MTQuant Documentation

# Contents



# 1 `MTQuant` Overview

This document details the usage of the MATLAB function `MTQuant`. `MTQuant` rapidly quantifies neuronal microtubule organization by extracting three parameters from fluorescence images of neuronal microtubules. Collectively referred to as the "organization parameters," the three parameters are:

- **Spacing** ($\bar{S}$): average number of microns between microtubule ends
- **Coverage** ($\bar{C}$): average number of microtubules in bundle cross-section
- **Length** ($\bar{L}$): average length of microtubules in a single neuron

The `MTQuant` code allows users to perform four tasks:

- **Task 1:** Generate Maximum Projections
- **Task 2:** Trace and Generate Neurite Masks
- **Task 3:** Generate Intensity Line Scans
- **Task 4:** Calculate Organization Parameters

Throughout this document, we assume a basic working knowledge of MATLAB: strings, cell arrays of strings, comments, using the command line, etc. Typically, a user will begin with either image stacks or maximum projection images, and then complete Tasks 2-4. Task 2, Trace Neurite Masks, is the only step that *requires* user input. The remainder of this document will describe the four tasks in greater detail and provide some usage examples. The specifcs of the input arguments, output files, and naming conventions are included as well.

### 1.1 Citation

If you use this code in your research, please cite it as:

> R. Cooper, S. Yogev, K. Shen, and M. Horowitz, *MTQuant: "Seeing" Beyond the Diffraction Limit in Fluorescence Images to Quantify Neuronal Microtubule Organization*, (in preparation, 2016).

## 2 `MTQuant` Functionality

This section will briefly describe the inputs and outputs of each of these steps, and also provide detailed instructions on the user input required in Step 2. There are a few interspersed examples of command line execution of `MTQuant` in MATLAB, which are collected and augmented in Section 3.2. By default, MTQuant completes all four of the tasks mentioned above.

`MTQuant` always requires two inputs: a list of directories in which to find the images to process, and the name of a CSV file to store the data. The program searches the directories in the list, as well as all subdirectories, for the files to analyze. ***The list of directories must be a cell array of strings***. Furthermore, the output filename must ***not*** have an extension. For example:

```
>> dirList = {'C:/path/to/images/'; './'};
>> fileOut = 'testOutput';
>> MTQuant(dirList,fileOut);
```

To change the default values of various variables, `MTQuant` uses MATLAB's name-value pair format. Examples, once again, are in Section 3.2, and a full list of the input and output arguments is in Section 4.1.

## 2.1 Task 1: Generate Maximum Projections

When generating maximum projections, by default `MTQuant` searches for green and red image stacks that end in '`*_w1488-single.TIF`' and '`*_w2561-single.TIF`,' respectively. Both of these strings can be changed with an input argument name-value pair. The program will simply ignore processing of the red channel if there are no red stacks present. Thus far, `MTQuant` has been tested with 16-bit .TIF images. The output projections are written with the same absolute path as the stack images, but with '`*_w1488-single.TIF`' and '`*_w2561-single.TIF`' replaced by '`*_g_proj.tif`' and '`*_r_proj.tif`', respectively. Naming conventions are described in more detail in Section 5. `MTQuant` also writes a CSV file which contains the stack slice from which each pixel was selected in a file saved with the suffix '`*_stackInds.csv`'.

## 2.2 Task 2: Trace and Generate Neurite Masks

The program requires some user input in order determine which portions of the neuron are of interest to the researcher. Figure 1 shows example images of the necessary human input. For each image ending in '`*_g_proj.tif`,' the image is displayed to the user (Figure 1(a)), and the user is asked whether or not they would like to trace a neurite. When the user selects yes, they are prompted to click along the neuron. The user needs to click points along the section of the neuron that they are interested in, beginning and ending with the endpoints of that section. For example, to trace part of the axon in, the user needs to click on the points labeled in Figure 1(b).

Then the user is prompted to click on the end of the trace that corresponds to the minus-ends of the MTs, denoted by the green star in Figure 1(c).

