Hessian Blob Particle Analysis in Igor Pro

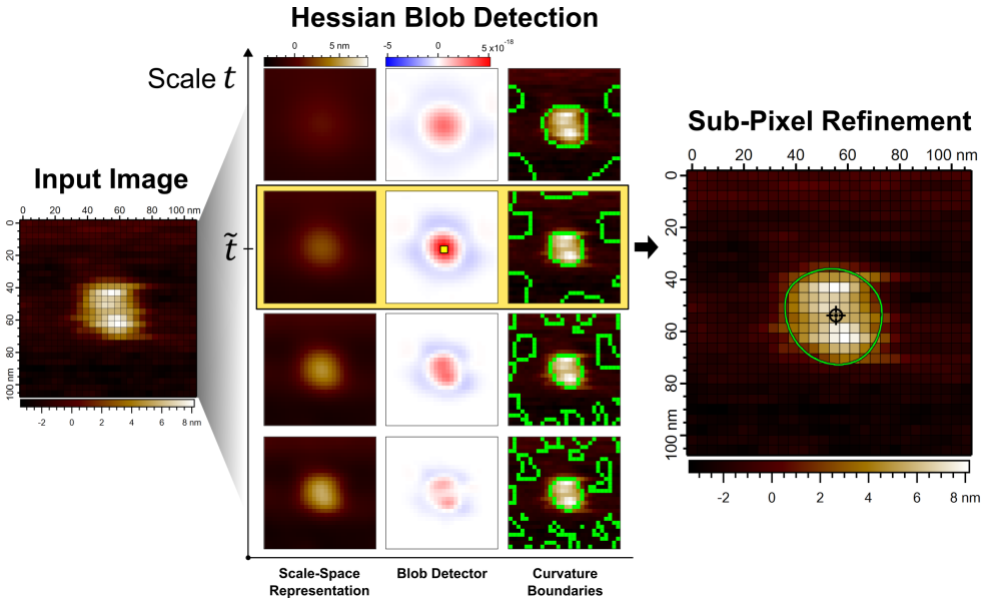
B.P. Marsh[2], G.M. King Laboratory[1]

[1]University of Missouri, Department of Physics and Astronomy
Email: kinggm@missouri.edu

[2]Stanford University, Department of Applied Physics
Email: marshbp@stanford.edu

Introduction

The Hessian blob algorithm is a general-purpose particle detection algorithm, designed to detect, isolate, and draw the boundaries of roughly "blob-like" particles an image. Hessian blobs are defined in the language of differential geometry and scale-space representation theory, lending them resistance against noise and consistency in their detection. Moreover, interpolation methods allow Hessian blobs to be detected to subpixel precision. The algorithm was born out of a need for a more precise and well-defined definition of a "particle" in the analysis of atomic force microscopy images, where it is used to detect single biomolecules in large-scale images.



Hessian blobs are *not* a catch-all detection method, however. Hessian blobs detect only features which appear blob-like, meaning in a more precise sense that the feature must be composed of a convex hull. Extended, linear features are not well detected as Hessian blobs (though an extension to such features is in development!).

In this tutorial, we walk through the use of the Hessian blobs algorithm with our provided software, written in the scientific analysis software Igor Pro (https://www.wavemetrics.com/). No prior experience with Igor Pro or coding is required, we will proceed in baby steps.

The full paper, "The Hessian Blob Algorithm: Precise Particle Detection in Atomic Force Microscopy Imagery" outlines the algorithm, benefits, implementation, and associated

concepts in scale-space representation theory. It may be found freely available at Scientific Reports and on our laboratory website. doi:10.1038/s41598-018-19379-x

We are delighted if the Hessian blob software is useful for your research or any other academic endeavor. If you use the algorithm in your research, please cite the Hessian blobs paper shared above. For commercial applications or otherwise, please contact us or the Office of Technology Management and Industrial Relations over email at tmir@missouri.edu or by phone (573) 882-6013.

We hope you find success using the Hessian blob algorithm and our provided software! Please do not hesitate to contact us with questions, suggestions, or bugs.

Contents

I. Preliminaries

In this tutorial, we go through the basics of loading images into Igor Pro 7 for analysis and creating a data folder for the loaded images. We will also review some of the basic concepts and language used in the Igor Pro environment.

Launching Igor Pro will begin what is called a new Igor "experiment". This is a moniker for a new file, which will hold all of our data and the results of our analysis.

