Measurements of Intensity Dynamics at the Periphery of the Nucleus

Kota Miura

K. Miura, N. Sladoje (eds.), *Bioimage Data Analysis Workflows*, Learning Materials in Biosciences,
https://doi.org/10.1007/978-3-030-22386-1_2

What You Learn from This Chapter

The aim of this chapter is to learn how to construct a workflow for measuring the fluorescence intensity localized to the nuclear envelope. For this purpose, the nucleus image is segmented to create a mask along the nuclear rim. The reader will learn a typical technique for automatically delineating the segmented area by post-processing using the mathematical morphology algorithm, and how to loop that piece of ImageJ macro and iterate through multiple image frames to measure changes in fluorescence intensity over time. This chapter is also a good guide for learning how to convert ImageJ macro commands recorded by the Command Recorder to a stand-alone ImageJ macro.

2.1 Introduction

In some biological research projects, we encounter problems that should be studied by measuring fluorescence intensity at the boundary between two different compartments. Here, we pick up an example analysis of the Lamin B receptor protein density targeting inner nuclear membrane. The protein changes its location from the cytoplasmic area (Endoplasmic Reticulum, ER) to the nuclear envelope (Boni et al. 2015).

We analyze a two-channel time-lapse image stack, a sequence of the process of the protein re-localization that causes increases in the protein density at the nuclear envelope. The data was acquired by Andreas Boni (Jan Ellenberg lab, EMBL Heidelberg) and have been used in many training workshops in EMBL as a great example for learning bioimage analysis. His work, with more advanced bioimage analysis workflows for analyzing the protein targeting dynamics, is published in The Journal of Cell Biology (Boni et al. 2015). Those codes and image data used in his study, which might be interesting for you after going through this chapter, are accessible through the supplementary data section in the journal website.[1]

Two images shown in ◘ Fig. 2.1 are from the first and the last time points of a time-lapse sequence.[2] Compare these images carefully. The green signal broadly distributed in the cytoplasmic area at time point 1 becomes accumulated at the periphery of nuclei (red) at time point 15—between these image frames, the signal changed its localization from ER to the nuclear envelope. We construct a workflow that measures this accumulation process by writing an ImageJ macro. The workflow involves two steps: First, we segment the rim of nucleus—nuclear membrane—using the first channel (histone). Second, we use that segmented nuclear rim as a mask to measure the intensity changes over time in the second channel.

Segmentation of nucleus using its marker (e.g. DAPI) is a popular image analysis technique used in many biological research projects, but to measure more specific location—in our case nuclear envelope—we need to add several more steps to refine the region-of-interest. When we are successful in determining the area of nuclear envelope, the measurement of intensity in that region over time is rather trivial. We just need to loop the same process for each time point. Especially for the analysis of time-lapse sequence, programming is highly recommended to iterate the measurement for each time point.

1 ► http://jcb.rupress.org/content/209/5/705

2 The images shown in the ◘ Fig. 2.1 are from a 4D hyperstack "NPC1.tif", which can be downloaded using ImageJ plugin "CMCI-EMBL". More details are in "Dataset" section.

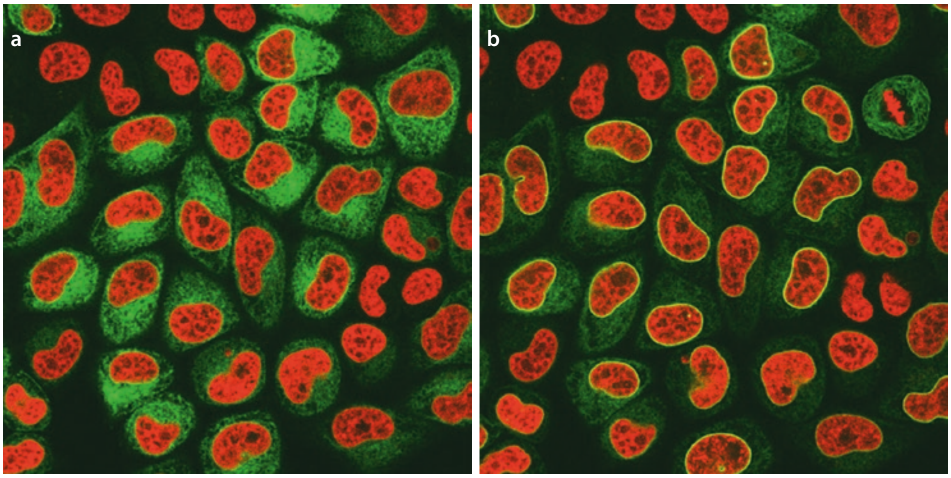


Fig. 2.1 Lamin receptor localization difference at two time points: More Lamin receptor in nucleus periphery. **a** Time point 1. **b** Time point 15

This chapter should be a good guide not only limited to study the intensity changes occurring at the nuclear envelope, but also in general for segmenting the edge (perimeter) of biological compartments such as the edge of organelle, plasma membrane and tissue boundaries. In principle, similar post-processing strategy is also applicable to 3D volumes by using 3D morphology filters.

2.2 Tools

We use Fiji (Fiji Is Just ImageJ) for image analysis.

- Fiji
  - Download URL: ► https://imagej.net/Fiji/Downloads
  - Please choose the latest version.

In addition, a plugin is required for loading the sample image data. Using the "Update sites" function, please add "CMCI-EMBL" to your Fiji installation. Please restart Fiji after this plugin installation.

2.3 Dataset

All ImageJ macro codes can be downloaded from the Github repository.[3]

The image data we used in this chapter can be downloaded using the plugin "CMCI-EMBL". After installation of this plugin, select the menu item `[EMBL > Sample Images > NPCsingleNucleus.tif]` to load the image data. This is a time-lapse

3 ► https://github.com/miura/NucleusRimIntensityMeasurementsV2/

sequence of a cell, extracted from "NPC1.tif" which can be also downloaded through the same plugin.

- Cell Type: Hela Cells
- Scale: 0.165 µm/pixel
- Frame Rate: 400 Sec/Frame
- Channels
  - Red channel (C1): H2B-mCherry (ex:561nm)
  - Green Channel (C2): Lamin B Receptor-GFP (ex:488nm)

2.4 Workflow

To simplify the development, we focus on a single cell/nucleus to construct the workflow. Load the image stack **NPCsingleNucleus.tif**. This is a hyperstack sequence. Slide the scroll bar at the bottom back-and-forth to watch the process of intensity changes. H2B-mCherry signal (red), used as a marker for nucleus, is more or less constant with its distribution. On the other hand, the Lamin receptor signal (green) exhibits strong accumulation to the nuclear membrane. To study this accumulation process, our aim is to measure the intensity changes of green signal intensity at the rim of the nucleus over time. The outline of the workflow is shown in the diagram (Fig. 2.2).