Finally, the user needs to click on a box inside the animal to capture the background intensity so that it can be subtracted later from the line scans. See Figure 1(d).

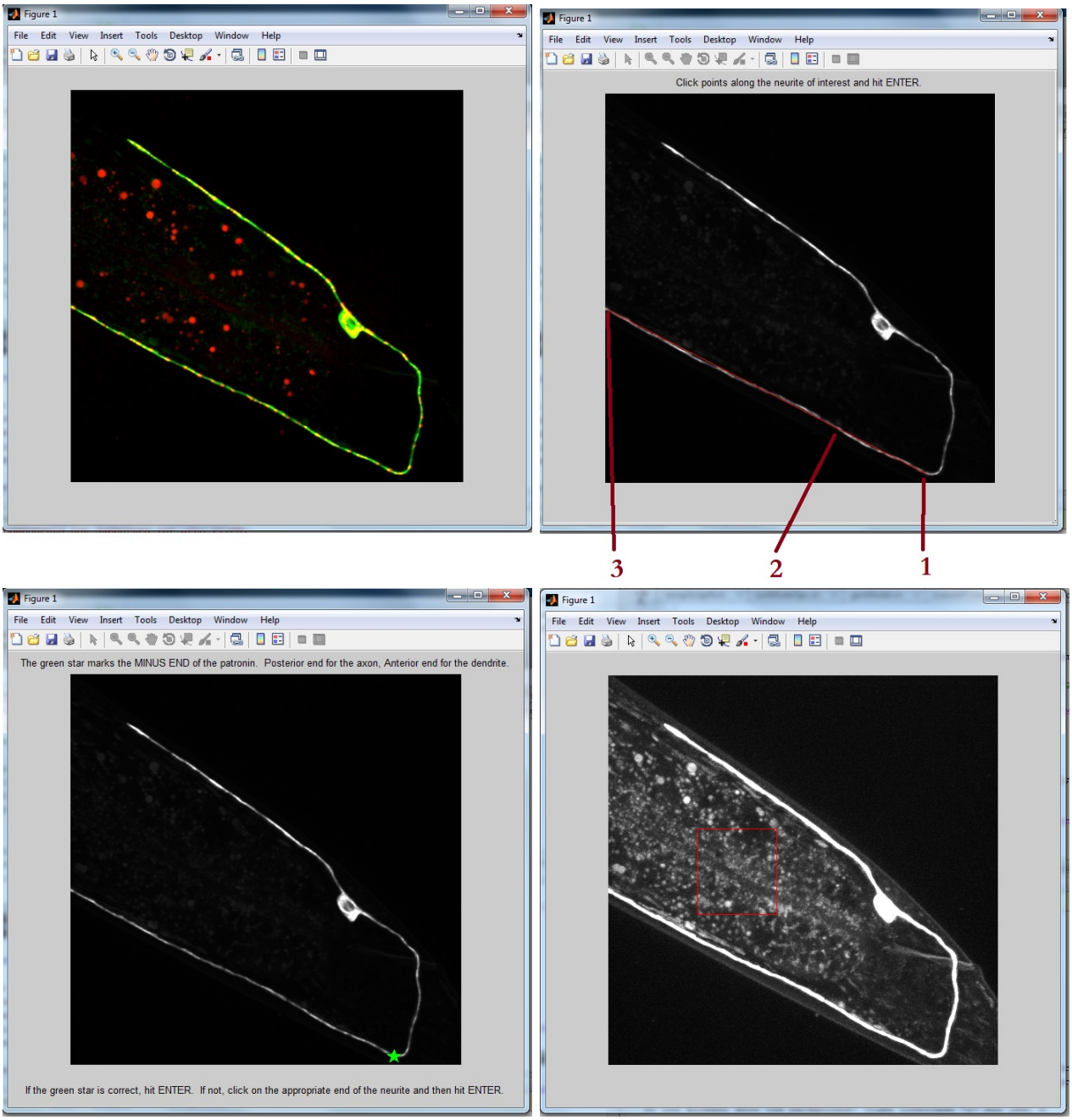


Figure 1: Clock-wise from top left: (a) Multi-color Image. (b) Tracing the Axon Mask: The numbers 1, 2, and 3 correspond to the locations clicked by the user to generate the red line tracing the axon. (c) Select the orientation of the MT minus-ends. (d) Select the background box.

```
>> Do you want to trace an axon?  (lowercase "n" for no) y
>> Are you happy with this trace?  (lowercase "n" for no) y
>> The green star marks the MINUS END of the patronin.  Posterior end
   for the axon, Anterior end for the dendrite.
>> If the green star is correct, hit ENTER. If not, click on the
   appropriate end of the neurite and then hit ENTER.
>> Select the green background.
>> Do you already know the background?  (type lowercase "y" for yes) n
>> Please click on two opposite corners of the box you want to average
   for the background
>> Are you happy with the location of this rectangle?  (type lowercase
   "y" for yes) y
>> background = 2146.8372
>> Select the red background.
>> background = 1984.9885
```

Figure 2: Tracing Neurite Code Snippet

At each step along the way, the program asks the user to confirm whether they are satisfied with their selection. If the user wants to change the trace, they can simply type 'n' at those prompts and re-select the trace or the background box. An example command line interaction is shown in Figure 2.2.

This function writes out two files: one is an image file (.TIF) of the mask, and another is a .CSV file which stores the mask as well as the background and endpoint information. These files end in '`*_Mask.tif`' and '`*_Mask.csv`,' respectively. To allow for separate labeling of various sections, the user can use the '`indivFileOut`' input argument to prepend a label to the mask files, such as '`axon`,' yielding, for example, '`_axonMask.csv`.'