There are three main types of data in Igor, these are variables, strings, and waves. Variables are numbers that we store and name on the computer, such as the variable `a = 3`. Strings are similar, but instead of numbers they are just sets of characters, such as `str = "Hello"`. We may use these variables to store information, perform calculations with them, or change them as we desire. Finally, waves are collections of variables or strings. Most often, we deal with numeric waves, which hold sets of numbers in a simple array. These waves might hold one-dimensional data arrays such as time-series data, two-dimensional data such as images, or even three and four-dimensional volumetric data.

The two main components of Igor we'll need for the tutorial are the data browser and the command line. In an Igor experiment, all data (including waves, variables, and strings) is organized in the data browser folder system. In the data browser, we are able to view our data and add or delete data folders to keep our analysis organized. The command line, similar to any other programming environment, is our interface with Igor where we execute commands to run our analysis, edit waves and variables, or execute any other Igor command.

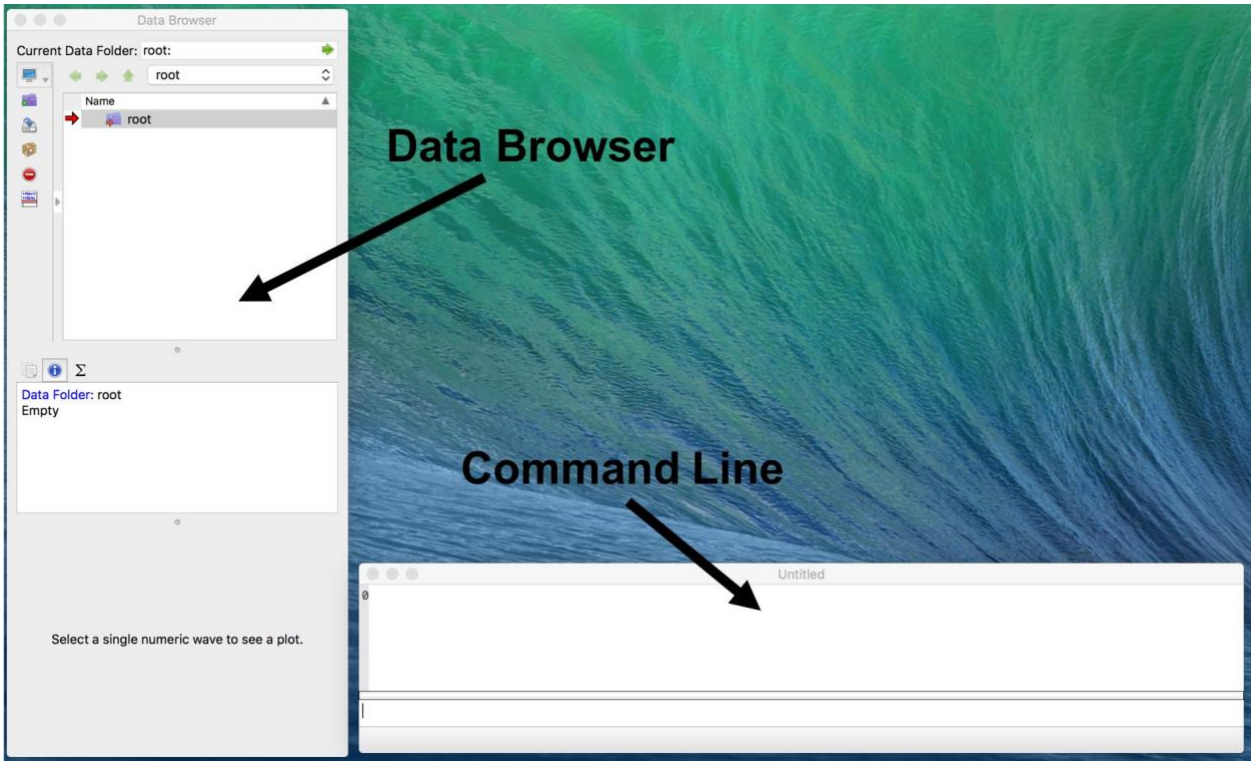


Figure 1: The Igor interface, including the data browser and command line. Note that if you are running Igor on a Windows machine, the entire experiment will appear in its own window. On a Mac (pictured here) the various Igor windows will all appear floating on your desktop.

Before we load any images into our experiment, let's create a new data folder to store them in. To create a new data folder, simply hit the new data folder button in the data browser and name your folder appropriately.