To achieve this aim we first need to identify the region of nucleus rim ("segmentation")—in other words, we create a mask of the nucleus rim. Using this mask we measure the changes in intensity over time.

2.4.1 Segmentation of Nucleus Rim

We first write a macro for the nucleus rim segmentation by taking following steps:

1. Split the original multi-channel image stack and create two image stacks of each channel for processing them independently (Fig. 2.3a)
2. Blur the image to attenuate noise (Fig. 2.3b)
3. Nucleus segmentation: Binarize the image by intensity thresholding (Fig. 2.3c)
4. Remove other Nuclei: At the right-bottom corner of the image, a small part of different nucleus is present. This should be removed.
5. Duplicate the image
   (a) Erode the original (Fig. 2.3e)
   (b) Dilate the duplicated (Fig. 2.3d)
6. Subtract the eroded from the dilated (Fig. 2.3f)

In the following we record these steps as macro commands using the Command Recorder (`[Plugins > Macros > Record...]`). We recommend you NOT to launch the command recorder from the beginning. Please first try to reproduce the workflow using mouse and the graphical user interface (GUI). This is like a rehearsal before recording your actions. When you become clear with the steps you have to take, record the processing steps. When you use the command recorder, be sure that "Macro" is selected in the "Record:" drop down menu at the top-left corner of the recorder.

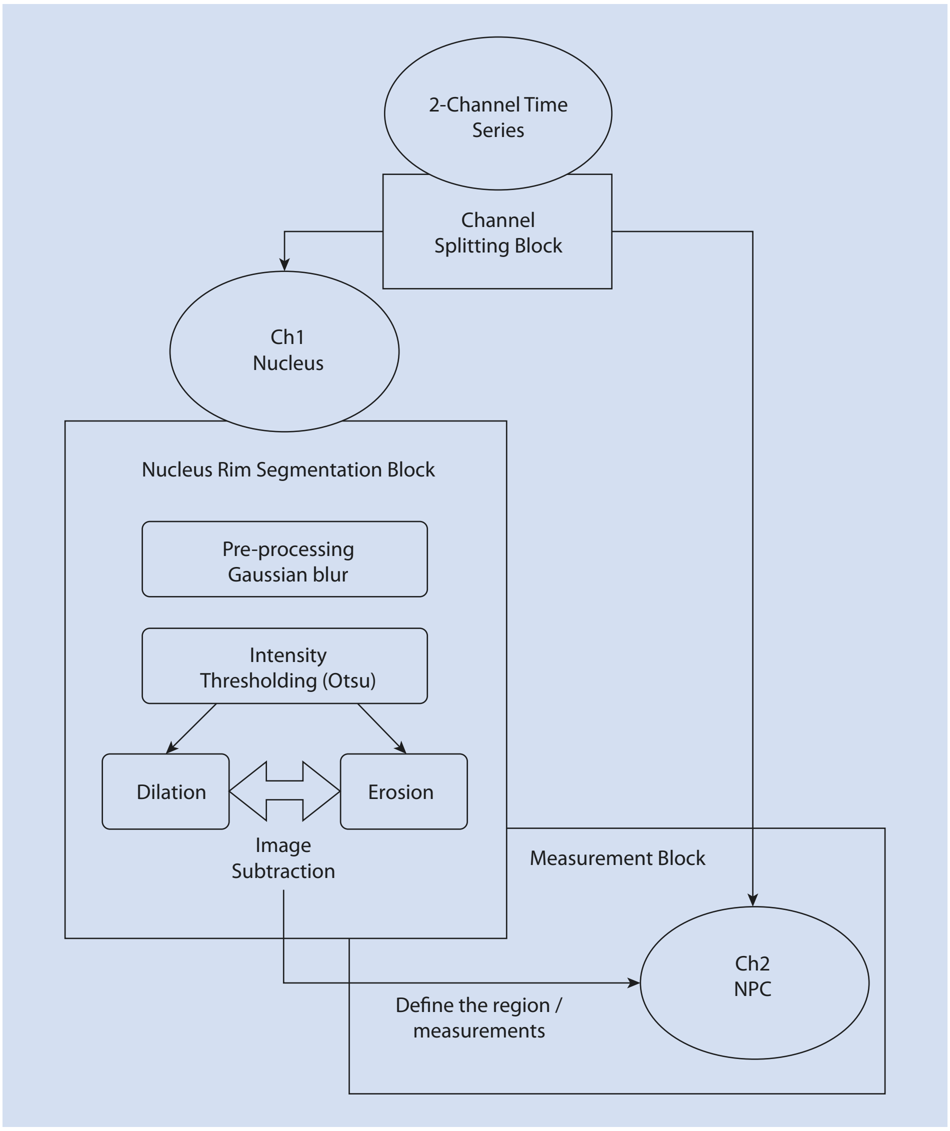

2-Channel Time Series
Channel Splitting Block
Ch1 Nucleus
Nucleus Rim Segmentation Block
Pre-processing Gaussian blur
Intensity Thresholding (Otsu)
Dilation
Erosion
Image Subtraction
Measurement Block
Ch2 NPC
Define the region / measurements

Fig. 2.2 The outline of the workflow

2.4.1.1 Block 1: Splitting Channels

To split the multichannel image stack from the GUI menu, do `[Image > Color > Split Channels]`. In the Recorder you will see the following command.

```
run("Split Channels");
```

`run` function is the most frequently used build-in macro function.

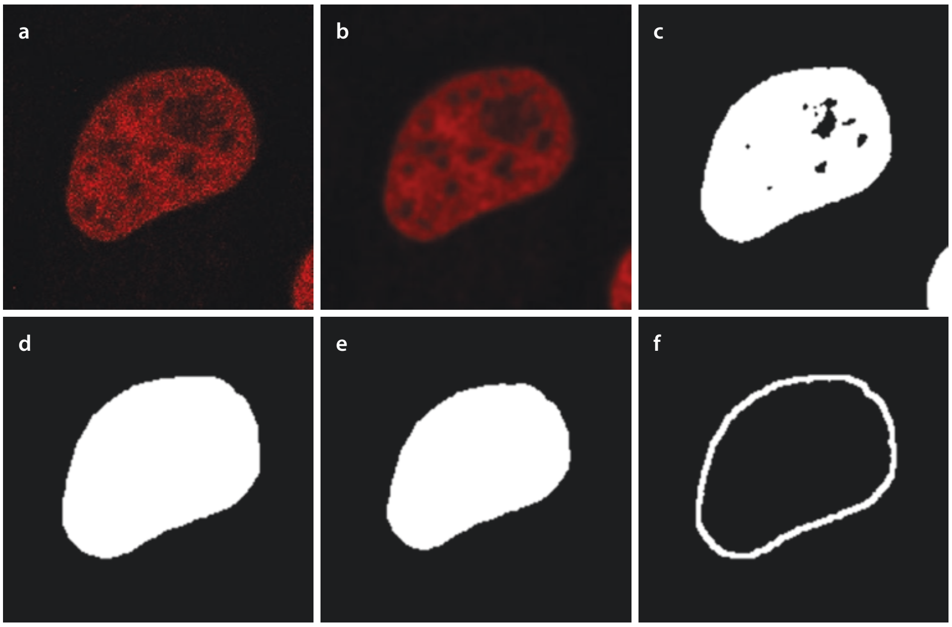


Fig. 2.3 The strategy of segmentation. **a** Original nucleus image. **b** Blurred nucleus image. **c** Binarized image, after thresholding. **d** Dilated binary image. **e** Eroded binary image. **f** Subtraction result, the rim

run("command"[, "options"])

