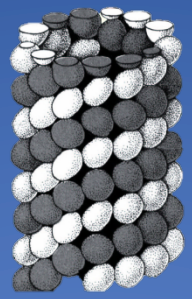


Using STADIA to quantify dynamic instability in microtubules

Shant Mahserejian, Ph.D.
Pacific Northwest National Laboratory
Holly Goodson, Ph.D.
University of Notre Dame

Tutorial by Michael Sinclair, M.A., *NBCT*
Kalamazoo Area Mathematics and Science Center

Last updated on 10/12/2021

STADIA:

The Statistical Tool for Automated Dynamic Instability Analysis — STADIA — is a MATLAB™-based program designed to quantify and characterize dynamic instability behavior in microtubules in a more objective, precise, and reproducible way than is presently possible.
This tutorial will guide you through the process of using STADIA.

Three key issues:

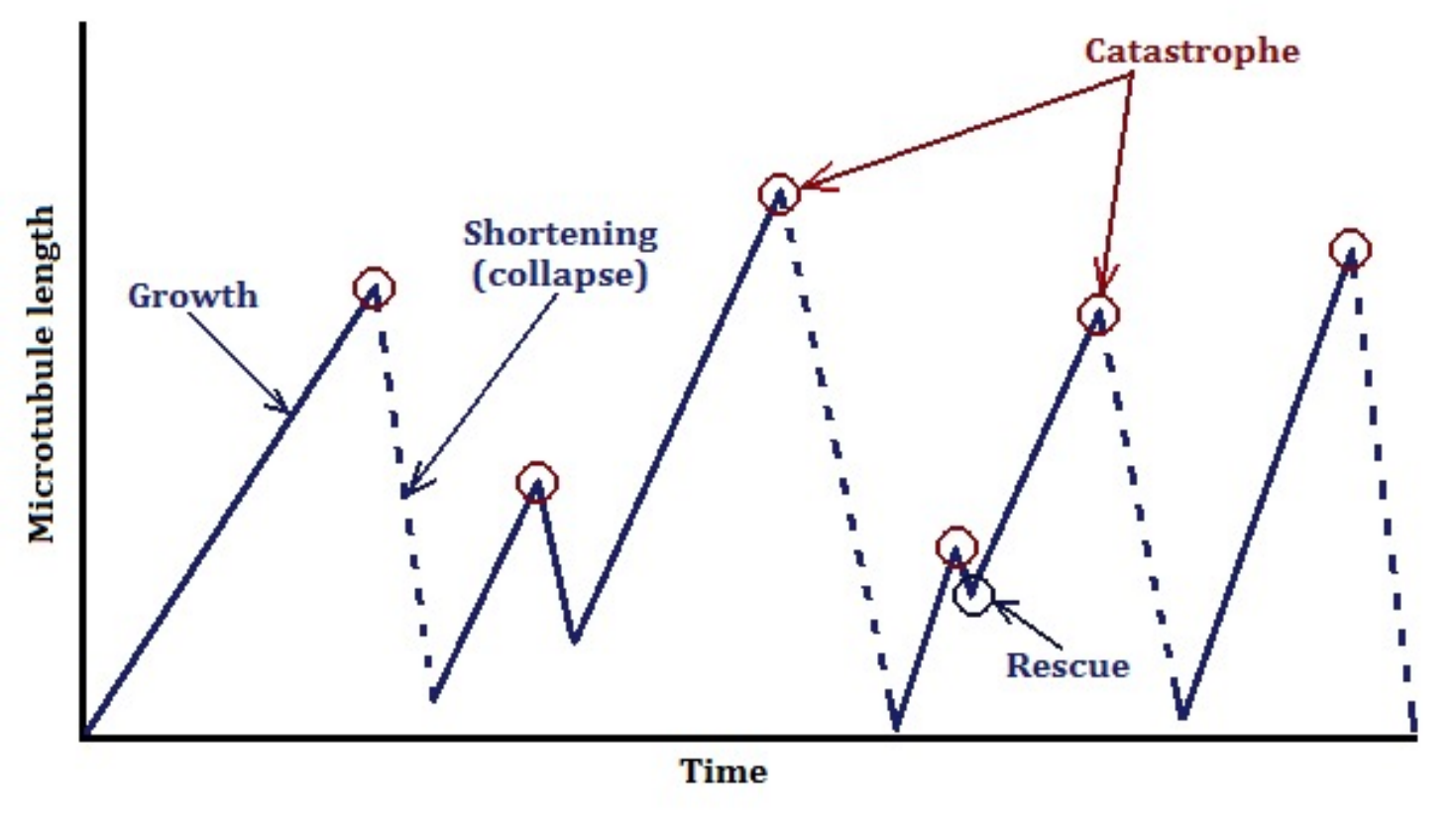
1. This program is not a Matlab™ tutorial; you are expected to have a working knowledge of Matlab™ in order to run it.
2. Once STADIA is running, do not use the computer; please allow the program to run to completion.
3. Due to differences in operating systems, MacOS and Windows will have slightly different layouts.

Objectives:

This new dynamic instability (DI) measurement method is designed to:

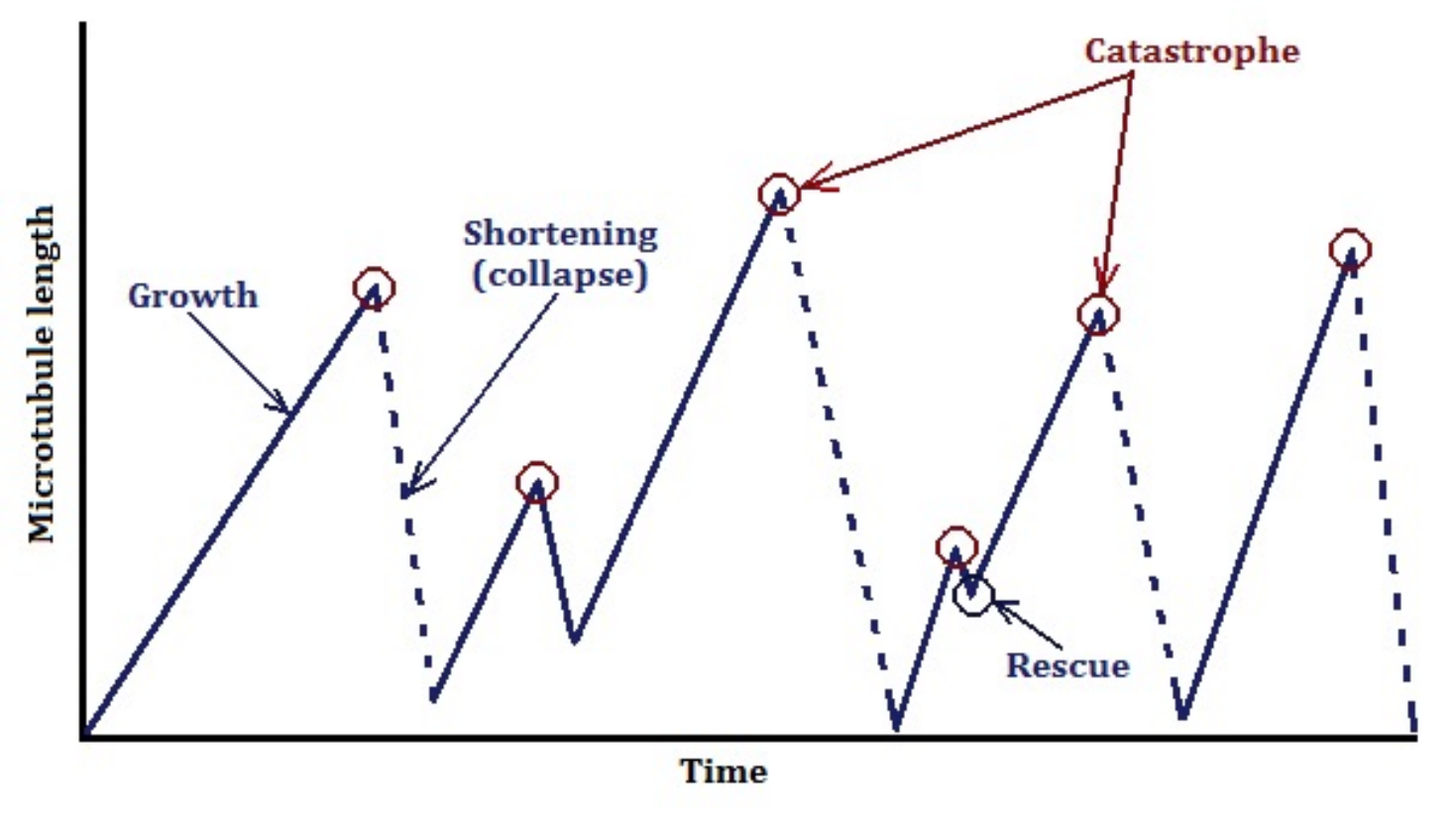
1. Pinpoint exact moments of any dynamic change.
2. Look for any intermediate dynamics beyond growth and shortening.
3. Calculate dynamic instability metrics more accurately.

“Standard” microtubule behavior.



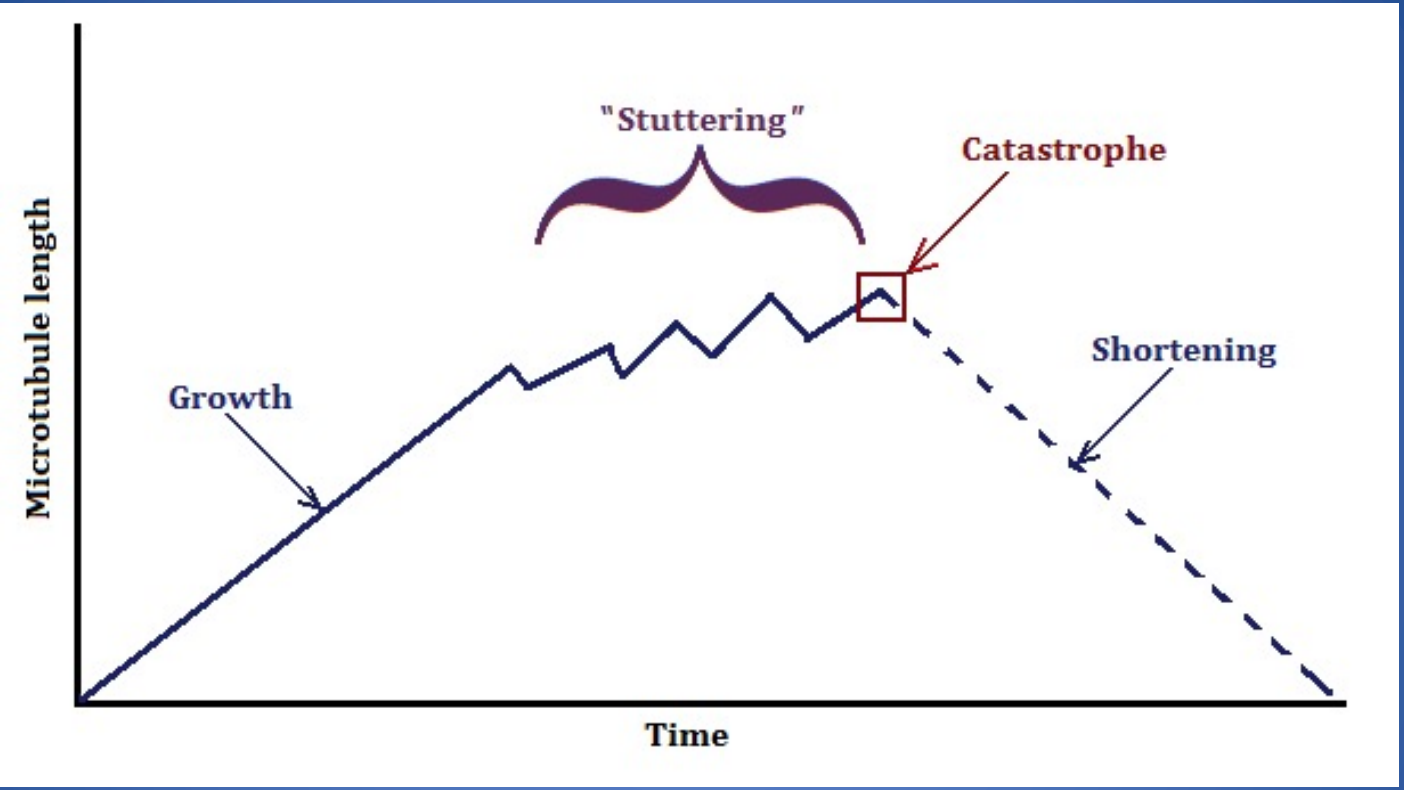
Traditional models of microtubule growth and shortening follow dynamic instability, with growth driven by the concentration of tubulin and “death” induced by a catastrophic drop-off leading to microtubule shortening caused by GTP to GDP hydrolysis before polymerization begins again.

“Standard” microtubule behavior.