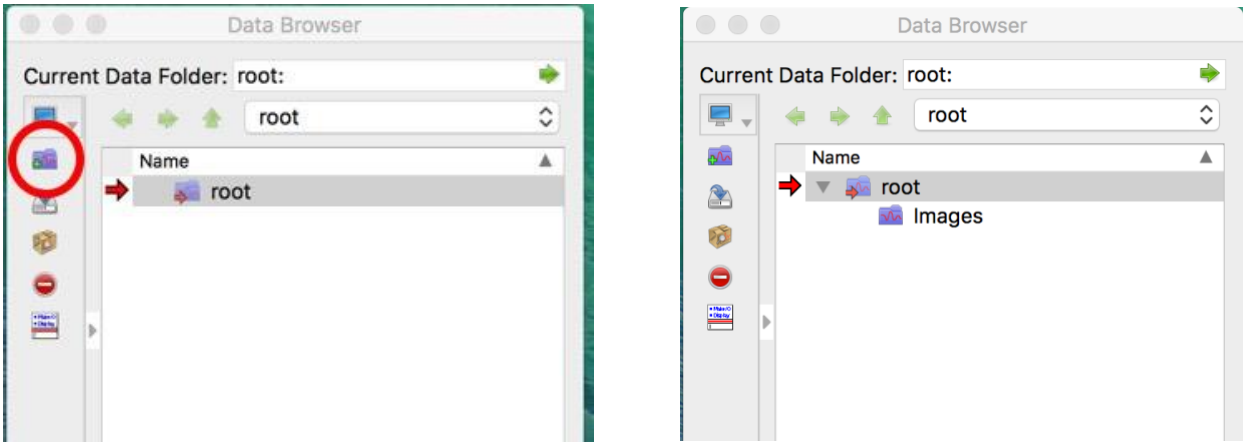


Figure 2: Creating a new data folder. In the data browser, click the new data folder button shown circled in red and name you data folder, here named "Images".

Your new data folder will appear beneath the root data folder, indicating that is inside the root data folder as everything will be. You'll notice that a red arrow points to the root data folder, this is an important point which indicates that the root data folder is the "current" data folder. It is important to keep track of which data folder is the current data folder, as any new or loaded data will automatically be put into this data folder. You can change the current data folder by right-clicking on any data folder and selecting "Set Current Data Folder".

Let's now load our images. Under the data tab at the top of the screen, we will need the "Load Waves" section. The option we choose in the "Load Waves" section will depend on how your images are stored. In order to perform our analysis, we will need to have the images in an Igor friendly format, AKA waves. If your data is already saved in native Igor format as ibw files, choose the Load Waves → Load Igor Binary option. If you images are stored in a standard image format such as tiff, jpeg, or png, choose the Load Waves → Load Image option. If your data is stored in a simple spreadsheet style such as Excel format or csv, you'll need the Load Waves → Load General Text or Load Delimited Text options.

After importing your images into Igor, they will appear in the current data folder. Let's move them into our Images data folder by simply dragging and dropping.

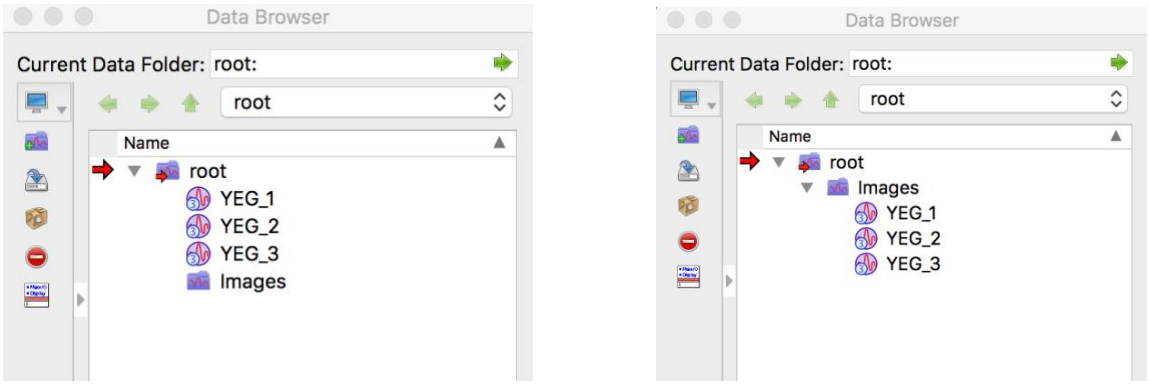


Figure 3: Loading images and moving them into a data folder.