Executes an ImageJ menu command. The optional second argument contains values that are automatically entered into dialog boxes (must be GenericDialog or OpenDialog). Use the Command Recorder (Plugins>Macros>Record) to generate run() function calls. Use string concatenation to pass a variable as an argument. With ImageJ 1.43 and later, variables can be passed without using string concatenation by adding "&" to the variable name.

The run function takes a menu item as the first argument and optional values (values you fill-in in a dialog window) in the second argument. In case of channel splitting, there is no such optional value so the second argument is ignored.

We then process the nucleus image. Click the nucleus image window to bring it up to the top—We call this action as "activating a window". By this clicking, we activated Channel 2 (red, nucleus image).

Please confirm that a new command shown below, is added to the recorder after activating the nucleus image.

```
selectWindow("C1-NPCsingleNucleus.tif");
```

…Here is the explanation from the macro function reference.

selectWindow("name")

Activates the window with the title "name".

This function takes a window title as an argument and activates a window with that title. When we used mouse to activate the nucleus channel window, we did it manually by visually recognizing the red nucleus image. On the other hand, in macro, we need to know the title of the windows of each individual channels to activate a specific window to provide it to macro as an argument of "selectWindow" command. How can we get the name of the nucleus channel window after splitting the channels of the original image?

Standard behavior of "Split Channel" command is that it automatically names the resulting stacks of individual channels by prefixing "C1-" or "C2-" or "C3" to the original image title. Based on this known behavior, we can construct these names if the original image title is known. For this we use the command `getTitle( )` which returns the title of currently active window as a string.

getTitle()

Returns the title of the current image.

Here is the code to activate the nucleus channel automatically after the splitting. More importantly, we also acquire "image ID". This will be explained later.

code/code_block1_ChannelSplitting.ijm

```
1  orgName = getTitle();
2  run("Split Channels");
3  c1name = "C1-" + orgName;
4  c2name = "C2-" + orgName;
5  selectWindow(c1name);
6  c1id = getImageID();
7  selectWindow(c2name);
8  c2id = getImageID();
```

Details:

- The first line grabs the window title as a string variable "orgName".
- The second line splits the stacks to each individual channel stack.
- 3rd and 4th lines compose the window title for each channel stack.
- 5th line activates the channel 1 stack.
- 6th line acquires the image ID of channel 1 stack.
- 7th line activates the channel 2 stack.
- 8th line acquires the image ID of channel 2 stack.

In lines 6 and 8, we acquire **image IDs**. Here is some more explanation about this: Each window has a unique ID number. To get this ID number from each image we use the command `getImageID( )`.

getImageID()

Returns the unique ID (a negative number) of the active image. Use the selectImage(id), isOpen(id) and isActive(id) functions to activate an image or to determine if it is open or active.

A window can be activated by `selectWindow` using its window title, but this could have a problem if there is another window with same name. Image ID has less problem since it is uniquely given to each window. To select a window using image ID, we use `selectImage(ID)` command.

selectImage(id)

Activates the image with the specified ID (a negative number). If id is greater than zero, activates the ID-th image listed in the Window menu. The ID can also be an image title (a string).

We acquire image IDs just after the splitting. From here on, we will use image IDs when we want to specify the image window we want to work on and to activate it.

Exercise 2.1

Test the code below and run it on several image windows. Confirm that each window has an unique ID number. Please ignore the line numbers when you write the code.

```
id = getImageID();
print(id);
name = getTitle();
print(name);
```

Save the channel splitting macro. When you name the file, add an extension ".ijm", as this indicates that the file is an ImageJ macro. This is only a part of the final workflow, and we call such part as a **"block"** of the workflow, and by assembling blocks with various functions, we construct a **workflow**. A block is a functional unit within the workflow. Each block is consisting of several **components**, each of which is the build-in function that implements a certain algorithm (see ▶ Chap. 1).

In the current case, we just finished the **Channel Splitting Block**, consisting of a channel splitter component, a window title getter component, an image window ID getter component, and window selector components.

2.4.1.2 Block 2: Segmentation of Nucleus Rim

Now we start working on the segmentation of nucleus rim. For this, we use only the nucleus image stack (channel 1) we got in the Block 1. Create a new tab in the script editor by `[File > New]`. We use this blank editor to write the next **block** for the detection of nucleus rim. We assemble all blocks as a single workflow later.

Following is the step-by-step procedure. Try first using the GUI (your mouse and the menu bar!). Then launch the Command recorder, redo the steps to record the history of commands. I recommend you to do so mainly because the initial trials with GUI let you visually understand what is going on, and also to get used to the sequence of operation for the command recording.

1. Gaussian Blur
   - `[Process > Filter > Gaussian Blur]`, sigma = 1.5, tick "Do Stack"
   - This diminutive blurring of the image attenuates noise and allows a better result for the segmentation.

2. Find Threshold
   - `[Image > Adjust > Threshold]`, select Otsu method
   - This simply changes the LUT, but not the data
3. Apply Threshold: Click 'Apply'
   - Changes the data to black and white using the threshold value using the Otsu method.
4. Find Threshold again (Otsu method)
   - We do this again for selecting the nucleus for the "AnalyzeParticle" in the following step.
5. Analyze Particles
   - `[Analyze > Analyze Particles]`
   - Options:
     - Size: 800-Infinity
     - Tick "Pixel Units"
     - Circularity: default (0–1.0)
     - Show: Mask
     - Tick Display Results, Clear results, Exclude on edges, Include holes.
   - We use AnalyzeParticle as a filter for segmented object. In our case, this filtering removes nucleus touching the edge of image. This way of usage is also effective in removing small none-nucleus signals.
6. Invert the LUT of the "Mask" created by AnalyzeParticle, so operations to be done in the following recognizes nucleus as the target of Dilation and Erosion.
   - `[Image > Look-up Table > Invert LUT]`
7. Duplicate the "Mask" Stack, and then apply "Dilation" to the original stack and apply "Erosion" to the duplicated.
   - `[Image > Duplicate]`
   - Set Iterations `[Process > Binary > Options]`
     - iterations 2 or 3
     - Tick dark background
   - Original: Dilate `[Process > Binary > Dilate]`
     - This increases the edge of nucleus by 2 or 3 pixels.
   - Duplicate: Erode `[Process > Binary > Erode]`
     - This decreases the edge of nucleus by 2 or 3 pixels.
8. Image Subtraction
   - `[Process > Image Calculator]`
   - tick "keep original", compute the difference of Dilated and Eroded.
     - Result: a band of 4 or 6 pixels at the edge of nucleus.