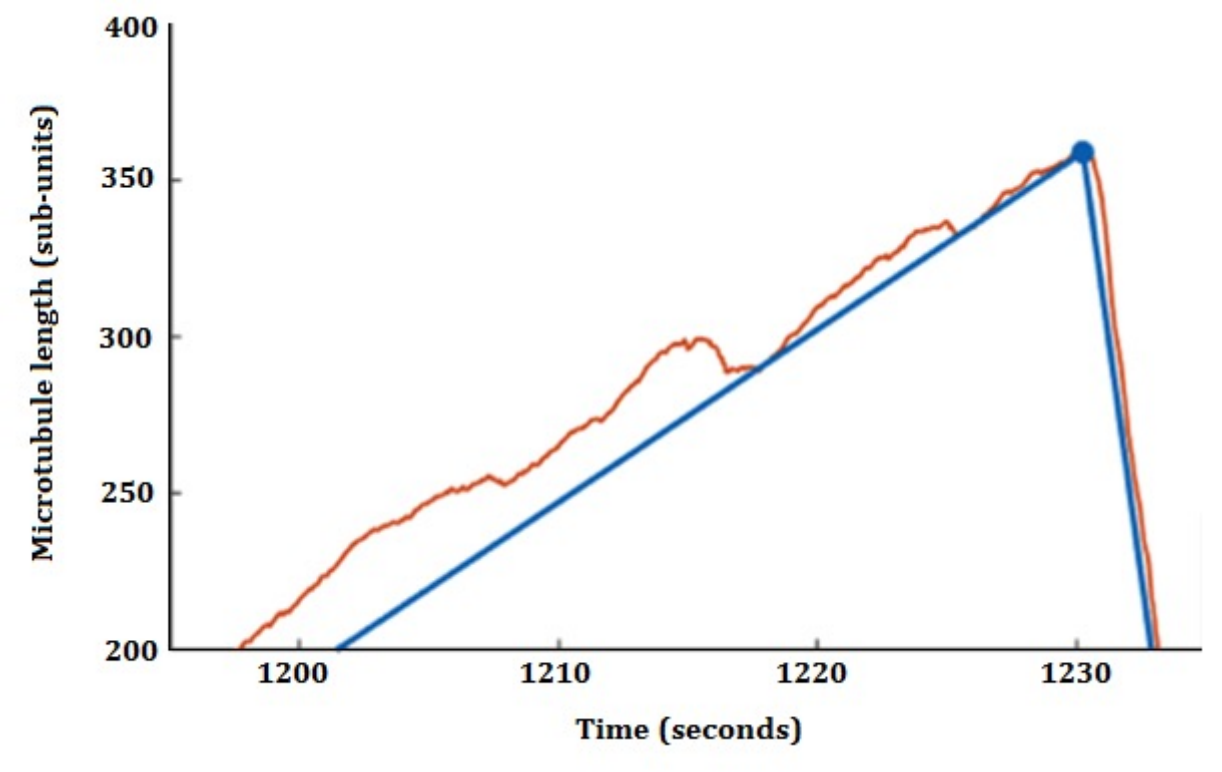
In other words, catastrophe is the change from growth to shortening while rescue is the change from shortening to growth. It is also important to note that rescues are rare, and the mechanism driving them is not yet understood.

“Stuttering”.



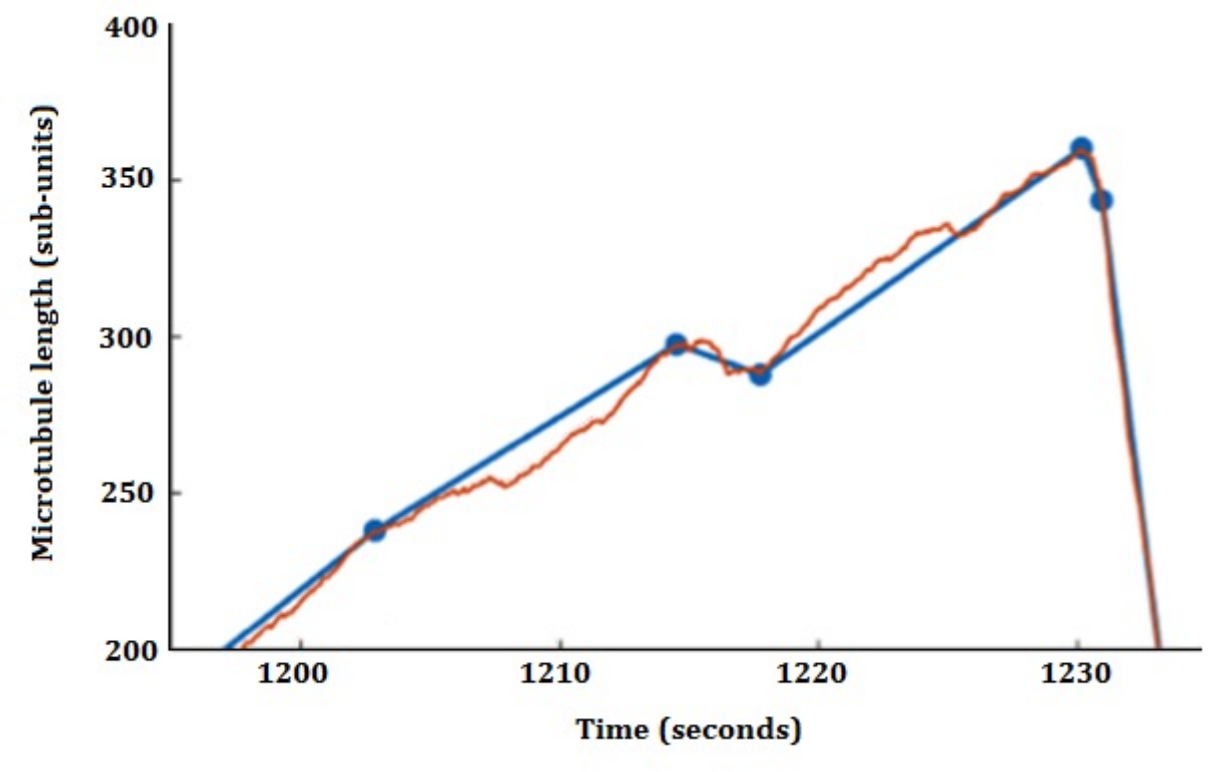
So, the fundamental question is: Where does growth end and shortening begin? The STADIA program — when applied to experimental and simulated data — shows compelling evidence that, just prior to catastrophe, microtubules can exhibit stuttering, where there appears to be another phase within dynamic instability.

Old vs. new approach.



The standard approach — when analyzing data collected with high acquisition rates (up to 1000 events per second) — is to assume that only strictly growth and shortening periods exist when approximating the behavior. But it is clear that more inconspicuous behavior can be missed with this process.

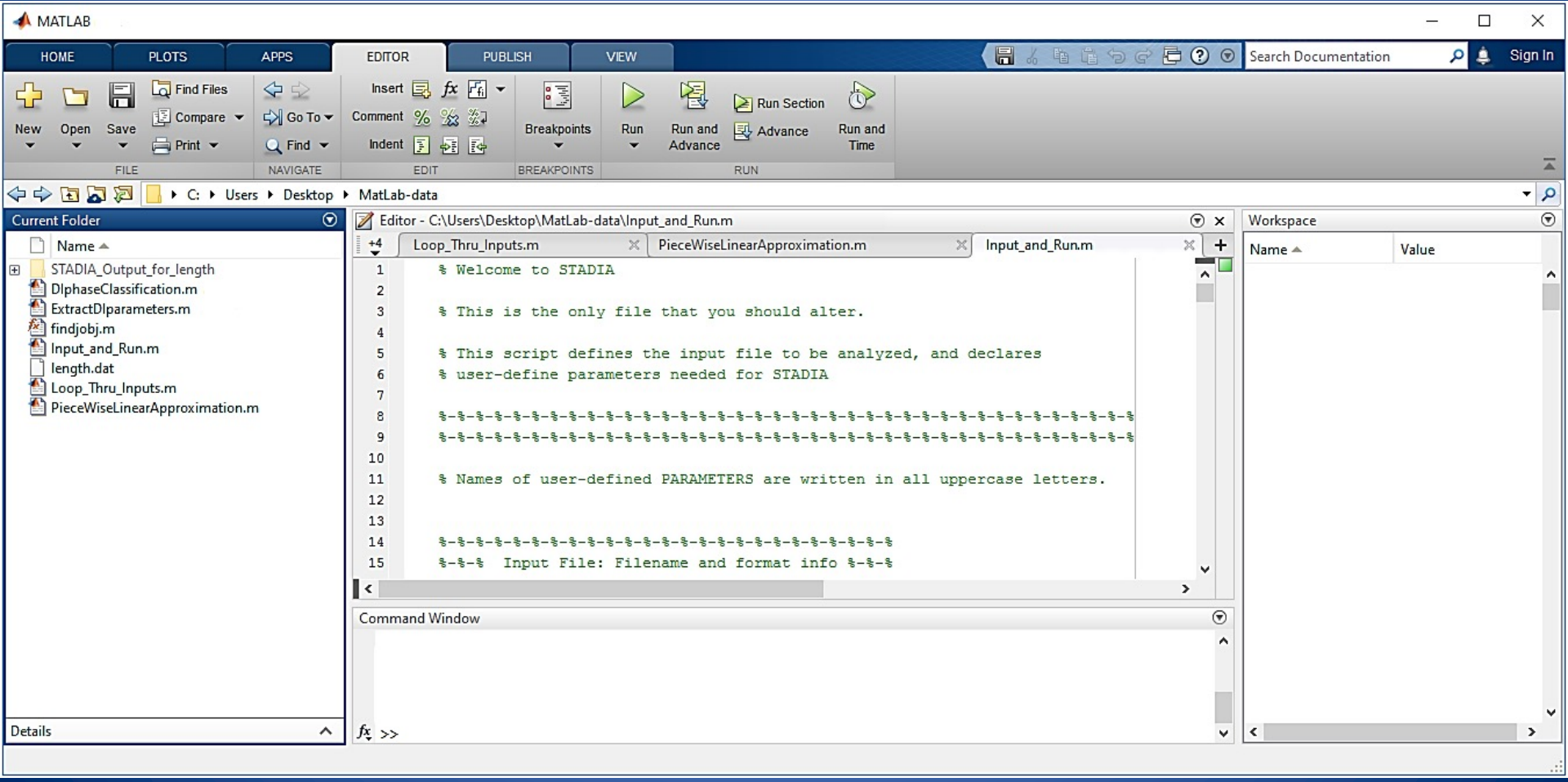
Old vs. new approach.



The approach employed here uses more precise piece-wise linear approximations in order to better study rapid, low-amplitude fluctuations. It can also reveal more subtle effects not previously seen.

Getting started.

This is the window (with **`Input_and_Run.m`** as the start folder) you see when you open up the MatLab™ STADIA program with a PC:

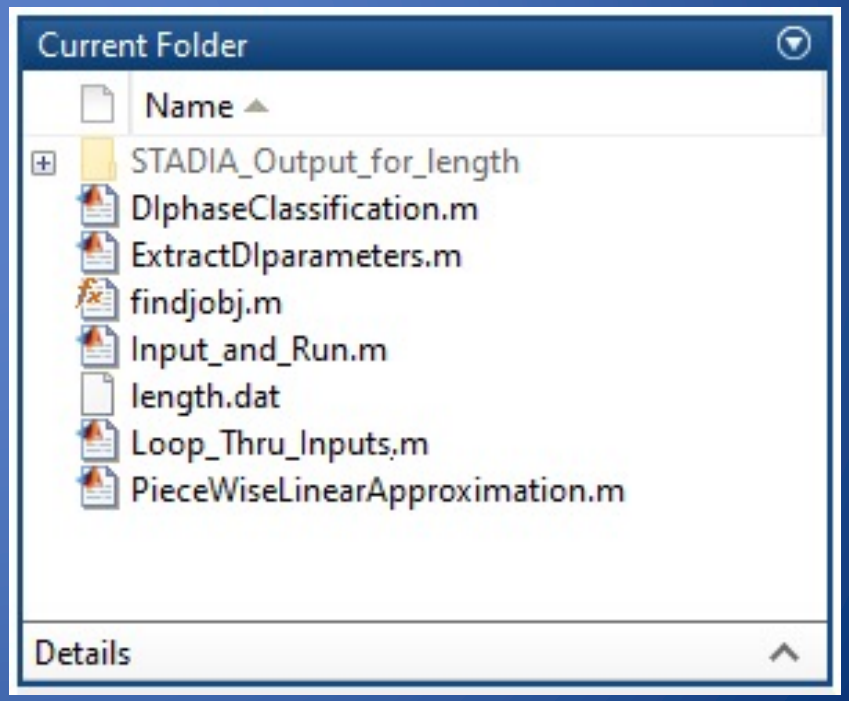


Getting started.