To view our images, we may either right click on them and select "New Image" or use a command line function. To view an image using the command line, we need to introduce the concept of the wave path.

Every wave, such as our images, has both a name and a path. The name simply identifies the wave, whereas the path specifies the full location of the wave in the data browser. For example, here we loaded three waves, one of which is called "YEG_1". The full path of this wave "root:Images:YEG_1".

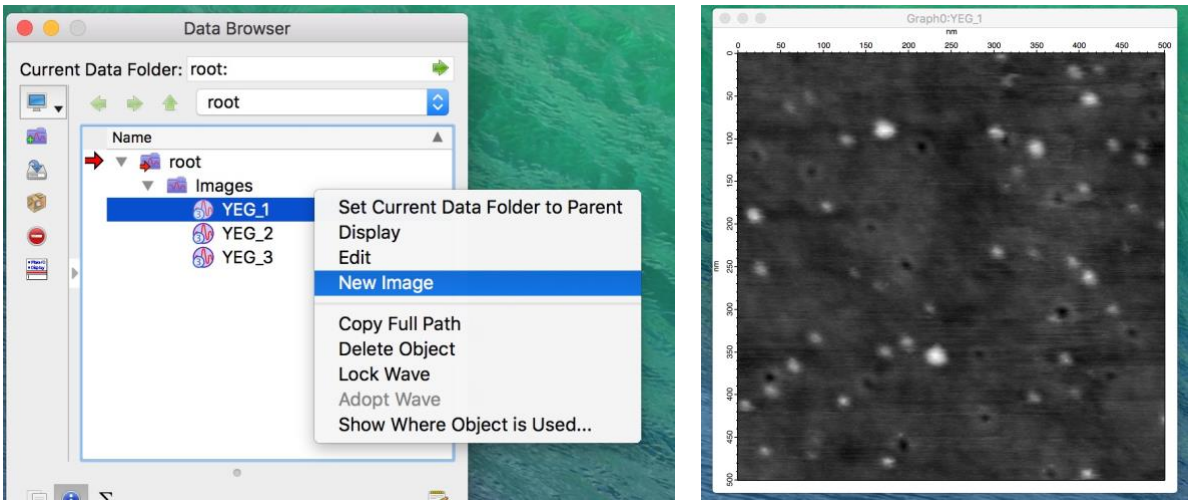


Figure 4: Viewing an image by right clicking and selecting "New Image".

To view an image using the command line, we use the Igor function NewImage. To execute the command, we type it into the command line along with the full path of the image we want to display, so Igor knows exactly which image you are talking about. For example, to view the YEG_1 image we type the following into the command line.

```
NewImage root:Images:YEG_1
```

Note that Igor automatically inverts the y-axis of the image. To view the image with an upward oriented y-axis, use the /F flag in the NewImage command:

```
NewImage/F root:Images:YEG_1
```

Also note that the NewImage function only works for two and three dimensional data, not one dimensional data. To view one dimensional data, we right click and choose the "Display" option or use the "Display" command analogously on the command line.

We may modify the appearance of our image, such as changing the color scale. To do so, simply right click inside the image and select "Modify Image". See the Igor manual for full documentation of Igor's image handling capabilities.

Next, we need to load what is called a procedure file, these are the codes we write to extend Igor's capabilities and design our own analysis procedures. The Hessian blob algorithm and the associated suite of analysis functions are all provided in a single procedure file which should be downloaded from our website, which should be named HessianBlobs_Vx.ipf. In Igor, go to File → Open File → Procedure to load the procedure file. Once loaded, it should look something like this.