When you are done with the macro recording, check the results in the recorder. Below is an example of the output from the recorder.

code/code_block2_recordNucSeg.ijm
```
selectWindow("C1-NPCsingleNucleus.tif");
run("Gaussian Blur...", "sigma=1.50 stack");

 //run("Threshold...");
setAutoThreshold("Otsu dark");
```

```
6   setOption("BlackBackground", true);
7   run("Convert to Mask", "method=Otsu background=Dark calculate
    black");
8   //run("Threshold...");
9   run("Analyze Particles...", "size=800-Infinity pixel circular-
    ity=0.00-1.00 show=Masks display exclude clear include stack");
10  run("Invert LUT");
11  run("Duplicate...", "title=[Mask of C1-NPCsingleNucleus-1.tif]
    duplicate range=1-15");
12  selectWindow("Mask of C1-NPCsingleNucleus.tif");
13  run("Options...", "iterations=2 count=1 black edm=Overwrite
    do=Nothing");
14  run("Dilate", "stack");
15  selectWindow("Mask of C1-NPCsingleNucleus-1.tif");
16  run("Erode", "stack");
17  imageCalculator("Difference create stack", "Mask of C1-
    NPCsingleNucleus.tif", "Mask of C1-NPCsingleNucleus-1.tif");
18  selectWindow("Result of Mask of C1-NPCsingleNucleus.tif");
```

This recorded macro already runs properly as it is, but there is a problem: the code works only with image data with a specific window title. See the line 1. The command looks like this.

```
selectWindow("C1-NPCsingleNucleus.tif");
```

The window title given in the argument of `selectWindow` is hard-coded, so that if you need to apply this macro to a image data with a different window title, it will not work. The macro needs to be improved to allow the general applicability to other images.

For this reason, we need to change the code so that it uses ImageID instead of a fixed image title. Since the ImageID of the nucleus channel was already acquired after splitting the original image, we can use that ID to activate a specific image window.

As we are working separately from the channel splitting block, we assume that the nucleus channel stack is active and is the top window at the starting of current code. We replace the first line `selectWindow` with `getImageID( )` command to capture the ID number of the nucleus image window. Next, we need to add `getImageID` in line 10 and 13 to capture IDs of newly created windows. Due to these changes, we need to replace `selectWindow` in line 12 and 15 to `selectImage` to consistently use ImageID for accessing specific window. After these replacement, the updated code will look like the one shown below.

code/code_block2_recordNucSegV2.ijm

```
1   orgID = getImageID();
2   run("Gaussian Blur...", "sigma=1.50 stack");
3
4   //run("Threshold...");
5   setAutoThreshold("Otsu dark");
6   setOption("BlackBackground", true);
7   run(Convert to Mask", "method=Otsu background=Dark calculate
    black");
8   //run("Threshold...");
9   run("Analyze Particles...", "size=800-Infinity pixel circular-
    ity=0.00-1.00 show=Masks display exclude clear include stack");
10  dilateID = getImageID();
```

```
11  run("Invert LUT");
12  run("Duplicate...", "title=[Mask of C1-NPCsingleNucleus-1.tif]
    duplicate range=1-15");
13  erodeID = getImageID();
14  //selectWindow("Mask of C1-NPCsingleNucleus.tif");
15  selectImage(duplicateID);
16  run("Options...", "iterations=2 count=1 black edm=Overwrite
    do=Nothing");
17  run("Dilate", "stack");
18  //selectWindow("Mask of C1-NPCsingleNucleus-1.tif");
19  selectImage(erodeID);
20  run("Erode", "stack");
21  //imageCalculator("Difference create stack", "Mask of C1-
    NPCsingleNucleus.tif","Mask of C1-NPCsingleNucleus-1.tif");
22  imageCalculator("Difference create stack", dilateID, erodeID);
```

Here is the explanation of what was done.

- line 1: The first line is replaced with the `getImageID( )` command.
- line 10: `getImageID( )` command was inserted for a new image created by Analyze Particle command (in line 9). The new image is the mask that is eliminated with edge-touching nucleus.
- line 13: `getImageID( )` command was inserted for the duplicated image.
- line 15: The `selectWindow` command in line 14 was commented out and replaced by the `selectImage` command.
- line 19: `selectWindow` command is replaced by the `selectImage` command.
- line 22: Because we now have ImageIDs of both dilated and eroded images, we replace the titles of image windows with imageIDs for image calculator arguments. Compare the line 21 (commented out) and the line 22.

We are now almost done with the generalization of the nucleus rim segmentation block, but there still is a part that can be more general instead of a fixed window name. See line 12. This line uses `run` command to duplicate the "Mask" stack.

`run("Duplicate...", "title=[Mask of C1-NPCsingleNucleus-1.tif] duplicate range=1-15");`

The first argument "Duplicate..." is the name of the menu item `[Image > Duplicate…]` and this is OK.

The second argument contains multiple optional values you chose in the GUI. The first is the title of the duplicated image, that was automatically created by suffixing "-1" to the image title. Square brackets surrounding this new image title is for avoiding the problem with spaces in the image title, because spaces are used as the separator for the options in the second argument. **`duplicate`** is a keyword of a checkbox in the duplication dialog, for choosing whether to duplicate multiple frames in a stack or just a single currently shown frame. The third option is the frame range (`range=`), which defines the range of frames to be duplicated. Since we want to duplicate all frames, the range is set to `1-15`, from the first frame to the last 15th frame.

Within this second argument, two values in this command are not flexible enough for applying the macro to other images with different names. First is the image title. We better have a more general name for the duplicated image. The second is the frame range. The duplication of full stack is better be applicable for stacks with any number of frames, not limited to 15-frames stacks. We can construct the option string of the second argument as shown below to allow the general applicability of the macro.

```
options = "title = dup.tif duplicate range=1-" + nSlices
```

nSlices is a macro function that returns the number of frames or slices in the current stack. This macro function allows the duplication all frames of a stack, regardless of the number of frames within that stack.