## 2.3 Task 3: Generate Intensity Line Scans

`MTQuant` generates line scans and saves them in files with suffix '`*_LineScans.csv`.' The first column of this file is the green line scan, with the first value corresponding to the location of the green star in Figure 1(c), or the minus-ends of the MTs. The second column is the red line scan, oriented the same way.

The program also saves the pixel locations that were summed to generate the line scans in the file '`*_LineScanInds.csv`.' As with the saved mask files, the '`indivFileOut`' input argument allows the user to prepend a label to these suffixes. The format of this file is as

follows: If the line scan has length $d$, Its dimensions are $d \times 402$. The first 201 columns refer to the $y$-coordinate of each point, and the second set of 201 columns refers to the $x$-coordinate. Hence, the starting $(x, y)$ location of the spline is in $(1, 302)$ and $(1, 101)$ in this CSV file.

### 2.4 Task 4: Calculate Organization Parameters

The final task of `MTQuant` is to calculate the organization parameters of the animals. If the flag `toRateImgs` is set to true (it is false by default), the program stops after each image to ask the user for a rating: 2 for a good image, 1 for a barely acceptable image, and 0 for a bad image. Then, it generates two CSV files for each of the following groupings: all good images (group 2), all acceptable images (groups 1 and 2), and all images (groups 0, 1, 2).

The first CSV file, with the name of the input argument `fileOut`, contains an entry for every image analyzed. The list of entries in the table is summarized in Section 4.2.

The second file is has the suffix '`*_grouped.csv`.' The program assumes that, if the input directory list is only one element long, that each subfolder of that directory is a separate group of worms. `MTQuant` assigns each group a number, which is included in `fileOut`, and used for both the distribution refinement. The '`*_grouped.csv`' file calculates the mean and standard deviation for each parameter in Section 4.2 for each separate group of worms.