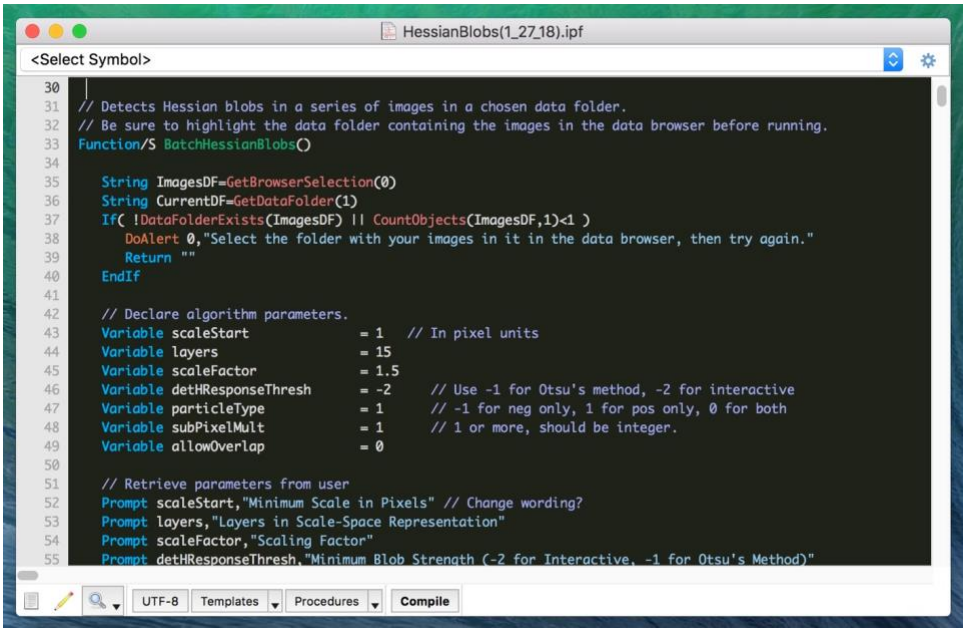


Figure 5: The loaded Hessian blobs procedure file.

All of the functions needed for Hessian blob analysis may be found here. To load all of the functions into Igor, you may need to hit the "Compile" button which will be present in the bottom of the procedure file. If it is not there, don't worry it has already been compiled. After compiling the code, all of the user-defined functions we provide have been loaded and you may minimize the window if you like.

To demonstrate how using these user-defined functions work, try using the `Testing` function we provide. In general user-defined functions may require some inputs from the user. The `Testing` function requires two inputs, a string and a variable. To run the function and supply the inputs, execute something like the following in the command line.

```
Testing("Some string",5)
```

The function should print a message in the command line history. If you have followed everything so far, congratulations, you have passed Igor boot camp and you're ready for tutorial two.

II. Image Preprocessing

The Hessian blob software comes equipped with two preprocessing routines which may be used to "clean up" your images before performing your analysis if you so choose. Igor Pro also contains a vast number of image processing methods, such as filtering and smoothing. In this tutorial we learn how to use the preprocessing routines provided by the Hessian blob software.

Before we begin, please note that the Hessian blob algorithm operates on what is called the scale-space representation of an image; this framework makes the algorithm largely insensitive to moderate levels of image noise. Thus, we recommend only applying the following preprocessing routines when the images are significantly distorted, as to minimize modification of your data. This point is made clear in the full paper, and a relevant figure is reproduced here at the end of this tutorial demonstrating Hessian blob invariance under imaging artifacts.

The first preprocessing routine is image flattening. In image flattening, we take each individual horizontal row of pixels in the image, determine a line / polynomial of best fit to that row of pixels, and subtract it off from the row of pixels. This effectively removes the background variation of each row pixels, and is common practice in such fields as scanning probe microscopy. The resulting image should have a more consistent and flatter background level, hence the name. In our provided implementation, one may choose any order of polynomial to fit to each row of pixels, and furthermore it is possible to mask off features of interest via a height threshold such that they do not interfere with the flattening.

The second preprocessing routine is called streak removal and is a more specialized routine which corrects for a kind of error seen in atomic force microscope images. These artifacts arise from tip-sample interactions and the "parachuting" tip effect, and present themselves as single pixel width horizontal streaks in the image. The streak removal function measures the level of "streakiness" at every pixel in the image, then identifies lines of pixels which consistently show streakiness levels much greater than the average streakiness level in the image. The user provides the number of standard deviations away from the mean streakiness which a line of pixels must attain before being considered a streak. Identified streaks are then removed and replaced by the average of their neighboring pixels above and below.