We can now replace the second argument for image duplication by this new variable `options`.

```
run("Duplicate...", options);
```

Exercise 2.2

Create a new script tab and write the code below (please ignore the line numbers when you write the code). Run the code with various stacks with different slice or frame numbers and confirm that this short macro successfully duplicate stacks with any slice or frame numbers.

```
1  print(nSlices);
2  options = "title=dup.tif duplicate range=1-" + nSlices;
3  print(options);
4  run("Duplicate...", options);
```

Below is the upgraded code. All the lines previously commented out were removed, and line 10 was inserted for preparing options for the `Duplicate` command. In addition, we added line 19–24 for closing all images that are not needed anymore.

code/code_block2_recordNucSegV3.ijm

```
1   orgID = getImageID();
2   run("Gaussian Blur...", "sigma=1.50 stack");
3
4   setAutoThreshold("Otsu dark");
5   setOption("BlackBackground", true);
6   run("Convert to Mask", "method=Otsu background=Dark calculate
    black");
7   run("Analyze Particles...", "size=800-Infinity pixel circular-
    ity=0.00-1.00 show=Masks display exclude clear include stack");
8   dilateID = getImageID();
9   run("Invert LUT");
10  options = "title = dup.tif duplicate range=1-" + nSlices;
11  run("Duplicate...", options);
12  erodeID = getImageID();
13  selectImage(dilateID);
14  run("Options...", "iterations=2 count=1 black edm=Overwrite
    do=Nothing");
15  run("Dilate", "stack");
16  selectImage(erodeID);
17  run("Erode", "stack");
18  imageCalculator("Difference create stack", dilateID, erodeID);
19  selectImage(dilateID);
20  close();
21  selectImage(erodeID);
22  close();
23  selectImage(orgID);
24  close();
```

We now have a block that segments nucleus rim. Save this code, and we are done with the second block of the workflow.

2.4.1.3 Block 3: Intensity Measurement Using Mask

Using the isolated nucleus rim image, we can specify the region for measuring the fluorescence intensity in the Lamin receptor channel. This will be the third block of the workflow.

Before start writing the third block of the workflow, we do a small preparation. We merge the rim-segmented stack and the Lamin receptor stack to create a multi-channel stack, which will be used as the input image data of this third block. Open the rim binary image (if you closed it already, run the second block macro again to regenerate it!) and the Lamin receptor image.

Two stacks can be merged to a two channel image stack by the following command.

`[Image > Color > Merge Channels…]` In the dialogue window, assign red color (C1) to the nucleus channel (nucleus rim binary image), and green color (C2) to the NPT channel. Make sure that "Create composite" is ticked. Clicking "OK" button, you will have an image stack that looks like Fig. 2.4.

We are now ready to start writing the third block of the workflow. Please follow the steps below using GUI. When you become sure with the operations, record your operations using Command Recorder.

1. `[Image > Color > Split Channels…]`
2. `[Analysis > Set Measurements…]`
   - You will see a dialog window with many check boxes (Fig. 2.5). Among the parameters to be measured, tick at least `Area, Mean gray value` and `Integrated density`. Integrated density is the sum of all pixel values.
3. Activate the rim image and do `[Edit > Selection > Create Selection]`
   - This selects the background, not the rim.
4. `[Edit > Selection > make Inverse]`
   - Inverting the selection, now we are selecting the nucleus rim.

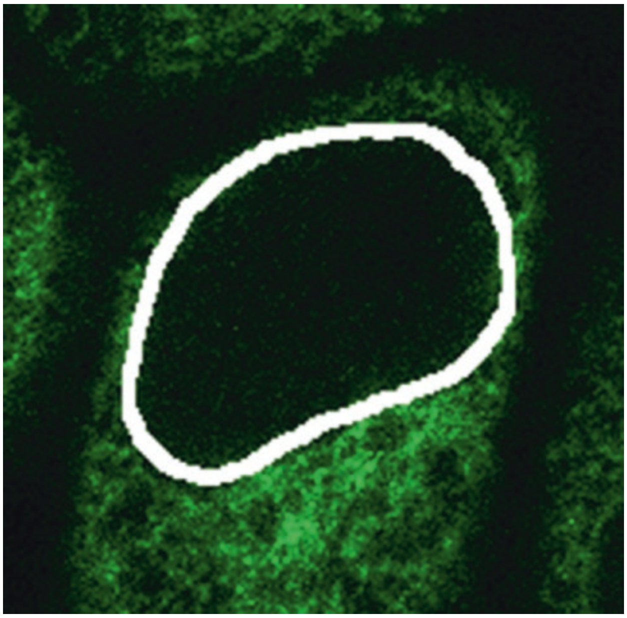
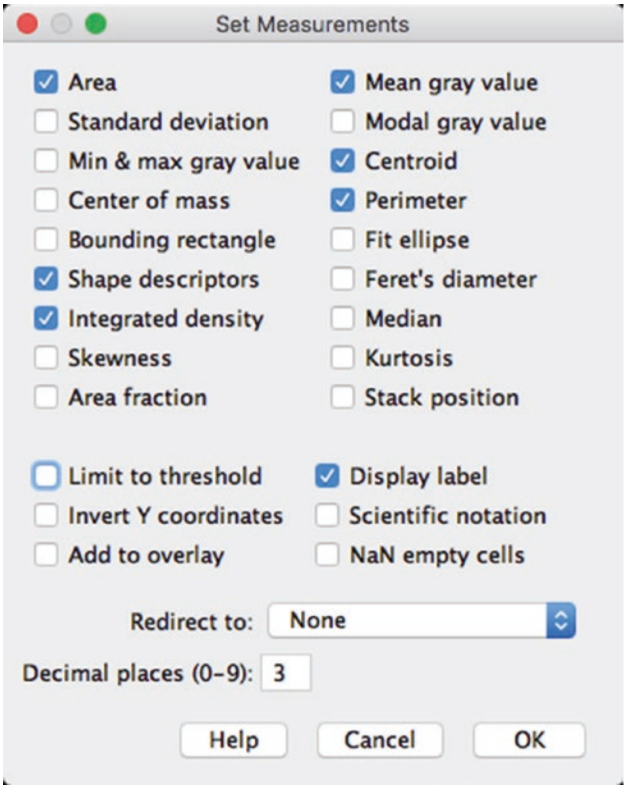


Fig. 2.4 Merged image of the segmented nucleus rim and the lamin receptor channel



[✓ Area]	[✓ Mean gray value]
Standard deviation	Modal gray value
Min & max gray value	[✓ Centroid]
Center of mass	[✓ Perimeter]
Bounding rectangle	Fit ellipse
[✓ Shape descriptors]	Feret's diameter
[✓ Integrated density]	Median
Skewness	Kurtosis
Area fraction	Stack position
Limit to threshold	[✓ Display label]
Invert Y coordinates	Scientific notation
Add to overlay	NaN empty cells

Redirect to: None

Decimal places (0-9): 3

Fig. 2.5 Measurement settings

5. Activate the Lamin receptor image (C2) and then [Edit > Selection > Restore Selection]
6. `[Analyze > Measure]`

Selection of the rim should look like Fig. 2.6.