## 3 Running `MTQuant`

### 3.1 Setup & System Requirements

`MTQuant` and its supporting code can be downloaded at:

`http://roscoope.github.io/MTQuant`

The code can be downloaded as a ZIP file or checked out as a Github repository. The code can be run from the top-level `MTQuant` directory, or elsewhere by typing the following commands into MATLAB to add the appropriate directories to the path:

```
>> addpath('C:/path/to/MTQuant')
>> addpath('C:/path/to/MTQuant/em')
>> addpath('C:/path/to/MTQuant/misc')
>> addpath('C:/path/to/MTQuant/neuronTracing')
```

```
>> addpath('C:/path/to/MTQuant/redScan')
>> addpath('C:/path/to/MTQuant/singleMT')
```

Thus far, `MTQuant` has been tested on Windows 7 and 8, using MATLAB versions R2014b and R2013b.

## 3.2 Example Usage

**Setup**

```
>> addpath('C:/path/to/MTQuant')
>> addpath('C:/path/to/MTQuant/em')
>> addpath('C:/path/to/MTQuant/misc')
>> addpath('C:/path/to/MTQuant/neuronTracing')
>> addpath('C:/path/to/MTQuant/redScan')
>> addpath('C:/path/to/MTQuant/singleMT')
```

**Run `MTQuant` with Default Settings**

```
>> dirList = {'C:/path/to/images/', './'};
>> fileOut = 'testOutput';
>> MTQuant(dirList,fileOut);
```

**Run `MTQuant` with Various Name-Value Input Argument Pairs**

```
>> dirList = {'C:/path/to/images/', './'};
>> fileOut = 'testOutput';
>> taskList = 14; % we have already made the projections
>> toRateImgs = 1; % we suspect many images are bad, let's rate them
>> MTQuant(dirList,fileOut,'taskList',taskList,'toRateImgs',toRateImgs);
```

# 4 Input and Output Arguments

## 4.1 Full List of Input Arguments

| Input Argument | Description | Default Value |
|---|---|---|
| Requred Arguments | | |
| `dirList` | List of directories in which to find the images to process The program searches the directories in `dirList`, as well as all subdirectories, for the files to analyze. ***Must be a cell array of strings.*** | |
| `fileOut` | Name of the file to store the data. Must ***not*** have an extension. | |
| General Arguments | | |
| `taskList` | Tasks `MTQuant` will complete. Add the values of the various tasks to complete in order to perform more than one task at a time.<br>1. 1 to make stacks into projs<br>2. 2 to make masks of projs<br>3. 4 to generate line scans<br>4. 8 to analyze line scans | 15 |
| `verbose` | How often to display which file is being processed<br>2 = every new file<br>1 = every 50 files<br>0 = never | 1 |
| `indivFileOut` | String to prepend to the outputs of `MTQuant`, such as 'axon' | '' |
| Arguments for Task 1: Generate Maximum Projections | | |
| `gFileExt` | Ending of green stack files to search for | `'*_w1488-single.TIF'` |
| `rFileExt` | Ending of red stack files to search for | `*'_w2561-single.TIF'` |
| Arguments for Task 2: Trace Neurite Masks | | |
| `diPix` | Number of pixels by which to dilate the user-selected line when creating the neurite mask | 30 |
| Arguments for Task 3: Generate Line Scans | | |
| `toAvg` | 0 = sum together intensities along perpendiculars to the neuron's spline<br>1 = average together intensities instead (NOT recommended) | 0 |
| Arguments for Task 4: Calculate Org Params (MT Minus-end Detection) | | |
| `toUseManualRPeaks` | 0 = detect MT minus-ends automatically<br>1 = User is asked to enter number of MT minus-ends for each image | 0 |