First, you will need seven (7) files in the folder you are going to work with;

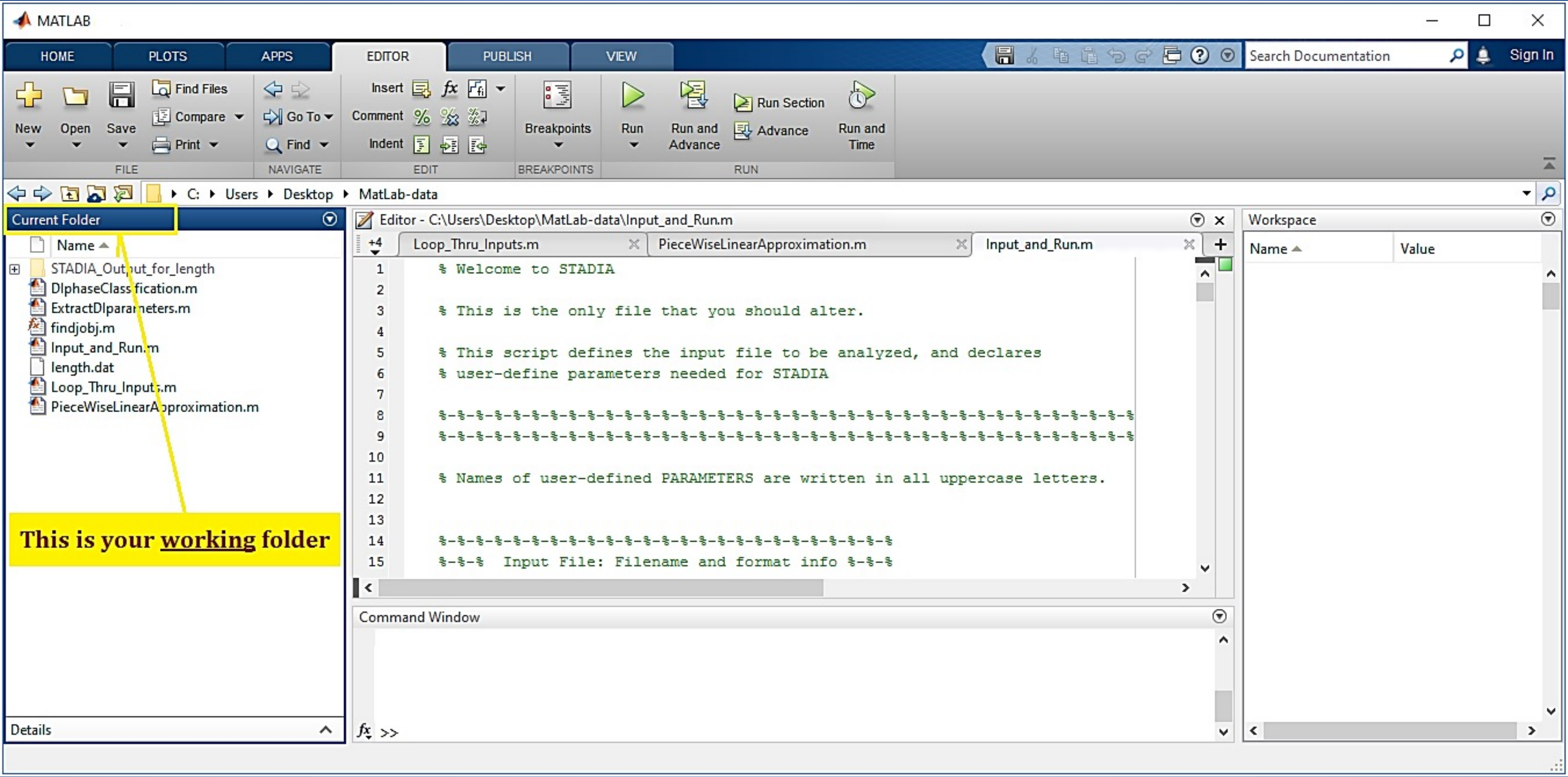
1. **`Input_and_Run.m`**,
2. **`Loop_Thru_Inputs.m`**,
3. **`PieceWiseLinearApproximation.m`**,
4. **`DIphaseClassification.m`**,
5. **`ExtractDIparameters.m`**, and
6. **`findjobj.m`**.

You will also need a **length history input file**(s) (in .dat, .txt, or .csv form). It is recommended that this file be in the same directory as the STADIA files listed above. This is shown at right.



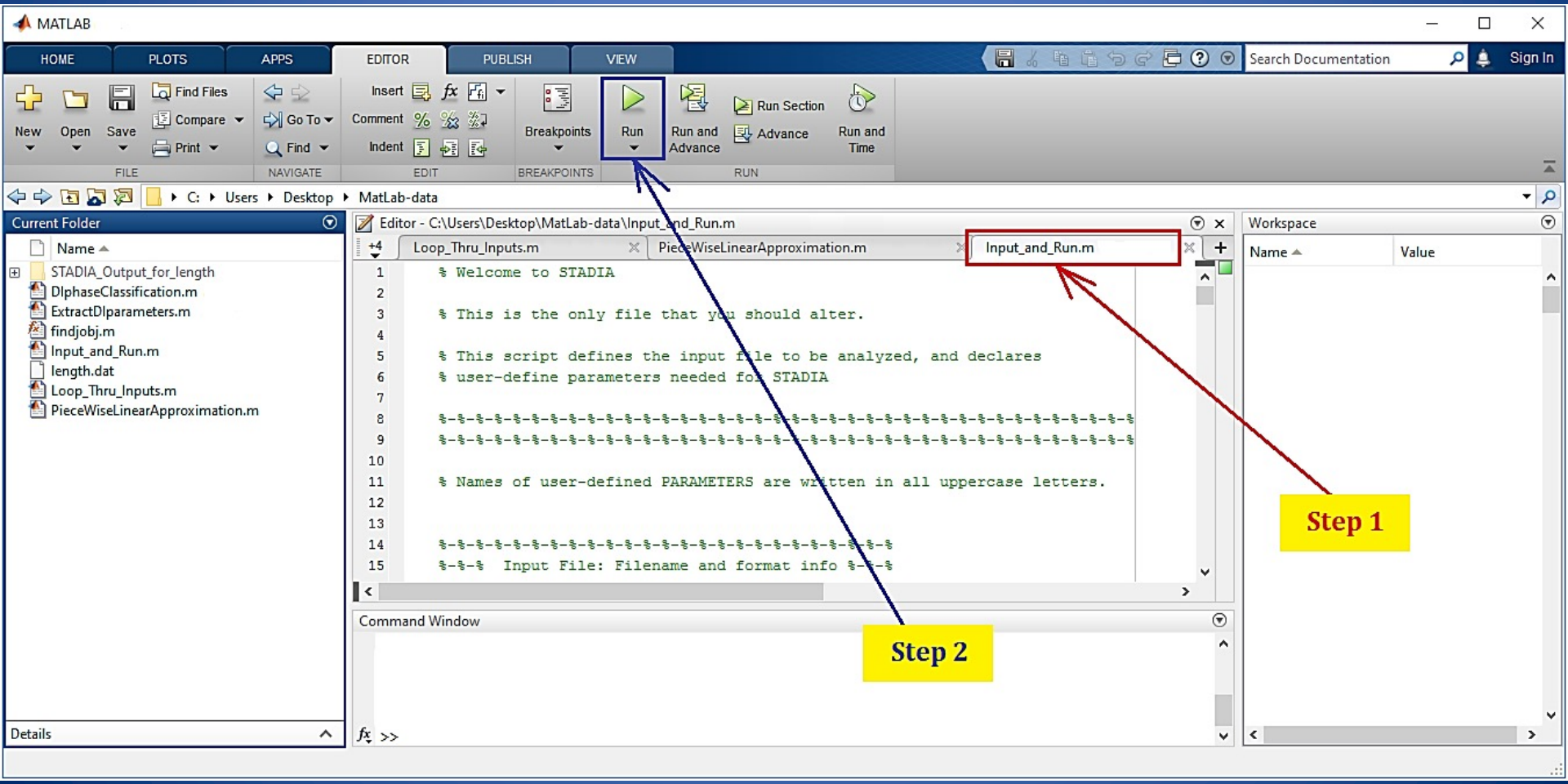
Getting started.

Find the Current Folder on the MatLab™ open page; be sure those six files are in the folder along with your length data — in .dat, .txt , or .csv — files (shown for a PC). You should have a total of seven (7) files.



Getting started.

Click on the **`Input_and_Run.m`** folder (Step 1), then simply click on Run (Step 2) to operate the program for the chosen method parameters:



Program stages.

The MatLab™ STADIA program runs through three (3) steps or stages with an “initialization” phase:

Step	Process
0	Initialization; file name, reading the file, and defining method parameters.
1	Segmentation of microtubule length history.
2	Classification of microtubule stages.
3	Analysis of the data.

Step 0: Initialization.

Step 0: Initialization.

```
SKIP_FILE_READ = 0;
```

If this file is set at 0, it erases the previous run(s) (i.e., “flushes” STADIA) and starts the program over from scratch. Otherwise, if you are running STADIA on a large data file repeatedly, set = 1 to help saves time. The default setting for this is 0.

Step 0: Initialization.

`FILE_NAME_INPUT = {'test_data.dat'};`.
This is the name of the length history input file or files. You can name it with whatever system you prefer, but just be sure to identify each file name in single quotes (`' '`) and the list of file names must be in a set of brackets (`{ }`).
Remember, the file must end with either a .dat, .txt, or .csv suffix.
Typical file names look like these:

`{'LengthHistory_Joe_10uM_1hr_070917_run1.dat'}`
`{'Length_Ann_run24_12uM_20min_071817.txt'}`.

Step 1: Segmentation.

Step 1: Segmentation.

`FIRST_DATA_ROW = 2;`
This annotates which row of the input file indicating the beginning of the data content that you wish to analyze.
`MT_LENGTH_COLUMN_INDICES = 2;`
This variable allows you to select which set or sets of input files (your data) that you wish to examine. If multiple columns of the file contain microtubule lengths, then the length history plots that are created will be "stitched together" with only the first column of data corresponding to the actual time values in the original file; the rest will be shifted to further time steps. In addition, you can select specific subsets to run through the program. For example, if

```
MT_LENGTH_COLUMN_INDICES = 2:128;
```

then the program will use data in all columns from 2 through 128. If

```
MT_LENGTH_COLUMN_INDICES = [2:5, 8, 11:18];
```

then the program will use only data in columns 2 through 5, column 8, and columns 11 through 18.

Step 1: Segmentation.

`TIME_COLUMN = 1;`

This places the time values in the first column; do not use more than one column for time.

`TIME_CONVERSION_FACTOR = 1;`

If you're using step numbers instead of time, then you will need to convert the steps to seconds; simply list the frame rate of your input data. If time is in minutes or hours, convert to seconds by multiplying by 60 and 3600, respectively. If the time values in the data are in seconds, then the default value = 1.

For example, if your input data is in minutes, then

`TIME_CONVERSION_FACTOR = 60;`

Step 1: Segmentation.

```
INPUT_FILE_DELIMITER = ' ';
```

This tells the program how you've separated data in your input file. The example above is when each data point is simply separated by a space (1 1 2 3 5 8 13…).

On the other hand, if the data are listed using commas (such as = 1, 1, 2, 3, 5, 8, 13, …), then

```
INPUT_FILE_DELIMITER = ',';
```

needs a comma between the single quotes so that the program looks for it in order to identify each data point.

If you use a semi-colon (1; 1; 2; 3; 5; 8; 13; …), then

```
INPUT_FILE_DELIMITER = ';';
```

If the data are separated by tabs, then

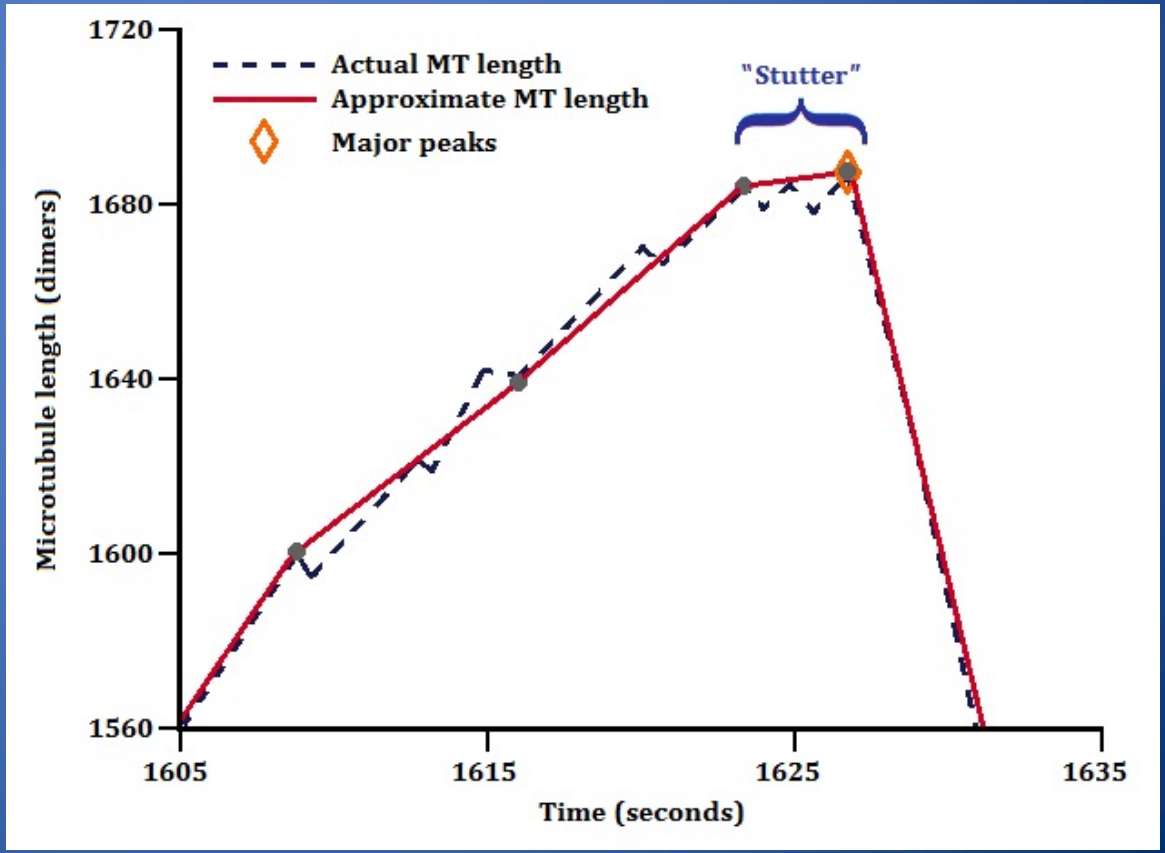
```
INPUT_FILE_DELIMITER = '\t';
```

Step 1: Segmentation.

```
MIN_SEGMENT_DURATION_INPUT = 0.1;
MAX_ERROR_TOLERANCE_INPUT = 20;
```

These are the two critical parameters in the linear approximation phase of the program. You are attempting to "balance" the minimum time used to differentiate segment vertices vs. the margin of error.

If the Minimum Segment Duration and/or Maximum Error Tolerance is **reasonably ideal** (the best fit), then the resulting graph will **be accurate enough** to see stutters if they exist.