You will then see results in the Results table such as shown in Fig. 2.7.

When you record these procedures by Command Recorder, the code will look like shown below. Create a new tab in the Script Editor and copy & paste (or it's possible to do the same by clicking "create" button in the Recorder).

```
run("Split Channels");
run("Set Measurements...", "area mean centroid perimeter shape
integrated display redirect=None decimal=3");
selectWindow("C1-Composite");
run("Create Selection");
run("Make Inverse");
selectWindow("C2-Composite");
run("Restore Selection");
run("Measure");
```

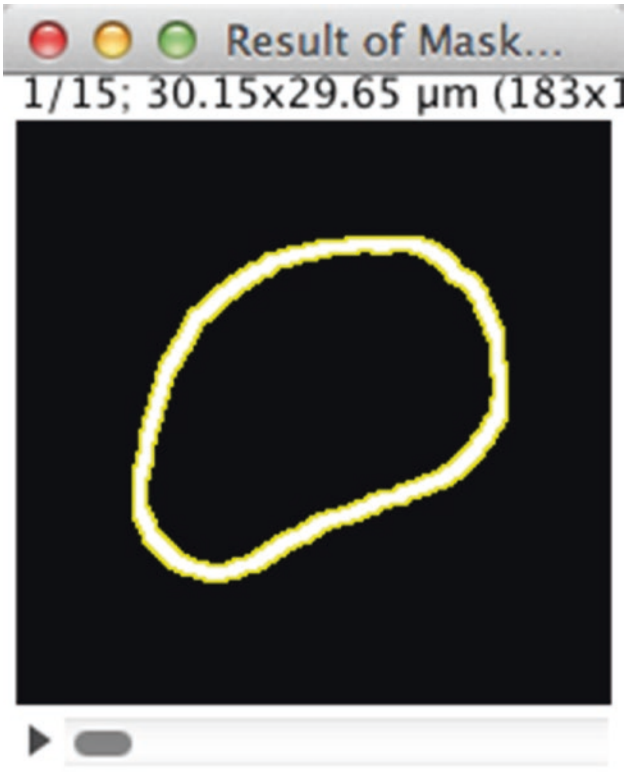
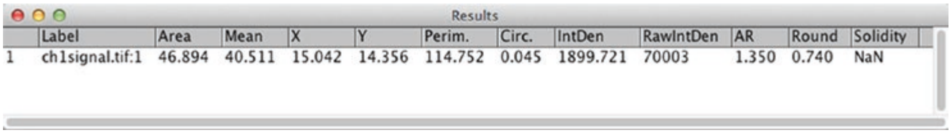


Fig. 2.6 ROI selection of nucleus rim



Results

	Label	Area	Mean	X	Y	Perim.	Circ.	IntDen	RawIntDen	AR	Round	Solidity
1	ch1signal.tif:1	46.894	40.511	15.042	14.356	114.752	0.045	1899.721	70003	1.350	0.740	NaN

Fig. 2.7 Results output

In the 1st line, we split the multichannel stack to do processing individually. In the 3rd and the 6th lines, specific window titles are used. Just like we did in the first block, we need to convert these lines by composing window title of stacks for individual channel by adding prefixes. We also need to acquire their image IDs. For composing window titles, we just need to reuse the code we wrote already in the ► Sect. "Block 1: Splitting Channels".

code/code_block3_measurements.ijm

```
1  orgName = getTitle();
2  run("Split Channels");
3  c1name = "C1-" + orgName;
4  c2name = "C2-" + orgName;
5  selectWindow(c1name);
6  c1id = getImageID();
7  selectWindow(c2name);
8  c2id = getImageID();
```

```
9   opt = "area mean centroid perimeter shape integrated limit dis-
    play redirect=None decimal=3";
10  run("Set Measurements...", opt);
11  selectImage(c1id);
12  run("Create Selection");
13  run("Make Inverse");
14  selectImage(c2id);
15  run("Restore Selection");
16  run("Measure");
```

- Line 1: We first need to capture the title of the multi-channel image.
- Line 2: Then the channels are separated into two stacks.
- Line 3–4: Since we know the rule of how the resulting image stack names are, we construct titles each for channel 1 and channel 2.
- Line 5–8: We then acquire image IDs.
- Line 9–10: To be more explicit, we compose the measurement options as `opt` in line 9 and then use that variable `opt` as an argument for `Set Measurements` in line 10.
- Line 11: Activate nucleus rim image using ImageID captured in line 6, instead of using `selectWindow`.
- Line 12–13: Create nucleus rim ROI (a selection).
- Line 14: Activate Lamin receptor image using ImageID captured in line 8.
- Line 15: Restore the ROI created in line 13.
- Line 16: We measure the region specified by the ROI created above.

Exercise 2.3
Merge the nucleus rim and the Lamin receptor image stacks as described in the beginning of this section and test the code
`code_block3_measurements.ijm` to measure the fluorescence intensity of the nuclear rim.

2.4.2 Integration: The Measurement Over Time

The code above measures only one time point. To measure the intensity changes over time, we need to add **looping** from line 11 to 16 in `code_block3_measurements.ijm` to repeat the measurement over time frames. For this, we need to modify the code by adding a **for-loop**.

code/code_block3_MeasurementOverTime.ijm
```
1  orgName = getTitle();
2  run("Split Channels");
3  c1name = "C1-" + orgName;
4  c2name = "C2-" + orgName;
5  selectWindow(c1name);
6  c1id = getImageID();
7  selectWindow(c2name);
8  c2id = getImageID();
```

```
9   opt = "area mean centroid perimeter shape integrated limit display
    redirect=None decimal=3";
10  run("Set Measurements...", opt);
11  for (i =0; i < nSlices; i++){
12      selectImage(c1id);
13      setSlice(i + 1);
14      run("Create Selection");
15      run("Make Inverse");
16      selectImage(c2id);
17      setSlice(i + 1);
18      run("Restore Selection");
19      run("Measure");
20  }
```

In this updated code, following 4 lines were added for looping through the time lapse frames and measure successively.

- A new line was inserted at line 11 to define the condition of for-looping.
- A new line was inserted at line 13 to activate a specific frame in the nucleus rim stack.
- A new line was inserted at line 17 to activate a specific frame in the stack.
- A curly brace was added at line 20 to close the looping.

Exercise 2.4

Merge rim and Lamin receptor image stacks and test the code `code_block3_MeasurementOverTime.ijm` to see if it measures the intensity of nucleus rim over time frames.

If you see 15 lines of measurement values in the Results window, you are successful.

2.4.3 Integrating Segmentation and Measurements

Finally, we can assemble three blocks of code: the channel splitting block, the segmentation block and the intensity measurement block. As the third block, the intensity measurement block, starts with a two-channel stack (nucleus rim segmentation image and the Lamin receptor signal image), all we need to do is to insert the segmentation block between line 4 and line 5 of block 3 code `code_block3_MeasurementOverTime.ijm`.