| Input Argument | Description | Default Value |
|---|---|---|
| `toUseRandRPeaks` | 0 = No correction to count single red puncta as multiple MT minus-ends<br>1 = Randomly count some single red puncta as multiple MT minus-ends | 1 |
| `rPeakTol` | Tolerance of peak selection. This value is multiplied by the maximum value of the red line scan. For a peak to be considered real (not noise), the difference between the peak and the nearest valley must be at least `rPeakTol` × `max(rLineScan)`. A higher value for `rPeakTol` means fewer peaks are identified. | 0.05 |
| `rPeakCorr` | Weight on Bernouli random variable that decided if a single red dot is 1 or 2 MT minus-ends. As coverage increases, this value should increase as well (up to 20 if necessary) | 0.1 |
| Arguments for Task 4: Calculate Org Params (Single MT Brightness Extraction) | | |
| `toPreproc` | 0 = Use the green line scan directly<br>1 = Smooth and sharpen the green line scan before extracting single MT brightness | 0 |
| `toUseBlurCorr` | 0 = Don't correct for overlapping MT starts and ends (not recommended)<br>1 = Correct for overlapping MT starts and ends | 1 |
| `roundLevel` | Rounding level for quantized signal. A fractional number of MTs below `roundLevel` is rounded down when forming the quantized signal, while a fractional number above `roundLevel` is rounded up. | 0.5 |
| `distThresh` | When calculating single MT brightness, this value is the number of allowable pixels between a step up in the quantized signal and a MT minus-end before the MT minus-end is classified as a mislocalization | 3 |
| `alpha1` | Weight on Mean Squared Error Cost | $\frac{0.26}{\texttt{mean(gLineScan)}}$ |
| `alpha2` | Weight on Cost of mislocalized steps up | 0.0013 |
| `alpha3` | Weight on Error in number of steps up | 1 |
| Arguments for Task 4: Calculate Org Params (Miscellaneous) | | |

| Input Argument | Description | Default Value |
|---|---|---|
| `toRateImgs` | 0 = do not rate images<br>1 = rate images<br>After each image's line scan has been extracted, the user is asked to rate the image:<br>2 = good image<br>1 = ok image<br>0 = bad image<br>Then the program outputs a set of CSV files for each of the following combinations of images: good images only (2), good and ok images (1 & 2), and all images (0, 1, 2)<br>If `toRateImgs` is false, then all images are given the rating of 2. | 0 |
| `toUseHalfMTs` | How to handle the number of MTs at the beginning and end of the signal when calculating length. Makes very inconsequential changes; can be ignored.<br>0 = no correction (Use the number of minus-ends detected, called `numMTs`<br>1 = Use `numMTs` - (value of quantized signal at end of line scan) + (average value of quantized signal at beginning and end of line scan)<br>2 = Use `numMTs` + (value of quantized signal at end of line scan) | 0 |
| `toEM` | 0 = do not implement distribution refinement<br>1 = implement distribution refinement | 1 |

Table 1: Input Arguments to MTQuant

## 4.2 Output Table Column Names

| # | Output Argument | Description | Units/ Possible Values |
|---|---|---|---|
| Image Information | | | |
| 1 | `Directory` | Directory containing the processed image | String |
| 2 | `DataFile` | Name of the line scan file that was processed to generate this row of data | String |

| # | Output Argument | Description | Units/ Possible Values |
|---|---|---|---|
| 3 | DirNum | The group number of the line scan file. If the dirList sent to MTQuant had one entry, then each subfolder inside of dirList is considered a separate group, and receives a distinct DirNum. Otherwise, each directory in dirList is considered a distinct group. | Integer |
| 4 | Rating | If the argument toRateImgs is false (as it is by default), this column is simply 2 for every image. However, if toRateImgs is true, then this column contains the user's rating:<br>2 = good image<br>1 = ok image<br>0 = bad image | 0, 1, or 2 |
| 5 | Max_G | Maximum intensity value of the green line scan | Intensity (AU) |
| 6 | Avg_G | Average intensity value of the green line scan | Intensity (AU) |
| 7 | Scan_Length | Length of the green line scan | Pixels |
| 8 | Num_MTs | Number of MT minus-ends detected in the red line scan | Integer |
| Organization Parameters | | | |
| 9 | Avg_Spacing | Average Spacing of MT minus-ends in the line scan | Pixels |
| 10 | Std_Dev_Spacing | Standard Deviation of Spacing of MT minus-ends in the line scan | Pixels |
| 11 | Single_MT_Brightness | Extracted Single MT Brightness | Intensity (AU) |
| 12 | Avg_Coverage | Average Coverage (number of MTs in neuron cross-section) | Number of MTs |
| 13 | Std_Dev_Coverage | Standard Deviation of Coverage | Number of MTs |
| 14 | Avg_Length | Average MT Length | Pixels |