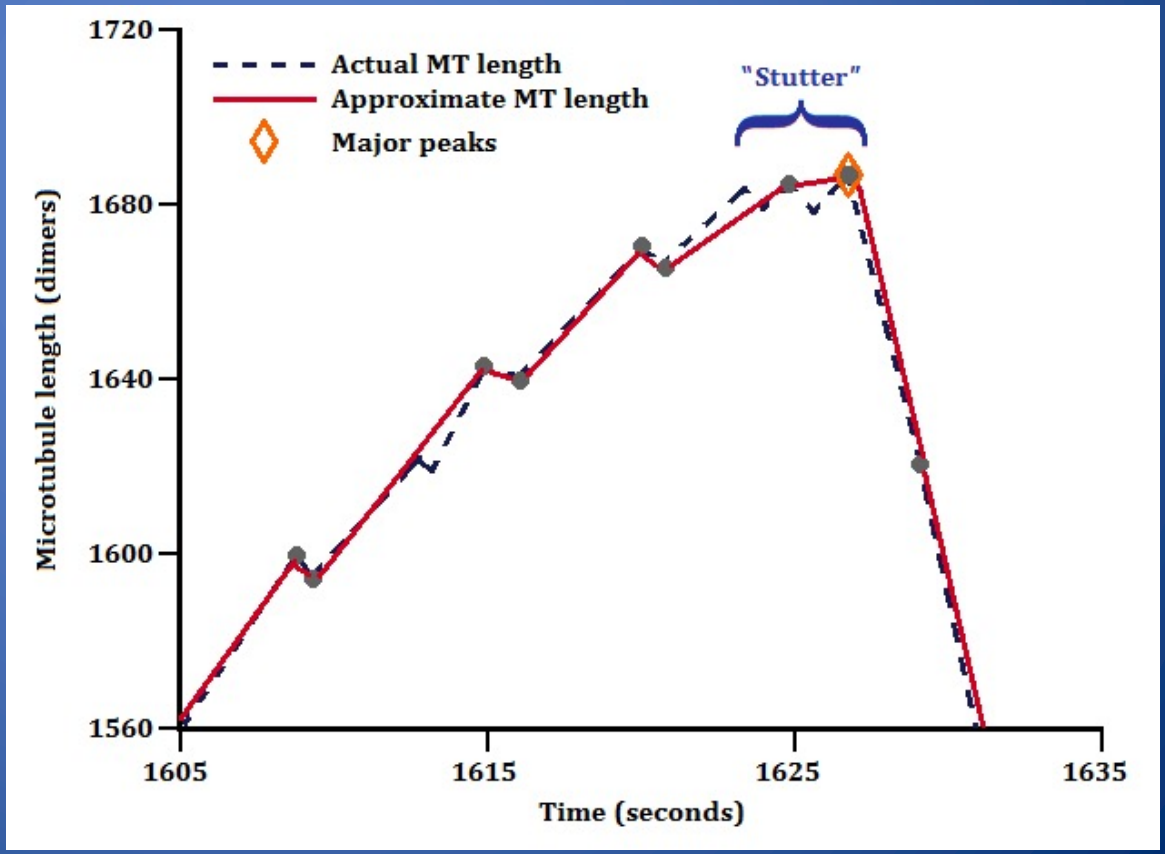


Step 1: Segmentation.

For example, suppose you set the following:

```
MIN_SEGMENT_DURATION_INPUT = 0.1;
MAX_ERROR_TOLERANCE_INPUT = 1;
```

In this case, you've set the Minimum Segment Duration and Maximum Error Tolerance as **too small**, with the resulting graph being "**too accurate**" and effectively looking like the original data.

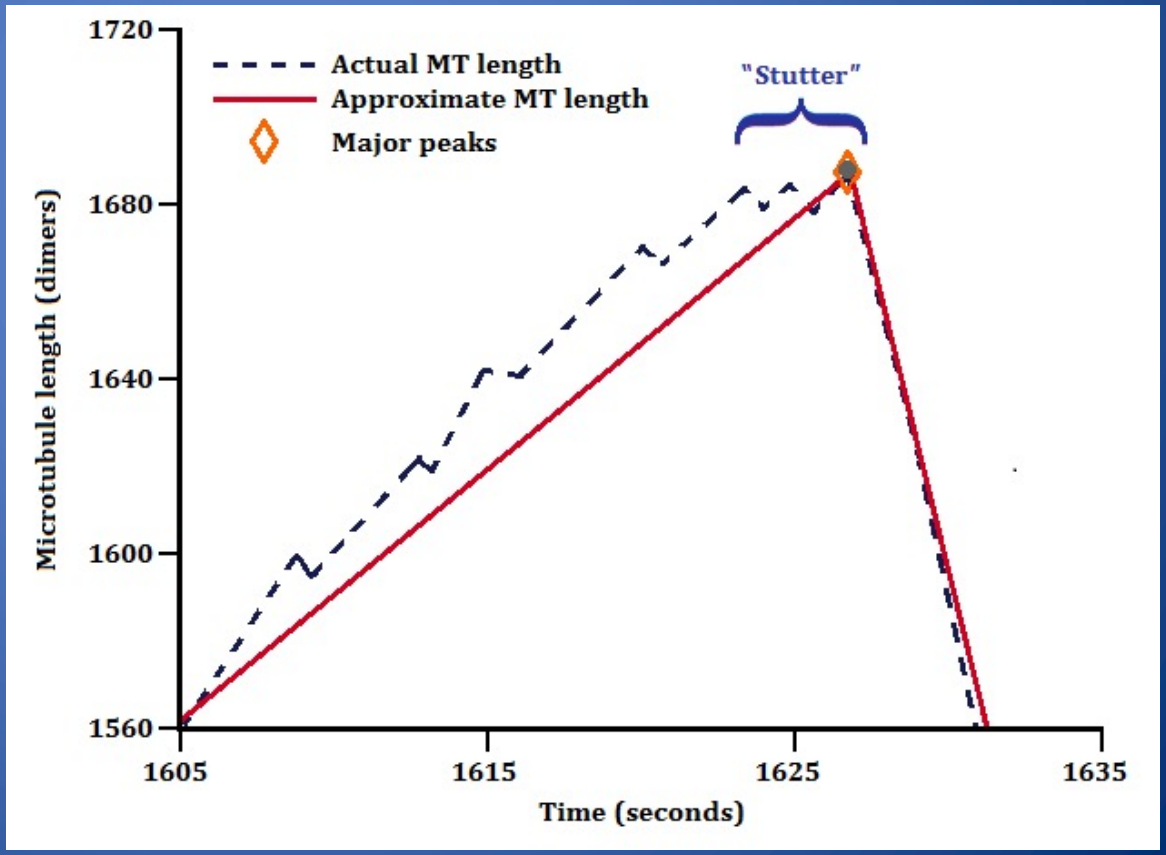


Step 1: Segmentation.

On the other hand, suppose you use these values:

```
MIN_SEGMENT_DURATION_INPUT = 1;
MAX_ERROR_TOLERANCE_INPUT = 30;
```

Now the Minimum Segment Duration and Maximum Error Tolerance are **too large**, with the resulting graph **not be accurate enough** to see subtle changes like stuttering.



Step 1: Segmentation.

Note that it is not always possible to satisfy both the Minimum Segment Duration and the Maximum Error Tolerance. In cases where these parameters are in conflict, STADIA will first attempt to find a neighboring point by ‘nudging’ the vertex in the direction that would increase the segment duration (thus trying to find a point that satisfies the Maximum Error Tolerance without violating the Minimum Segment Duration). If attempts to reconcile the conflict with nudging fails five times, STADIA produces ‘irreconcilable errors.’ Warnings to the user are outputted in the command window for both vertex nudges and irreconcilable errors.

Step 2: Classification.

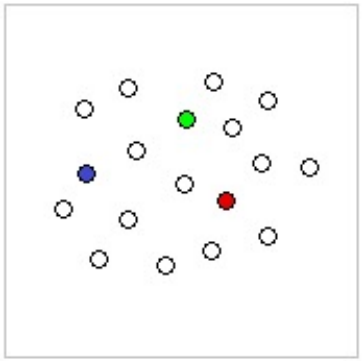
Step 2: Classification.

```
FLATSTUT_MAX_SEGMENTHEIGHT_THRESHOLD_INPUT = 3;
FLATSTUT_MAX_SEGMENTSLOPE_THRESHOLD_INPUT = 0.5;
NUC_HEIGHT_THRESHOLD_INPUT = 75;
```

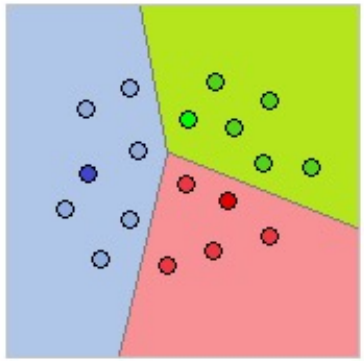
These parameters enable you to classify the behavior of growth and shortening.

Step 2: Classification.

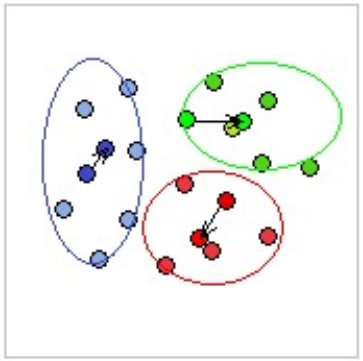
STADIA uses *K*-means clustering to classify the piecewise linear segments into phase classes. Generically, *K*-means clustering is an algorithm designed to partition *n* data points into *k* clusters. It does this by assigning each data point to the cluster with the "nearest" mean. The next part of the program organizes the data into clusters that can be identified as growth, stutter, or shortening.



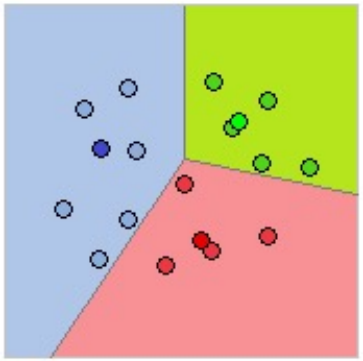
1. *K* number of *means* (*k*=3 as shown) are randomly generated in the set.



2. *K* clusters are then created by linking every value with the nearest *mean*.



3. The centroid of each of the *K* clusters now becomes the new *mean*.

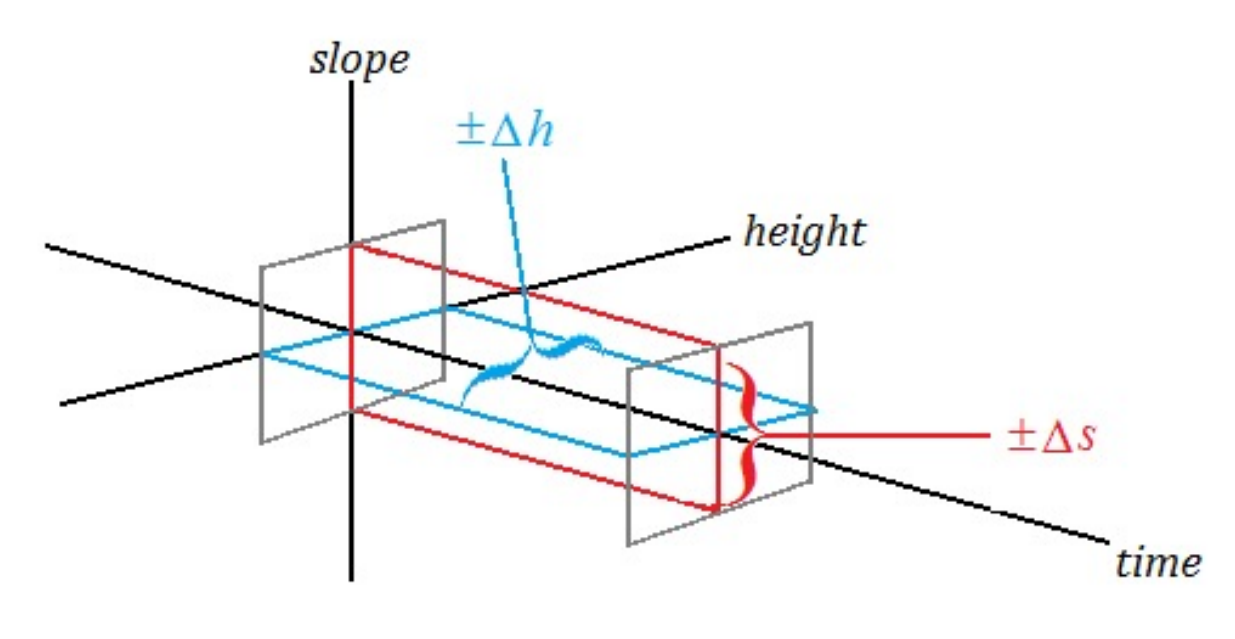


4. Steps 2 and 3 are repeated until a clear convergence has been reached.

Step 2: Classification.

```
FLATSTUT_MAX_SEGMENTHEIGHT_THRESHOLD_INPUT = 3;
FLATSTUT_MAX_SEGMENTSLOPE_THRESHOLD_INPUT = 0.5;
```

