raw data publicly available. We thank Jan Eglinger (Friedrich Miescher Institute for Biomedical Research, Basel) for reviewing this chapter.

## Bibliography

- Briane V, Kervrann C, Vimond M (2018) Statistical analysis of particle trajectories in living cells. *Phys Rev E* 97:062121 <https://doi.org/10.1103/PhysRevE.97.062121>. <https://link.aps.org/doi/10.1103/PhysRevE.97.062121>
- Chenouard N, Smal I, de Chaumont F, Maška M, Sbalzarini IF, Gong Y, Cardinale J, Carthel C, Coraluppi S, Winter M, Cohen AR, Godinez WJ, Rohr K, Kalaidzidis Y, Liang L, Duncan J, Shen H, Xu Y, Magnusson KE, Jalden J, Blau HM, Paul-Gilloteaux P, Roudot P, Kervrann C, Waharte F, Tinevez JY, Shorte SL, Willemsse J, Celler K, van Wezel GP, Dan HW, Tsai YS, Ortiz de Solorzano C, Olivo-Marin JC, Meijering E (2014) Objective comparison of particle tracking methods. *Nat Methods* 11(3):281–289
- Einstein A (1905) Investigations on the theory of the brownian movement. *Ann der Physik*. [http://www.physik.fu-berlin.de/~kleinert/files/eins\\_brownian.pdf](http://www.physik.fu-berlin.de/~kleinert/files/eins_brownian.pdf)
- Hansen AS, Woringer M, Grimm JB, Lavis LD, Tjian R, Darzacq X (2018) Robust model-based analysis of single-particle tracking experiments with spot-on. *eLife* 7:e33125. ISSN: 2050-084X. <https://doi.org/10.7554/eLife.33125>
- Hauer MH, Seeber A, Singh V, Thierry R, Sack R, Amitai A, Kryzhanovska M, Eglinger J, Holcman D, Owen-Hughes T, Gasser SM (2017) Histone degradation in response to DNA damage enhances chromatin dynamics and recombination rates. *Nat Struct Mol Biol* 24(2):99–107
- Jaqaman K, Loerke D, Mettlen M, Kuwata H, Grinstein S, Schmid SL, Danuser G (2008) Robust single-particle tracking in live-cell time-lapse sequences. *Nat Methods* 5(8):695–702
- Kalman RE (1960) A new approach to linear filtering and prediction problems. *Trans ASME J Basic Eng* 82(Series D):35–45

- 4
- Michalet X (2010) Mean square displacement analysis of single-particle trajectories with localization error: Brownian motion in an isotropic medium. *Phys Rev E Stat Nonlin Soft Matter Phys* 82(4 Pt 1):041914
- Qian H, Sheetz MP, Elson EL (1991) Single particle tracking. analysis of diffusion and flow in two-dimensional systems. *Biophys J* 60(4):910–921. ISSN: 0006-3495. [https://doi.org/10.1016/S0006-3495\(91\)82125-7](https://doi.org/10.1016/S0006-3495(91)82125-7). <http://www.sciencedirect.com/science/article/pii/S0006349591821257>
- Tarantino N, Tinevez J-Y, Crowell EF, Boisson B, Henriques R, Mhlanga M, Agou F, Israël A, Laplantine E (2014) TNF and IL-1 exhibit distinct ubiquitin requirements for inducing NEMO–IKK supramolecular structures. *J Cell Biol* 204(2):231–245. ISSN: 0021-9525. <https://doi.org/10.1083/jcb.201307172>. <http://jcb.rupress.org/content/204/2/231>
- Tinevez J-Y, Dragavon J, Baba-Aissa L, Roux P, Perret E, Canivet A, Galy V, Shorte S (2012) A quantitative method for measuring phototoxicity of a live cell imaging microscope, chap 15. In: Michael Conn P (ed) *Imaging and spectroscopic analysis of living cells*. *Methods in enzymology*, vol 506, pp 291–309. Academic. <https://doi.org/10.1016/B978-0-12-391856-7.00039-1>. <http://www.sciencedirect.com/science/article/pii/B9780123918567000391>
- Tinevez J-Y, Perry N, Schindelin J, Hoopes GM, Reynolds GD, Laplantine E, Bednarek SY, Shorte SL, Eliceiri KW (2017) TrackMate: an open and extensible platform for single-particle tracking. *Methods* 115: 80–90. ISSN: 1046-2023. <https://doi.org/10.1016/j.jymeth.2016.09.016>. <http://www.sciencedirect.com/science/article/pii/S1046202316303346> Image Processing for Biologists

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (► <http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