Table 2: Outputs of MTQuant

In the '*_grouped.csv' files, the data in each column of the output data has been consolidated by the group numbers (DirNum). For each group identified as described in Section 5.1, the mean and standard deviation are calculated for each of the parameters in Table 2. The names of the columns are the same in the '*_grouped.csv' files as for the non-grouped

data, but a 'M_' or 'S_' is prepended to the name in the '*_grouped.csv' files, representing the mean and standard deviation, respectively.

# 5 Naming Conventions

## 5.1 How Image Directory Organization Affects the Output DirNum

Every analyzed line scan is given a group number (described in DirNum in Table 2). DirNum is assigned as described below. First, consider the following directories:

- C:/path/to/data
- C:/path/to/data/wild_type
- C:/path/to/data/mutant1
- C:/path/to/data/mutant2

Assume that C:/path/to/data contains the three subdirectories wild_type, mutant1, and mutant2, and no other images. The three subdirectories can contain images, subdirectories with more images, or a combination of both. Depending on the input dirList to MTQuant, the assignment of the output DirNum varies.

If dirList is just a single directory (e.g., dirList = {'C:/path/to/data'};), then all subdirectories contained in dirList are considered a separate group. Hence, images in the wild_type subdirectory (and its subdirectories) would be in group 1, mutant1 images in group 2, and mutant2 images in group 3. The output DirNum is 1 for the images in group 1 (wild_type), 2 for images in group 2 (mutant1), and 3 for images in group 3 (mutant2).

If dirList is a list of directories (e.g., dirList = {'C:/path/to/data/mutant2'; 'C:/path/to/data/mutant1'};), then each directory in the list is considered a separate group, and numbered based on the order it appears in dirList. Therefore, images in the mutant2 subdirectory (and it's subdirectories) would be in group 1 and mutant1 images in group 2. Again, DirNum is 1 for the images in group 1 (mutant2) and 2 for images in group 2 (mutant1).

## 5.2 Input and Output Filenames for Each Task

| File Suffixes | Description | Output of Task... | Input to Task... |
|---|---|---|---|
| '*_w1488-single.TIF' gFileExt | Green stack file | - | 1 |
| '*_w1561-single.TIF' rFileExt | Red stack file | - | 1 |
| '*_g_proj.tif' | Green maximum projection file | 1 | 2,3 |
| '*_r_proj.tif' | Red maximum projection file | 1 | 2,3 |
| '*_stackInds.csv' | Stack slice containing green maximum projection values | 1 | - |
| '*_Mask.csv' [indivFileOut,'*_Mask.csv'] | Mask info CSV file | 2 | 3 |
| '*_Mask.tif' [indivFileOut,'*_Mask.tif'] | Mask image file | 2 | 3 |
| '*_LineScans.csv' [indivFileOut,'*_LineScans.csv'] | Line scans | 3 | 4 |
| '*_LineScanInds.csv' [indivFileOut,'*_LineScanInds.csv'] | Pixel locations that were summed to generate the line scans | 3 | 4 |
| [fileOut,'_2.csv'] | All data for images classified as "good" (all images by default) | 4 | - |
| [fileOut,'_2_grouped.csv'] | Grouped data for images classified as "good" (all images by default) | 4 | - |
| [fileOut,'_12.csv'] | All data for images classified as "good" or "ok" (Only if toRateImgs is 1) | 4 | - |
| [fileOut,'_12_grouped.csv'] | Grouped data for images classified as "good" or "ok" (Only if toRateImgs is 1) | 4 | - |
| [fileOut,'_012.csv'] | All data for all images (Only if toRateImgs is 1) | 4 | - |
| [fileOut,'_012_grouped.csv'] | Grouped data for all images (Only if toRateImgs is 1) | 4 | - |

Table 3: File Naming Convention. *=wild card