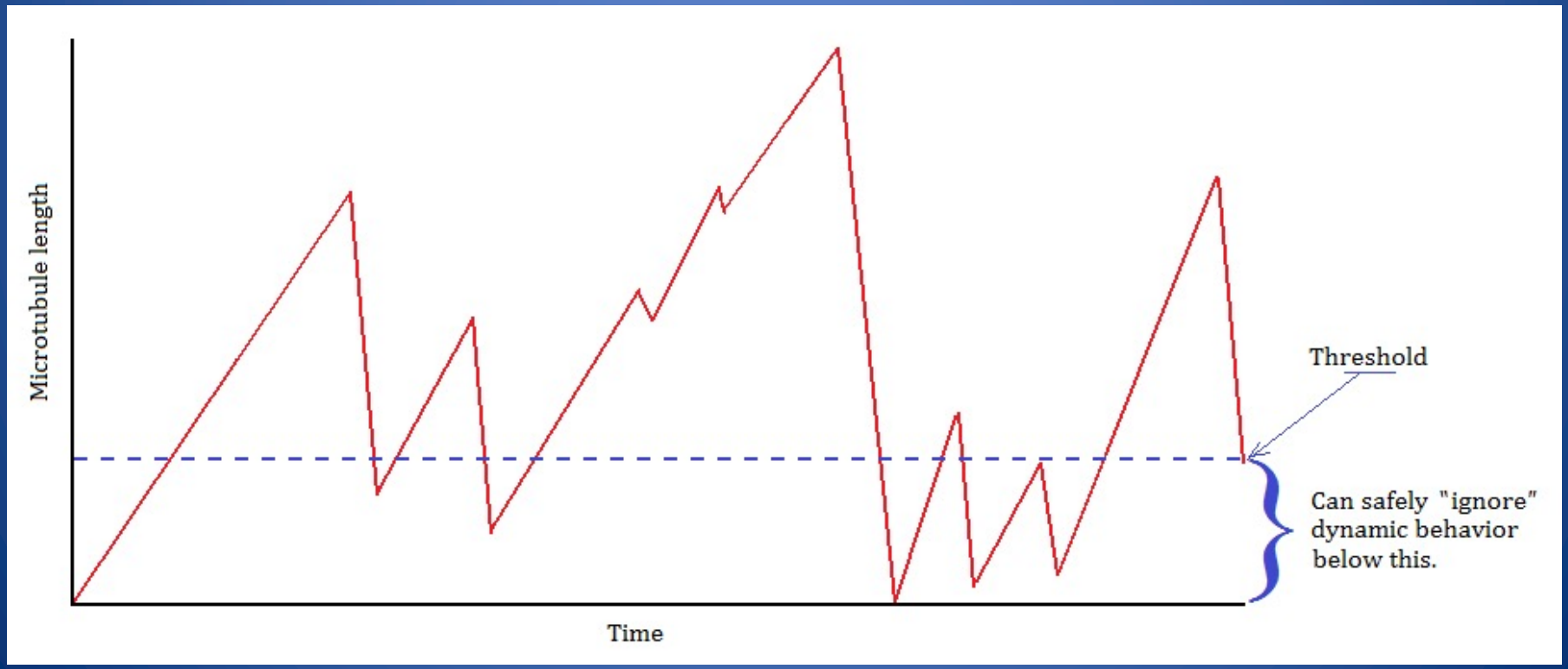
These are geometrical restrictions that classifies the change in height of the microtubule along with the "flatness" of the slope with respect to time. Since these inputs limit both changes simultaneously, it allows the researcher to identify where flat stutters occur during the growth phase.



Step 2: Classification.

`NUC_HEIGHT_THRESHOLD_INPUT = 75;`

This is a threshold where you effectively "ignore" dynamic behavior below this minimum. This is analogous to the length of the microtubule being "too short" for anything to be seen experimentally; in other words, it is too close to the microtubule seed to analyze any changes.



Step 2: Classification.

```
KMEANS_DIAGNOSTIC_AllFLAG = 0;
KMEANS_DIAGNOSTIC_PosSlopeFLAG = 0;
KMEANS_DIAGNOSTIC_NegSlopeFLAG = 0;
```

AllFLAG will generate plots to develop *K*-means for all segments, while **PosSlopeFLAG** and **NegSlopeFLAG** will do the same for positive and negative slopes, respectively. The default for all three parameters is 0 (off) while 1 turns it on.

NOTE:

- For any of these flags, turning them on (setting a value of 1) will run STADIA in "Diagnostic Mode". Please see the accompanying "STADIA Diagnostic Mode Tutorial" document for instructions on how to use and interpret the results generated from the Diagnostic Mode.

- If all these flags are turned off (set to a value of 0), then STADIA will run in "Automated Mode", which will use parameters and produce output as described throughout the remainder of this tutorial.

Step 2: Classification.

```
KMEANS_NumClust_PosSlope = #;
KMEANS_NumClust_NegSlope = #;
```

These commands set the number of clusters used to identify all dynamic instability phases. The **PosSlope** and **NegSlope** values are limited to 1, 2, or 3 and do not have to be the same.

Step 2: Classification.

```
KMEANS_Pos2_Option = 'A';
KMEANS_Neg2_Option = 'B';
```

These are *K*-means classification options only if $k = 2$ is chosen.

For the positive slope segments, `'A'` uses two growth phases (long and brief growth) and `'B'` uses one up-stutter phase and one long growth phase.

For the negative slope segments, `'A'` uses two shortening phases (long and brief shortening) and `'B'` uses one down-stutter phase and one long shortening phase.

Step 3: Analysis.

Step 3: Analysis.

STADIA will produce seven (7) plots:
1 – Microtubule length history plot
2 – Clustered results for scaled and standardized slope segments
3 – Classification results within the DI phase variable space
4 – Length history plot with color labels of DI phases for each segment
5 – Average measurements for the different DI phase segments
6 – Total measurements for the different DI phase segments
7 – Resulting measurements for the possible changes in DI phase

Note: MacOS will automatically open all the figures while Windows will generate them but you will need to Maximize each plot in order to see them on the screen.

Step 3: Analysis.

Two key commands are:

`PLOT_FIG_# = 1;`

Set = 1 **plots** the figures while 0 does not. The default setting is 1.

`SAVE_FIG_# = 1;`

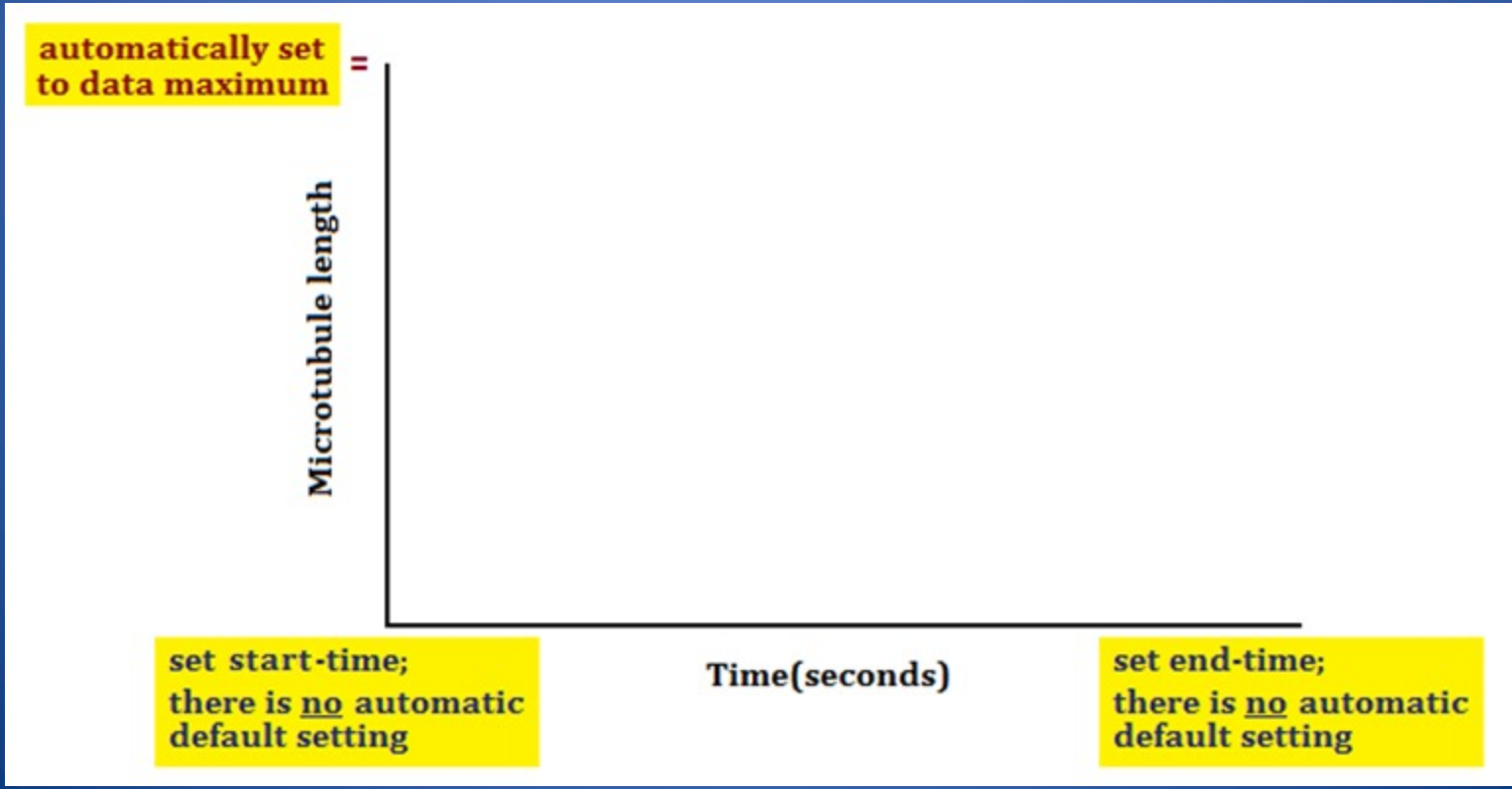
Set = 1 **opens** the figures while 0 saves and closes the figures. Default is 1.

Note: `SAVE_FIG_3_movie = 0;` creates an animated GIF file in .gif file format that spins a 3D plot of the DI classification.

Step 3: Analysis.

FIG_WINDOW_START = #;
FIG_WINDOW_END = #;

These set the viewing window for Figures 1 and 4 in time values. The MT height window is automatically set to the data maximum. There is no initial default setting for the start-time or end-time. Be sure to set the start-time and end-times before running STADIA.



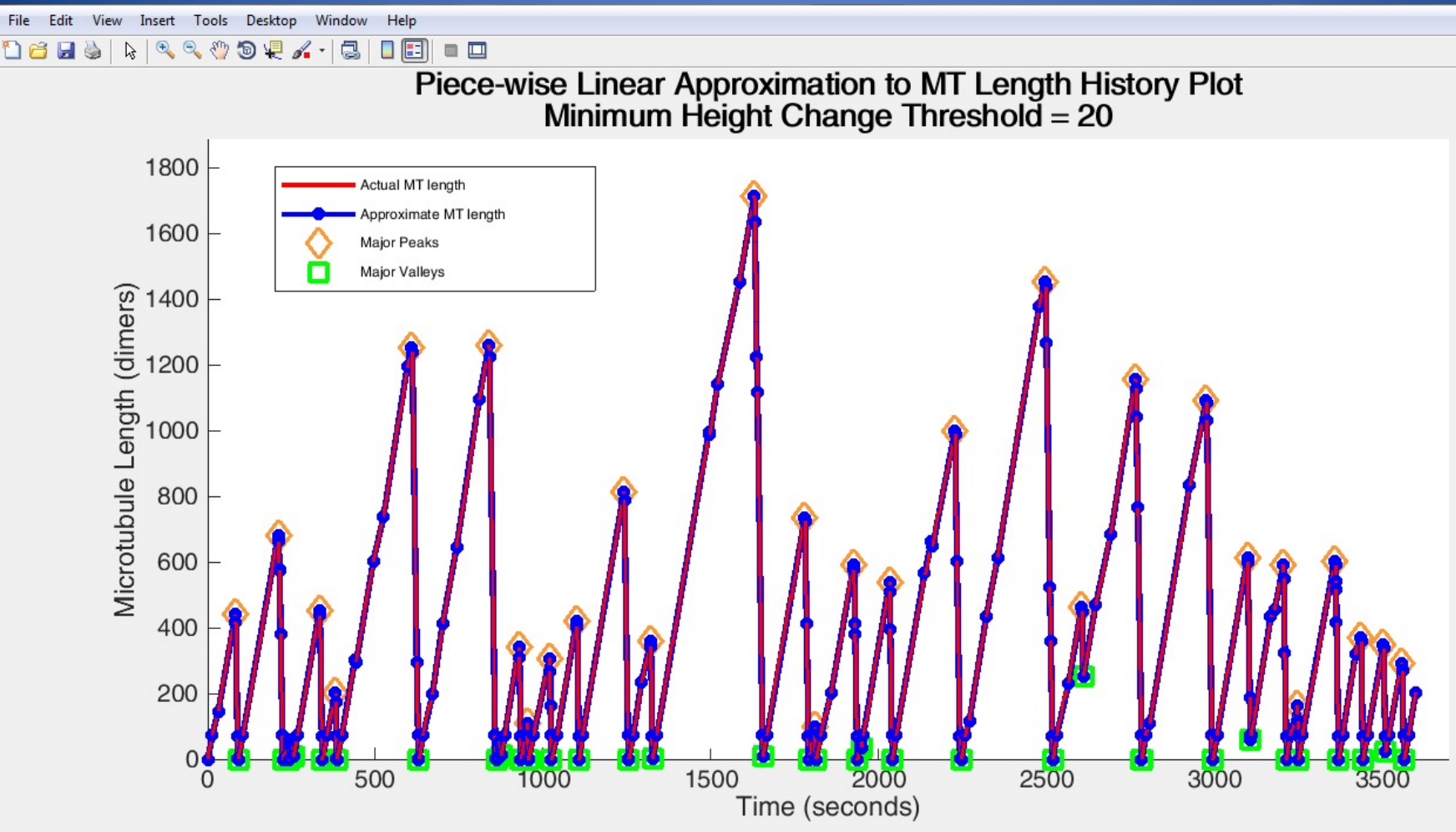
Step 3: Analysis.

`SAVE_MATLAB_WORKSPACE = 0;`

Set = 1 if you wish to save the variables in the MatLab™ workspace as a `.mat` file; the time and length data from the input data files won't be saved because they are already in your input files.