Instead of copy and pasting the segmentation block to the measurement block, a better way to do this is to convert the segmentation block to a user-defined function. Like all the macro commands that you see in the Build-in ImageJ macro function reference, we can create our own function by ourselves. We briefly learn how to write a custom function with a simple example.

If we have a code like below:

```
a = 10;
b = 20;
c = a + b + a * b ;
print(c);
```

Evidently, “230” will be printed in the log window. Now, We can convert this formula to a custom function `calc1` that does the calculation in line 3.

```
1  a = 10;
2  b = 20;
3  c = calc1(a, b);
4  print(c);
5
6  function calc1(n, m){
7    return n + m + n * m;
8  }
```

Three lines were added to the original code. Line 6 declares a new user-defined function named “calc1”. It takes two arguments, `n` and `m`. Commands between curly braces is the content of this function, and in this case there is only one line that returns a value. To be more explanatory, this function can be rewritten as follows to do the same thing.

```
function calc1(n, m){
    answer = n + m + n * m;
  return answer;
}
```

Exercise 2.5

1. Modify the code above so that the function `calc1` calculates m to the power of n. Use the build-in command pow(m, n).
2. Change the name of function to `calc2` and run the code. If there is error, fix the code.

In a similar way, we can convert the segmentation block to a single custom function that takes an ImageID as input, does pre-processing, does segmentation, and then returns an ImageID of the segmented image as the output. Here is the code:

code/code_block2_recordNucSegV3_function.ijm
```
function nucseg(orgID){
  //orgID = getImageID();
  selectImage(orgId);
  run("Gaussian Blur...", "sigma=1.50 stack");

  setAutoThreshold("Otsu dark");
  setOption("BlackBackground", true);
  run("Convert to Mask", "method=Otsu background=Dark calculate black");
  run("Analyze Particles...", "size=800-Infinity pixel circularity=0.00-1.00 show=Masks display exclude clear include stack");
  dilateID = getImageID();
  run("Invert LUT");
  options =  "title = dup.tif duplicate range=1-" + nSlices;
13    run("Duplicate...", options);
14    erodeID = getImageID();
15    selectImage(dilateID);
16    run("Options...",  "iterations=2  count=1  black  edm=Overwrite
      do=Nothing");
17    run("Dilate", "stack");
18    selectImage(erodeID);
19    run("Erode", "stack");
20    imageCalculator("Difference create stack", dilateID, erodeID);
21    resultID = getImageID();
22    selectImage(dilateID);
23    close();
24    selectImage(erodeID);
25    close();
26    selectImage(orgID);
27    close();
28    run("Clear Results");
29    return resultID;
30  }
```

Only several lines were added to the original `code_block2_recordNucSegV3.ijm`.

- In line 1, we declare that this is a custom function named `nucseg` that takes a single argument `orgID`. In the original code, `orgID`, which is the imageID of the histone channel image, was captured using `getImageID` command.
- Line 2 is commented out. This is because We do not need to do `getImageID` since the imageID of the histone channel image is provided through the argument of the function.
- Line 3 is inserted, to explicitly activate the image with id `orgId`.
- One line is inserted at line 21, to capture the imageID of the resulting image stack—the mask of nuclear rim—of Image Calculation in line 20. This imageID is named as a variable "resultID" in the function and is returned in the line 29 as the final output of the function.
- `run("Clear Results");` is added at the bottom (line 28) to clear the results table, as we want to have only the results of intensity measurement later.
- In the last line, a curly brace is added to mark the boundary of function.

We can paste this function `nucseg(orgID)` below the intensity measurement macro, and call this function to segment the nucleus rim. In below, I show only the part in the block 3 intensity measurement where function call was added. line 7 to line 8 was inserted to `code_block3_MeasurementOverTime.ijm`.

```
1  orgName = getTitle();
2  run("Split Channels");
3  c1name = "C1-" + orgName;
4  c2name = "C2-" + orgName;
5
6  selectWindow(c1name);
```

```
7   nucorgID = getImageID();
8   nucrimID = nucseg(nucorgID);
9
10  selectWindow(c2name);
11  c2id = getImageID();
12  opt = "area mean centroid perimeter shape integrated display
    redirect=None decimal=3";
13  run("Set Measurements...", opt);
14  for (i =0; i < nSlices; i++){
15      selectImage(nucrimID);
16      setSlice(i + 1);
17      run("Create Selection");
18      run("Make Inverse");
19      selectImage(c2id);
20      setSlice(i + 1);
21      run("Restore Selection");
22      run("Measure");
23  }
```

In line 6 and 7, the image ID of the nucleus (histone) channel is captured. As we do not know if the nucleus channel image stack is the top window, we explicitly call it to the top by `selectWindow`, and then get its ImageID. This ImageID `nucorgID` is then passed to the segmentation function in line 8 (`nucseg( nucorgID)`).

After the image segmentation is done in the function `nucseg`, the ImageID of segmentation result is returned. We capture this ImageID by a variable `nucrimID`. From there, everything is same like we already coded, except that the image selection at line 15 now uses `nucrimID`.

Here is the final code.

code/code_final.ijm

```
1  orgName = getTitle();
2  run("Split Channels");
3  c1name = "C1-" + orgName;
4  c2name = "C2-" + orgName;
5
6  selectWindow(c1name);
7  nucorgID = getImageID();
8  nucrimID = nucseg(nucorgID);
9
10  selectWindow(c2name);
11  c2id = getImageID();
12  opt = "area mean centroid perimeter shape integrated display
    redirect=None decimal=3";
13  run("Set Measurements...", opt);
14  for (i =0; i < nSlices; i++){
15      selectImage(nucrimID);
16      setSlice(i + 1);
17      run("Create Selection");
18      run("Make Inverse");
19      selectImage(c2id);
```

```
    setSlice(i + 1);
    run("Restore Selection");
    run("Measure");
}

function nucseg(orgID){
    selectImage(orgId);
    run("Gaussian Blur...", "sigma=1.50 stack");

    setAutoThreshold("Otsu dark");
    setOption("BlackBackground", true);
    run("Convert to Mask", "method=Otsu background=Dark calcu-
    late black");
    run("Analyze Particles...", "size=800-Infinity pixel circular-
    ity=0.00-1.00 show=Masks display exclude clear include
    stack");
    dilateID = getImageID();
    run("Invert LUT");
    options =  "title = dup.tif duplicate range=1-" + nSlices;
    run("Duplicate...", options);
    erodeID = getImageID();
    selectImage(dilateID);
    run("Options...", "iterations=2 count=1 black edm=Overwrite
    do=Nothing");
    run("Dilate", "stack");
    selectImage(erodeID);
    run("Erode", "stack");
    imageCalculator("Difference create stack", dilateID, erodeID);
    resultID = getImageID();
    selectImage(dilateID);
    close();
    selectImage(erodeID);
    close();
    selectImage(orgID);
    close();
    run("Clear Results");
    return resultID;
}
```

2.5 Results and Conclusion

The final output is a list of nucleus rim intensity values for each time point in Results window. These values can be saved in a CSV file and further analyzed using other software tools more suited for data analysis such as R or Python. Here, to summarize the analysis in this chapter, we plot the changes in the total fluorescence intensity over time using ImageJ Macro `code_plotResults.ijm` (▣ Fig. 2.8). The code appears after the paragraph below.