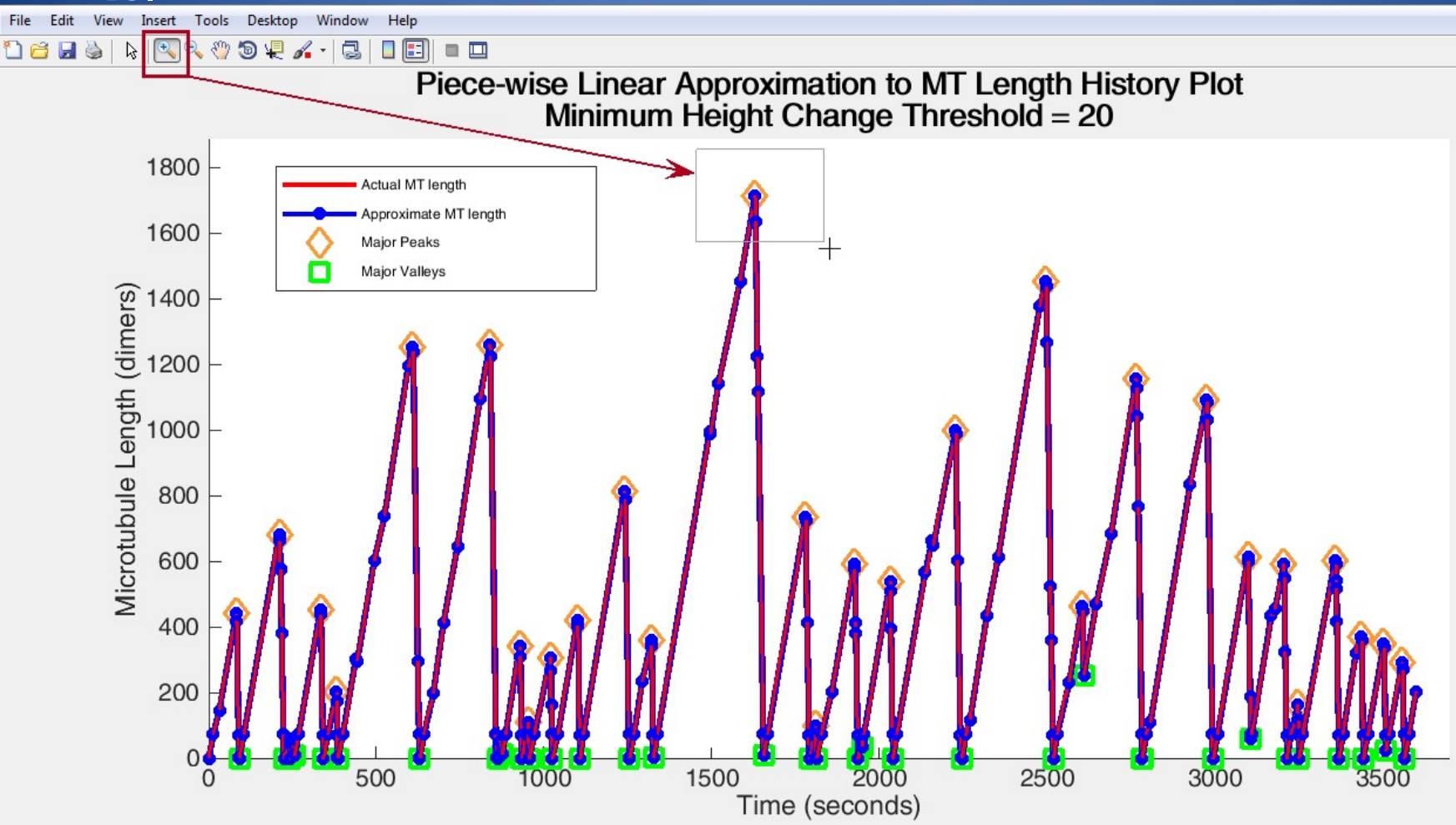
Step 3: Analysis.

The first figure (Figure 1 below) is the microtubule length history plot with the actual lengths in red and linear approximations in blue. Major peaks and valleys are also plotted.



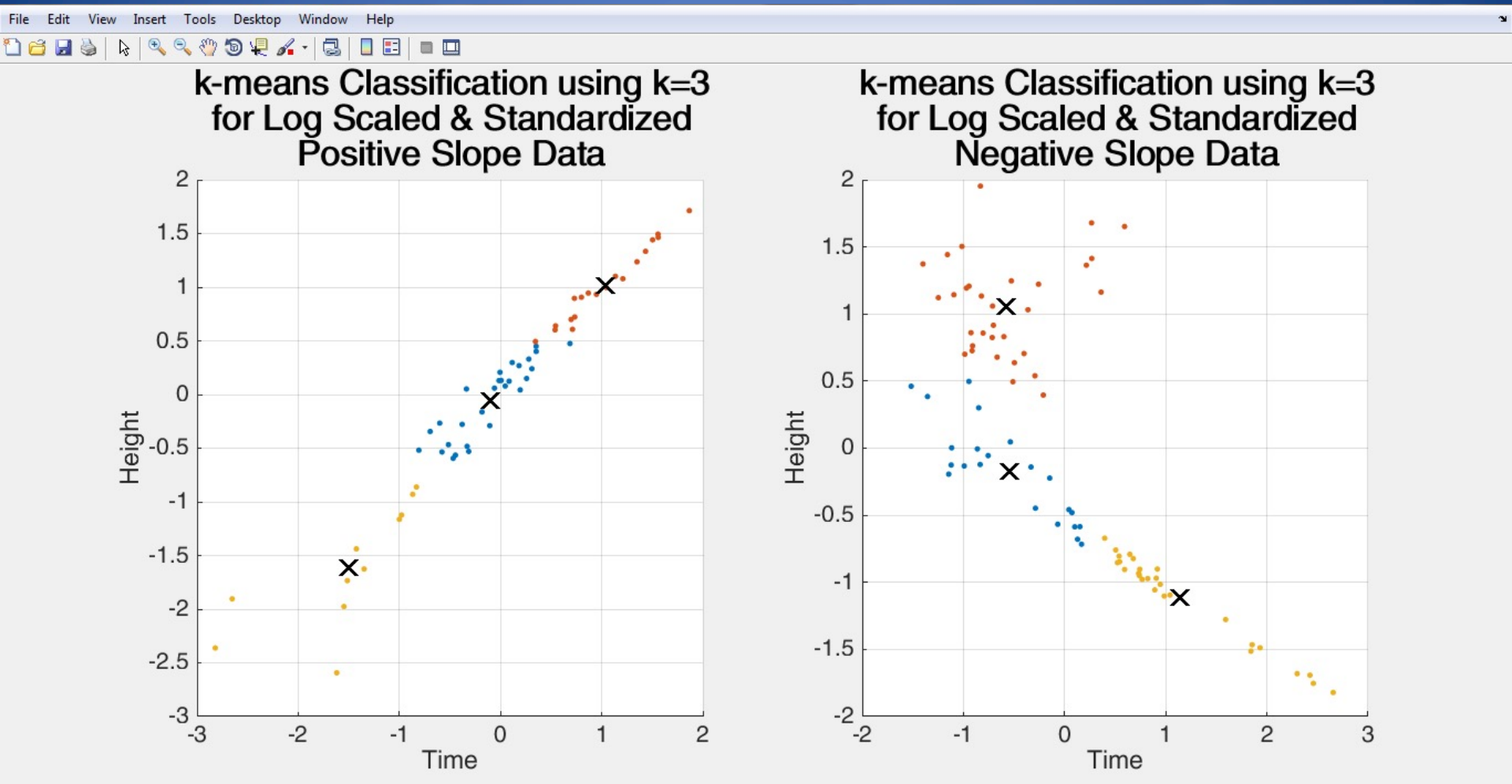
Step 3: Analysis.

You can use the zoom tool (upper left as shown) to progressively hone in on points you wish to study. Another way of scaling up the image is to use *x*- and *y*-limits in the MatLab™ command window, as illustrated in Slide 45.



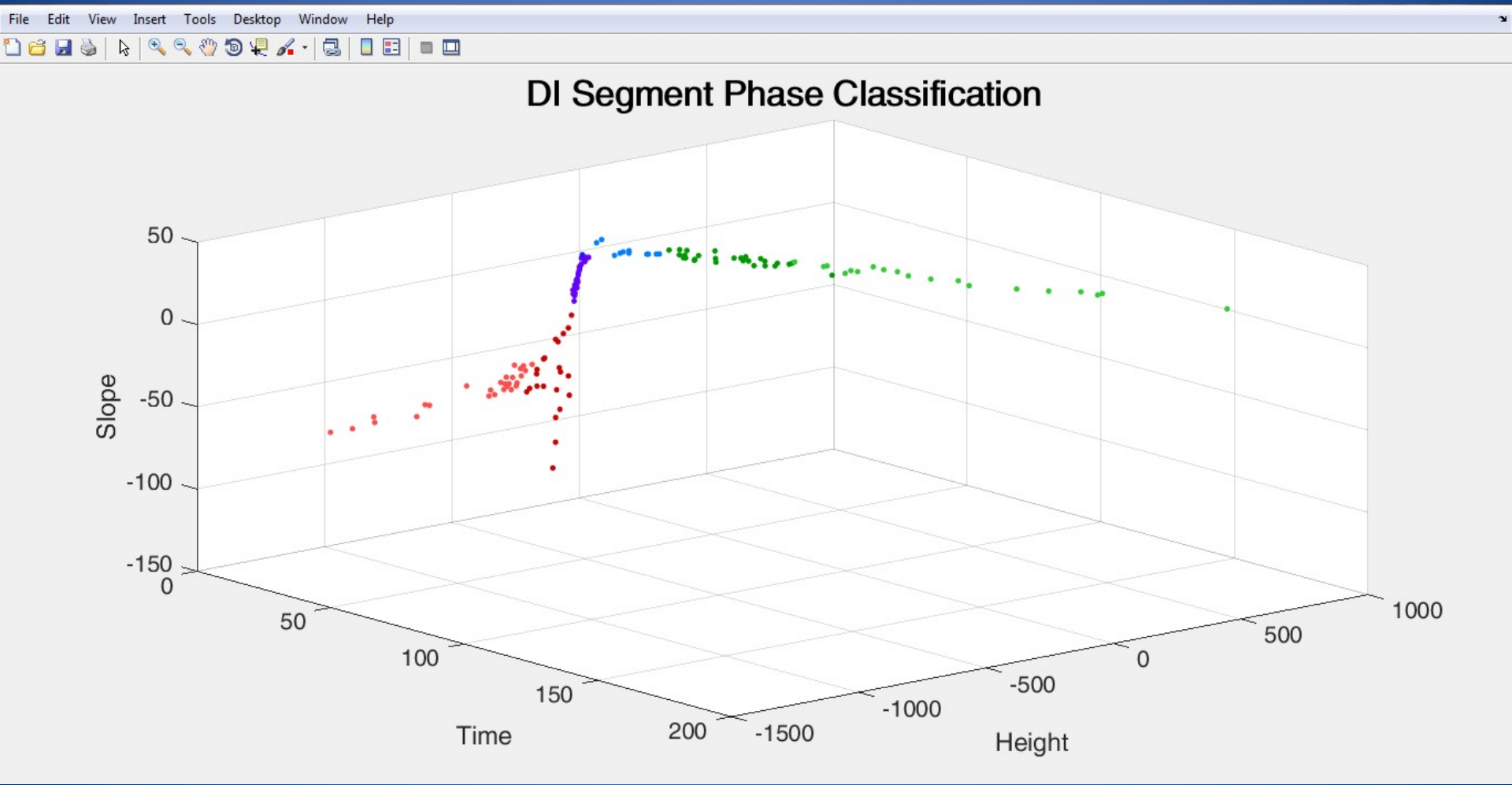
Step 3: Analysis.

Figure 2. Since we are working with three different behaviors (growth, stutter, and shortening), the *K*-means classification uses $k = 3$. These plots show the means for each cluster.



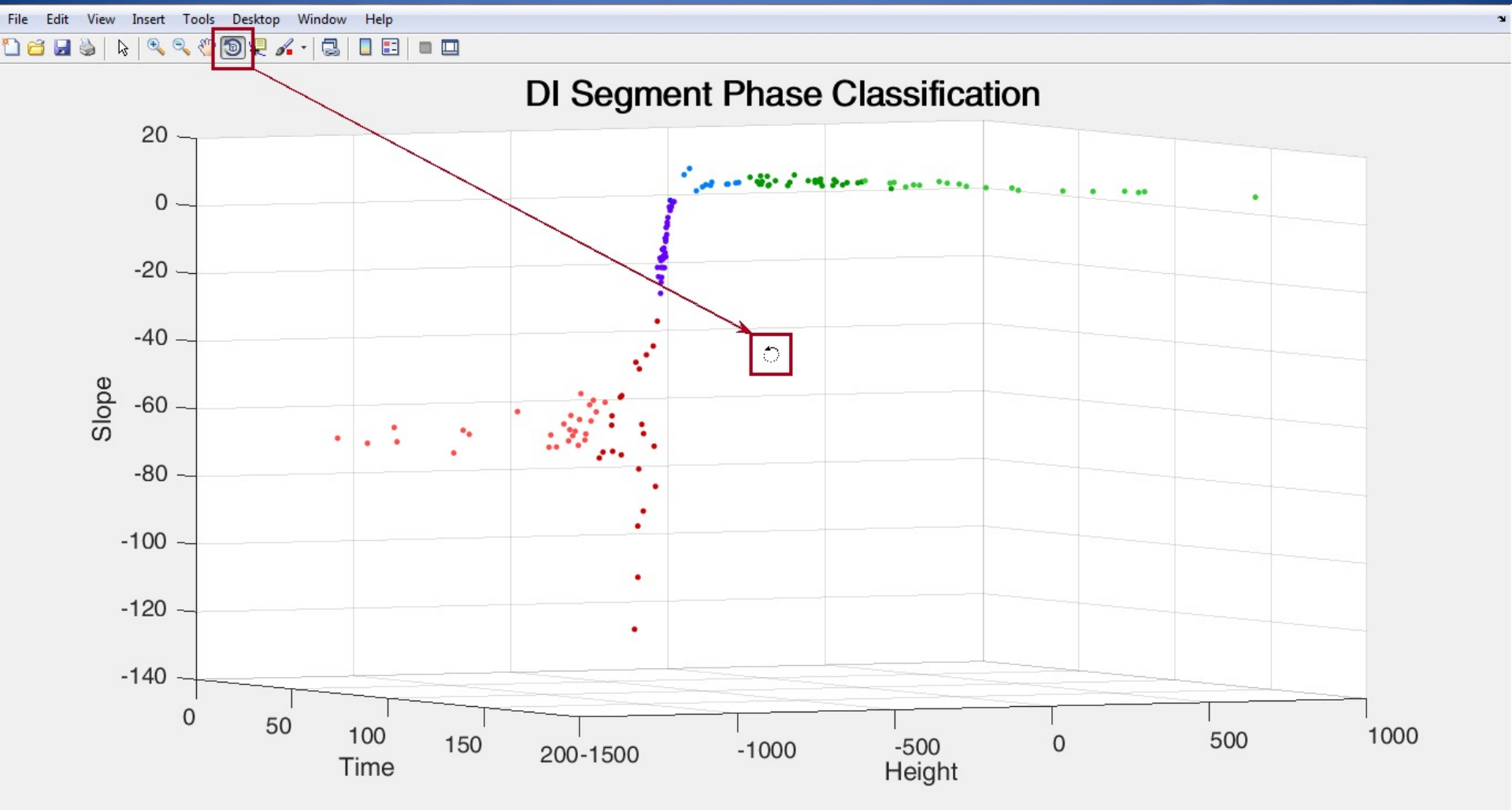
Step 3: Analysis.

Figure 3 shows classification results within the dynamic instability phase variable space; illustrating growth, stutter, and shortening phases within their *k*-means clusters using green, blue, and red, respectively.



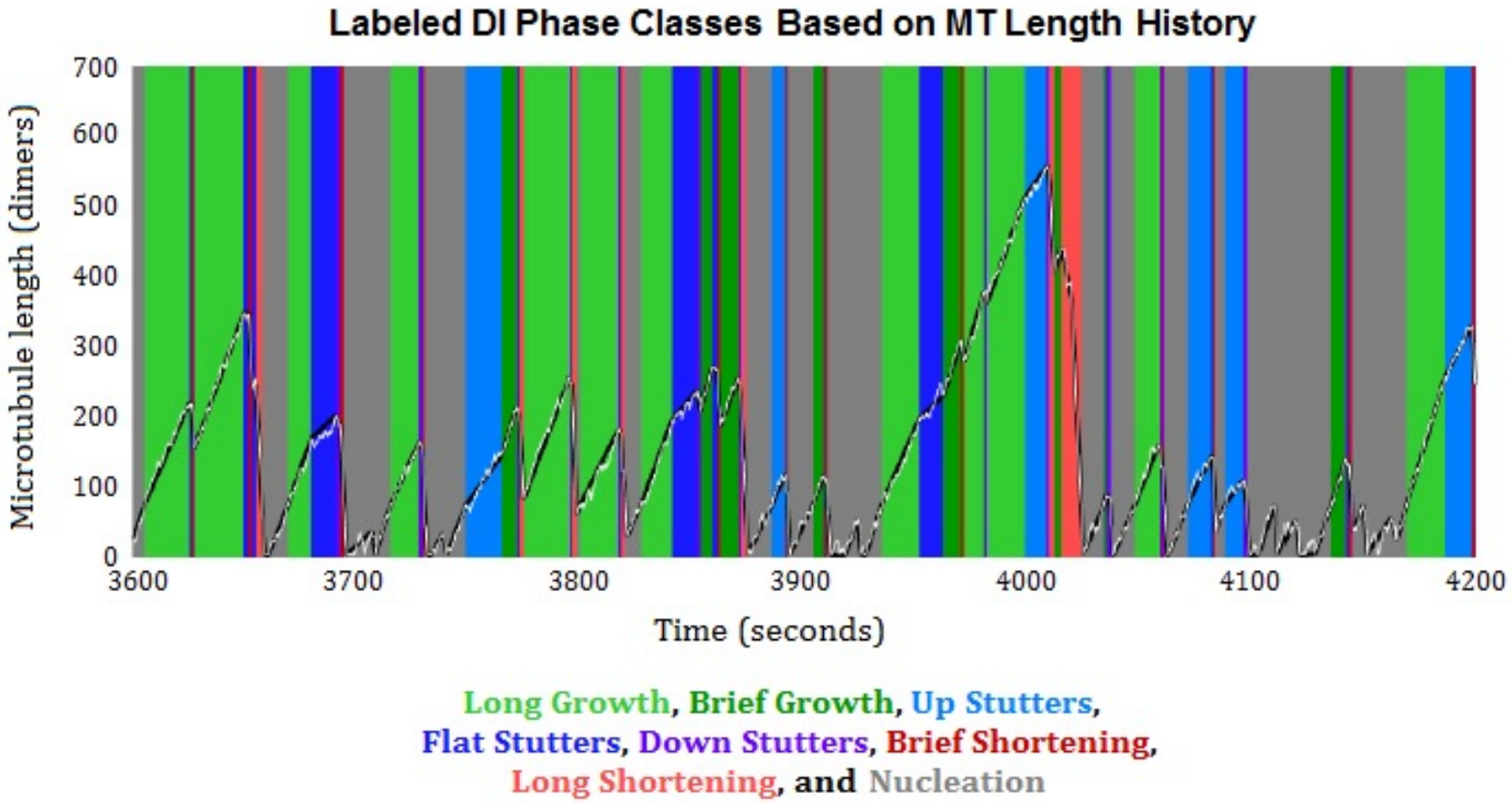
Step 3: Analysis.

Since this is a 3-D plot, you can use the rotate tool (shown below) to view the relationships between slope, height, and time from different perspectives in order to clarify growth and shortening behavior.



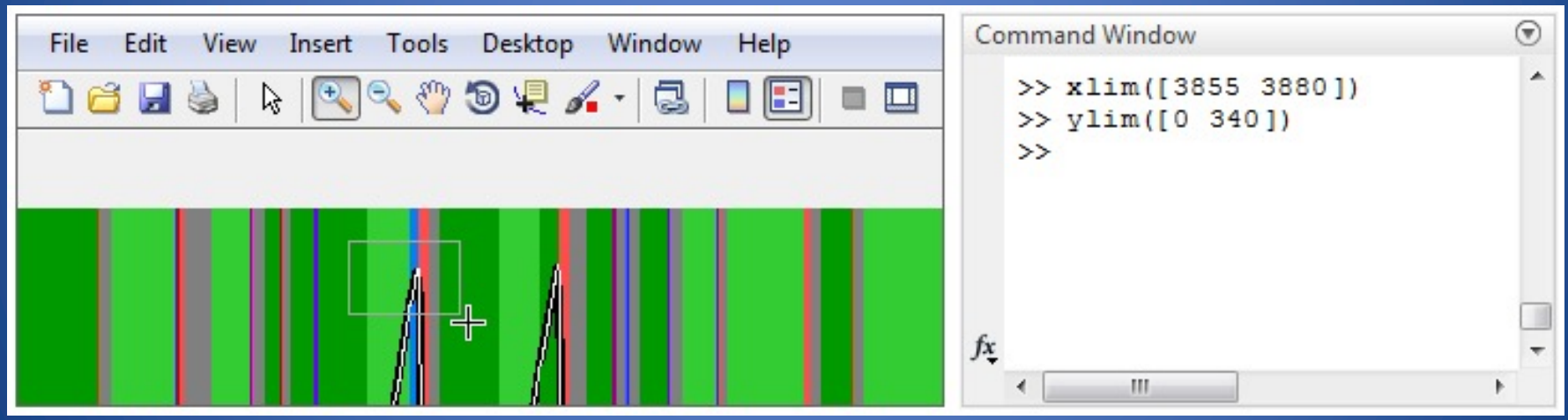
Step 3: Analysis.

Figure 4. The key intervals in the dynamic instability phase classes are those labeled in blue; they represent stutters. As you can see, they do not occur at every growth-to-shortening transition, but they do occur often enough to be worth closer examination.



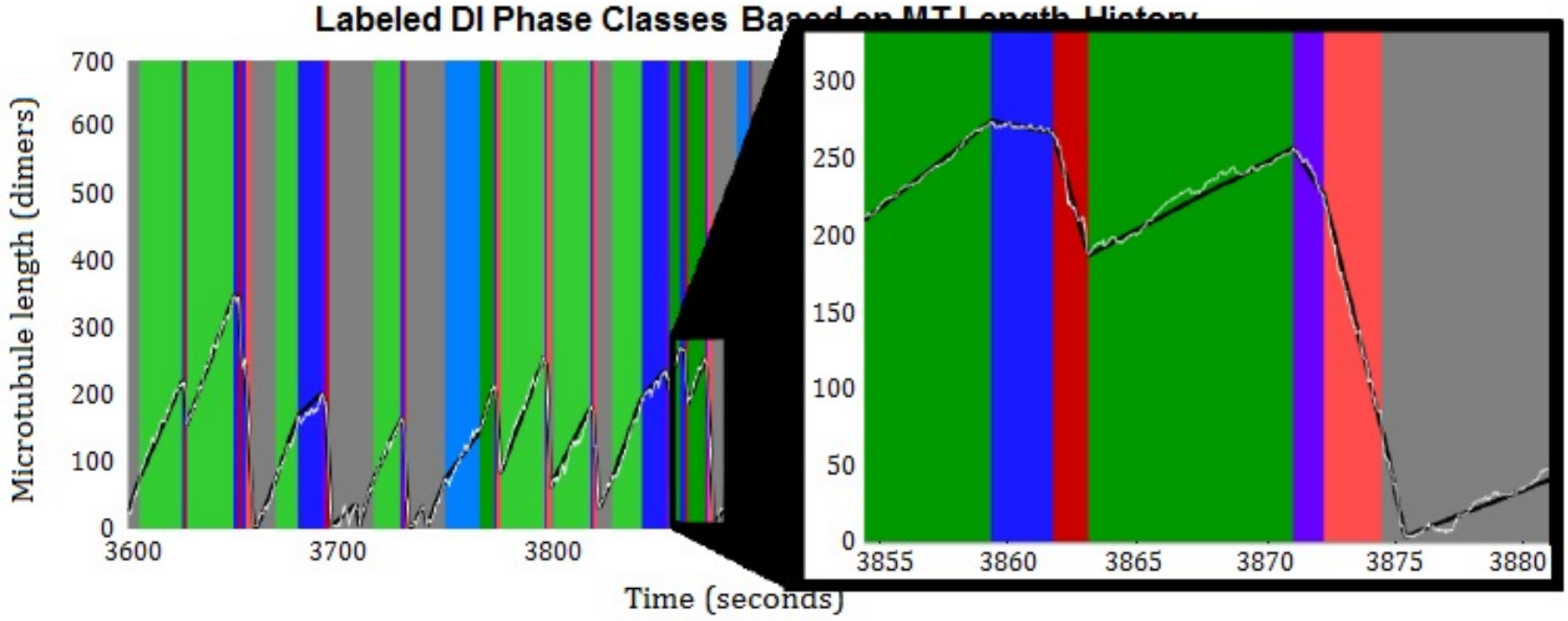
Step 3: Analysis.

By changing the observing window, either by “zooming in” on the region where stuttering appears to occur (shown at bottom left in the actual generated plot) or by direct command (in the MatLab™ Command Window, below right), you can then focus in on those transition regions.



Step 3: Analysis.

By zooming in, you can see more subtle transitions from growth to shortening of microtubules as well as compare the linear approximation to the actual data.



Long Growth, Brief Growth, Up Stutters,
Flat Stutters, Down Stutters, Brief Shortening,
Long Shortening, and Nucleation

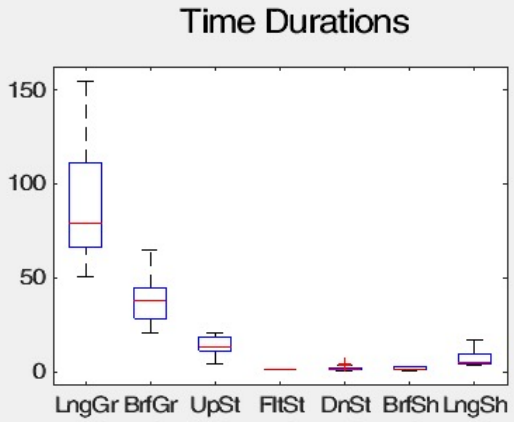
Step 3: Analysis.

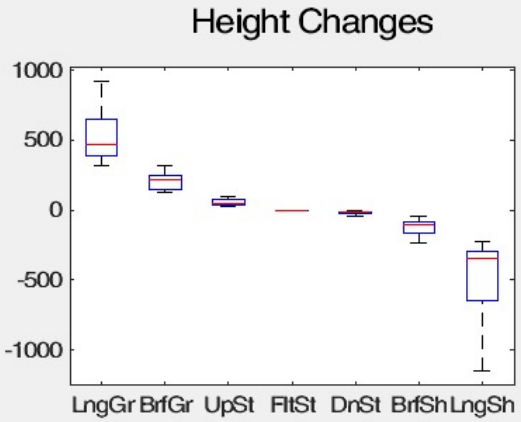
The last three figures yield graphs and tables suitable for more advanced statistical analysis. Figure 5, for example, shows time, height, and slope means and standard deviations with a set of whisker plots for each.