The plot in ▣ Fig. 2.8 shows an increase in total fluorescence intensity by 1.3-folds in the initial five time points, and then it becomes mostly constant. To know the baseline level intensity more precisely, it might be better to start the imaging and measurement from an earlier time point. In addition, ideally, more measurements could be done with other nuclei to compute an averaged curve for a more reliable results.

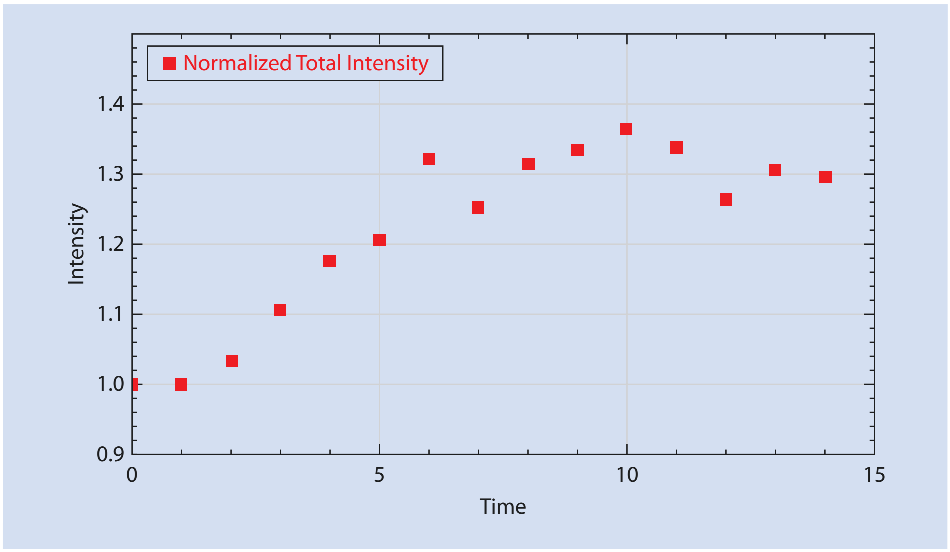


Fig. 2.8 Changes in the total fluorescence intensity over time

Here is the code for this plotting. Explanations follow.

code/code_plotResults.ijm
```
1   // store normalized total intensity values in an array
2  intA = newArray(nResults);
3  for (i = 0; i < nResults; i++)
4    intA[i] = getResult("RawIntDen", i) / getResult("IntDen", 0);
5
6   //prepare x-axis values
7  t = Array.getSequence(intA.length);
8
9   // get the statistics of the total intensity array.
10  Array.getStatistics(intA, amin, amax, amean, astdDev);
11
12   // Create the plot
13  Plot.create("Total Intensity at Nuclear Membrane", "Time", "Inten-
    sity");
14  Plot.setLimits(0, intA.length, amin * 0.9, amax * 1.1);
15  Plot.setColor("red", "red");
16  Plot.setLineWidth(3);
17  Plot.add("circle", t, intA);
18  Plot.setFontSize(14);
19  Plot.addLegend("Normalized Total Intensity");
```

- Line 2 creates a new array for storing intensity measurements listed in the Results table. `nResults` is a build-in function that returns number of rows in the table. This array will be the Y-axis value in the plot.
- Line 3–4 gets the result of non-calibrated integrated density (Column “RawIntDen”) of each row, and divide that value by the integrated density at the time point 0 (the first frame).

- Line 7 creates a new array for X-axis values, where we will store time points. To simplify, we use the frame number as time points, starting from 0.
- Line 10: For fitting the plot in a good range, we first get the minimum and the maximum values of measured intensity. With function `Array.getStatistics`, descriptive statistics values are called back to the provided variables in the argument. In this case, `amin` is the minimum value and `amax` is the maximum value of the array.
- Line 13–19: Plotting commands.
  - Line 13 creates the plot with specified title, X-axis label and Y-axis label.
  - Line 14 sets the range of values to be shown in the plot. Here, the minimum and the maximum value of measurement results are used.
  - Line 15 sets the color of the marker.
  - Line 16 sets the line width of the marker.
  - Line 17 sets the shape of the marker, X-axis values (the array `t`) and Y-axis values (the array `intA`)
  - Line 18 sets the font size of the title and labels
  - Line 19 adds the legend of the plot.

Take Home Message

To measure the changes in the fluorescence intensity over time at the nuclear membrane, we post-processed segmented image of nucleus by mathematical morphology processing "Erosion" and "Dilation" to create a mask for the region-of-interest. In the same way, boundaries of biological structures can be segmented and analyzed.

2.6 Exercise Answers

2.6.1 Exercises 2.1–2.4

In these exercises, one only needs to follow the instructions.

2.6.2 Exercise 2.5

1. Modify the code above so that the function `calc1` calculates m to the power of n. Use the build-in command pow(m, n).

```
a = 10;
b = 20;
c = calc1(a, b);
print(c);

function calc1(n, m){
  return pow(m , n);
}
```

2. Change the name of function to `calc2` and run the code. If there is error, fix the code.
   Answer: Be sure that`calc1` in line 3 needs to be replaced by `calc2` as well.

```
1  a = 10;
2  b = 20;
3  c = calc2(a, b);
4  print(c);
5
6  function calc2(n, m){
7    return pow(m, n);
8  }
```

Acknowledgements This workflow was initially developed together with Andreas Boni for teaching in a practical course at EMBL Heidelberg. We are grateful to his contributions. The same topic was taught in many courses during last five years and we thank all the feed-backs we received from those teaching sessions from participants. We thank Andreas Boni, Nathalie Daigle, and Jan Ellenberg for providing the sample image data. We thank Christian Tischer (EMBL Heidelberg) for reviewing this chapter.

Bibliography

Boni A, Politi AZ, Strnad P, Xiang W, Hossain MJ, Ellenberg J (2015) Live imaging and modeling of inner nuclear membrane targeting reveals its molecular requirements in mammalian cells. J Cell Biol 209(5):705–720. ISSN: 0021-9525. https://doi.org/10.1083/jcb.201409133. http://www.jcb.org/lookup/doi/10.1083/jcb.201409133