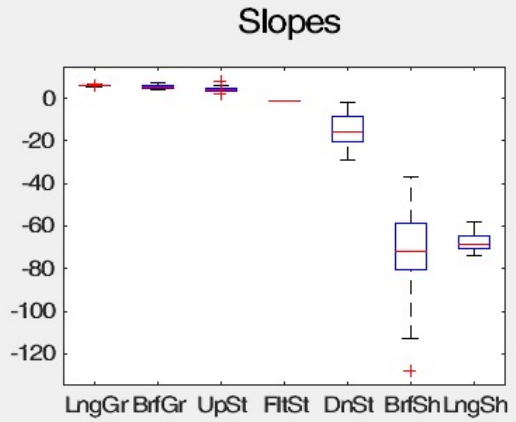
File Edit View Insert Tools Desktop Window Help

Segment Measurements for each DI Phase

	# of Segments	Time Mean	Time StDev	Height Mean	Height StDev	Slope Mean	Slope StDev
LngGr	19	87.6	28.4	525.5	169.1	6.0	0.3
BrfGr	28	37.3	10.4	205.3	60.3	5.5	0.8
UpSt	11	13.6	5.7	55.8	26.0	4.4	1.6
FltSt	1	1.4	0.0	-2.1	0.0	-1.4	0.0
DnSt	32	1.7	0.9	-20.7	10.8	-14.3	7.6
BrfSh	22	1.8	0.9	-123.4	59.9	-72.4	22.1
LngSh	27	7.0	3.9	-475.4	272.6	-67.5	4.5



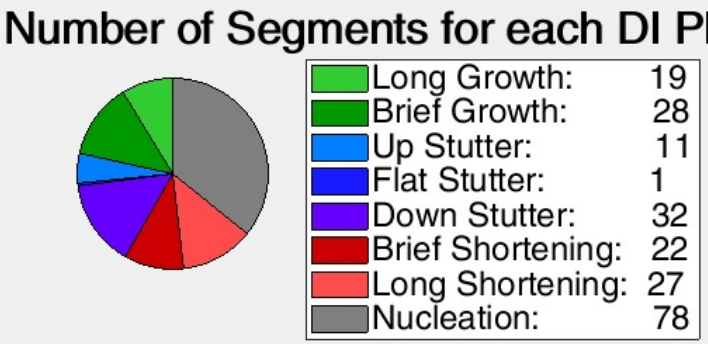


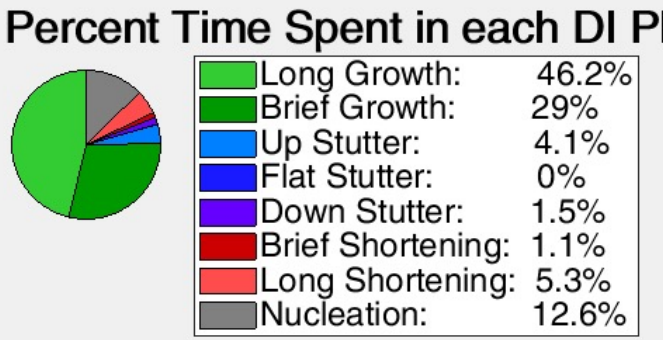


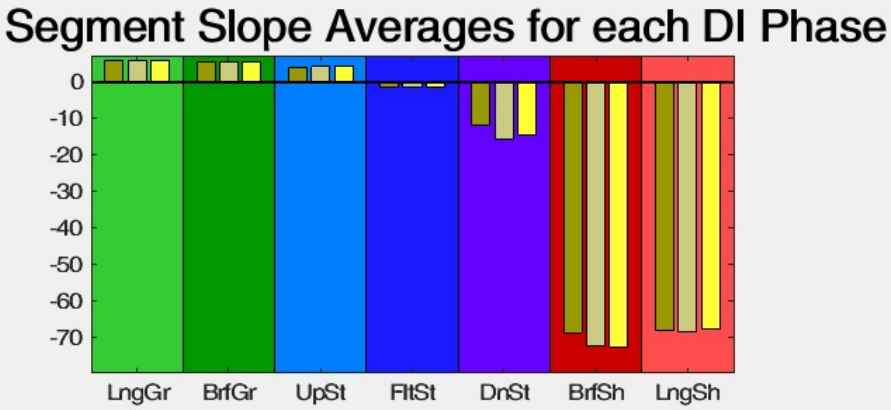
Step 3: Analysis.

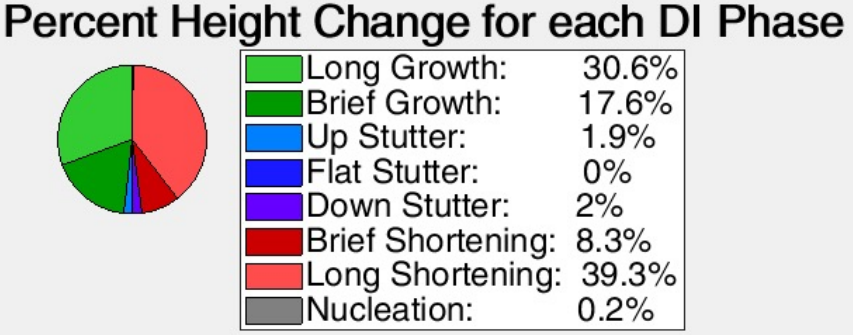
Figure 6 lists measurements for the different dynamic instability phase segments, including % time and height changes as well as the actual number of segments for each phase (growth, stutter, and shortening).

File Edit View Insert Tools Desktop Window Help





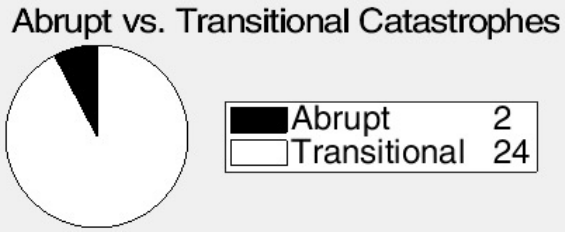


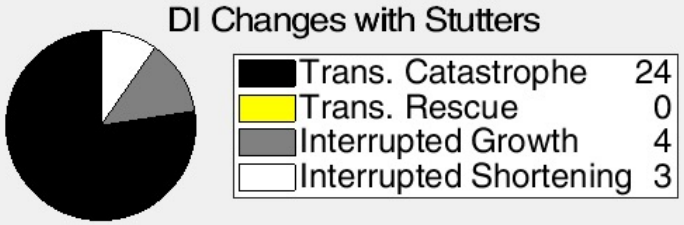


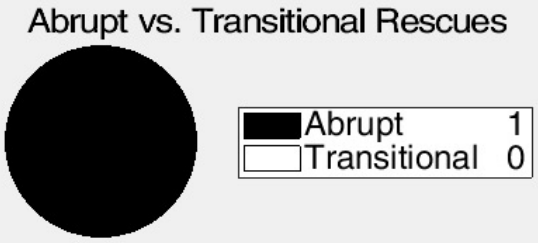
Step 3: Analysis.

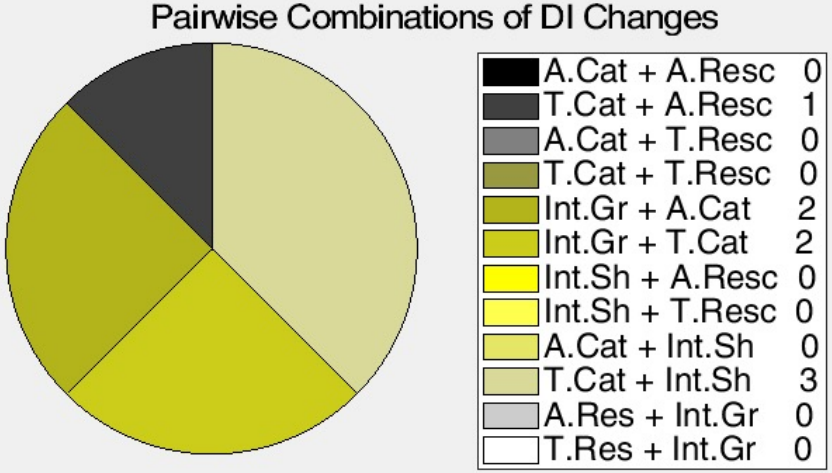
And, finally, Figure 7 illustrates abrupt vs. transitional catastrophes and dynamic instability changes with stutters along with phase change measurements and pairwise combinations of sets of changes.

File Edit View Insert Tools Desktop Window Help









DI Phase Change Measurements

	Number	Frequency
Abrupt Cat.	2	7.3808e..
Trans. Cat.	24	0.008
Abrupt Rescues	1	0.004
Trans. Rescues	0	0
Interrupted Growths	4	0.001

Step 3: Analysis.

Other STADIA output:

1. Segment and DI label info:
   - Test file of data for segments identified from inputted length history
   - Info includes:
     - Time and MT length at segment start
     - Time duration, Height change and Slope of each segment
     - DI Phase labeled for each segment
       - Nan = Nucleation segment
       - -12 = Long Shortening
       - -11 = Brief Shortening
       - -1 = Down Stutter
       - 0 = Flat Stutter
       - 1 = Up Stutter
       - 11 = Brief Growth
       - 12 = Long Growth
   - Filename: **`DIsegmentPhaseData.txt`**

Step 3: Analysis.

Other STADIA output:

2. DI parameters
   - Text file organizing the input values used to run STADIA and the corresponding measurements relevant for DI analysis
   - Measured DI parameters include:
     - Slope statistics for each phase (weighted average, mean, median, standard deviation)
     - Total time spent in each phase
     - Number of segments identified in each phase
     - Frequency of phase transitions
     - Number of pairwise phase transition combinations
   - Filename: `DI_parameters_output.txt`

Step 3: Analysis.

Other STADIA output:

3. Phase transitions
   - Helps identify where to find a change in phase identified in the length history data for each possible combination of phases
   - Organized by starting data point index, starting time, and height at starting time of the first segment that's part of the transition.
   - Generated filenames for each possible phase:
     - `Abrupt_Catastrophe_output.txt`
     - `Abrupt_Rescue_output.txt`
     - `Transitional_Catastrophe_output.txt`
     - `Transitional_Rescue_output.txt`
     - `Interupted_Growth_output.txt`
     - `Interupted_Shortening_output.txt`

Step 3: Analysis.

STADIA Output Saving Format:

- New files/directories created with unique time-stamp
  - Organizes STADIA outputs with the time that code was run
  - Prevents file over-writing

- Running in **Automated Mode** (all Diagnostic Flags = 0)
  - Creates new directory with the following name criteria:
    - "`STADIA_Output_for_`" as prefix
    - The `input length history filename`
    - `Time-stamp` for STADIA run as suffix
  - Example directory name:
    - `STADIA_Output_for_length_13PF_10uM_10hr_20211027_09h48min28sec`

- Running in **Diagnostic Mode** (any Diagnostic Flags = 1)
  - Saves multiple files with different time-stamp in the same directory with the following name criteria:
    - "`STADIA_DiagnosticMode_Files_for`" as prefix
    - The `input length history filename`
  - Example directory name:
    - `STADIA_DiagnosticMode_Files_for_length_13PF_10uM_1hr`

And the rest is up to you...

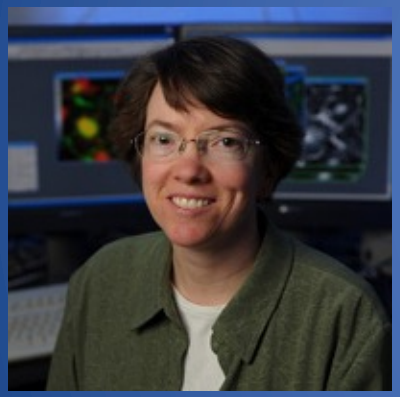
All that's left is for you to input your files, set your parameters, let the program run, analyze the data, and publish your results.

We hope this tutorial will assist you in your research using this new dynamic instability measurement method. Please let us know how the program works, any bugs you've run across, and any improvements to the STADIA program and the tutorial you'd like to see.

Thank you.

Contributors.

Dr. Shant Mahserejian
Research scientist
Applied Statistics and Computational Modeling Group
Pacific Northwest National Laboratory
shant.mahserejian@pnnl.gov



Dr. Holly Goodson
Professor of Biochemistry
Department of Chemistry and Biochemistry
University of Notre Dame
holly.v.goodson.1@nd.edu

Kristopher Murray
Integrated Biomedical Sciences
Department of Chemistry and Biochemistry
University of Notre Dame
kmurray5@nd.edu

Contributors.

Michael Sinclair
Tutorial author
Physics & calculus instructor
Kalamazoo Area Mathematics and Science Center
dragonphysics@gmail.com
All illustrations by Mahserejian and Sinclair, 2019.

Contributing members of the team:

Jared Scripture
University of Notre Dame

Riya Patel
Penn High School and Indiana University

Peter Martin
Trinity School at Greenlawn

Contributors to design and development of STADIA code:

Shant Mahserejian
Ava Mauro
Jared Scripture
Jun Li
Erin Jonasson
Mark Alber
Holly Goodson

References.

1. Shant M. Mahserejian, Jared P. Scripture, Ava J. Mauro, Elizabeth J. Lawrence, Erin M. Jonasson, Kristopher S. Murray, Jun Li, Melissa Gardner, Mark Alber, Marija Zanic, Holly V. Goodson. Quantification of Microtubule Stutters: Dynamic Instability Behaviors that are Strongly Associated with Catastrophe. bioRxiv. 2020. doi: https://doi.org/10.1101/2019.12.16.878603

2. Patel, Riya J., Kristopher S. Murray, Peter O. Martin, Michael Sinclair, Jared P. Scripture, Holly V. Goodson, and Shant M. Mahserejian. 2020. "Using STADIA to Quantify Dynamic Instability in Microtubules." In *Methods in Cell Biology*, 158:117–43. Academic Press Inc. https://doi.org/10.1016/bs.mcb.2020.03.002