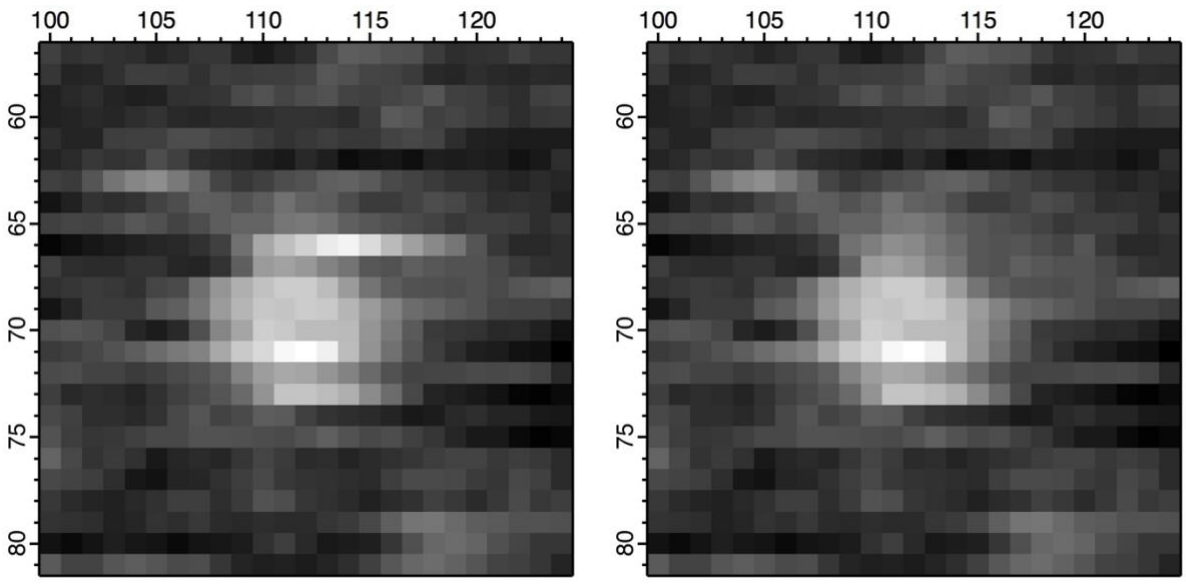


Figure 7: A before and after comparison of the same feature after using the streak removal algorithm, where we required 3 standard deviations from mean streak levels. On the left, an image feature contains an artificial "streak" across the image which is identified and removed on the right.

To begin the preprocessing, load the images and place them in their own data folder, as covered in tutorial I.

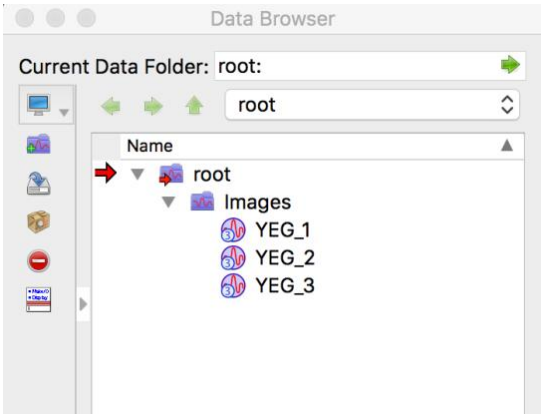


Figure 8: Loaded images placed in their own data folder.

It's wise to store the original images and perform the preprocessing on a duplicate set of images, that we can always go back to the originals if we need to start over. So, lets duplicate the data folder with our images. To do so, we use the DuplicateDataFolder command. We have to provide the full path to the data folder to be duplicated, and provide the path of the new data folder containing the duplicated images, including the name of the data folder. Calling our duplicate data folder Images_dup, we use the following command.

```
DuplicateDataFolder root:Images , root:Images_dup
```

In the data browser, we should now see two data folders.

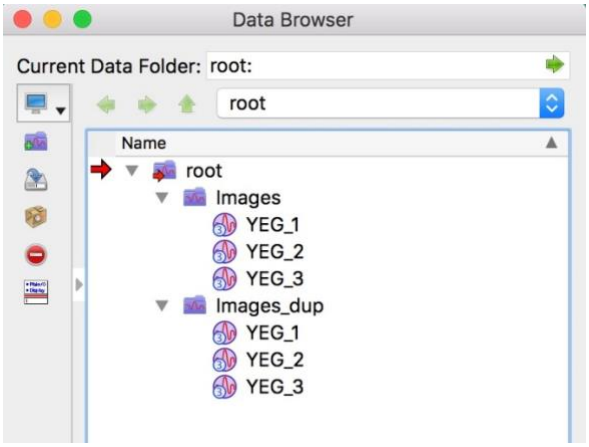


Figure 9.1: A duplicated data folder.

Next, we will execute a command to preprocess the images. However, in order for Igor to know where our images are located in the folder system, we must first highlight the data folder containing our images. To do so, simply single click on the data folder with the images you wish to preprocess.

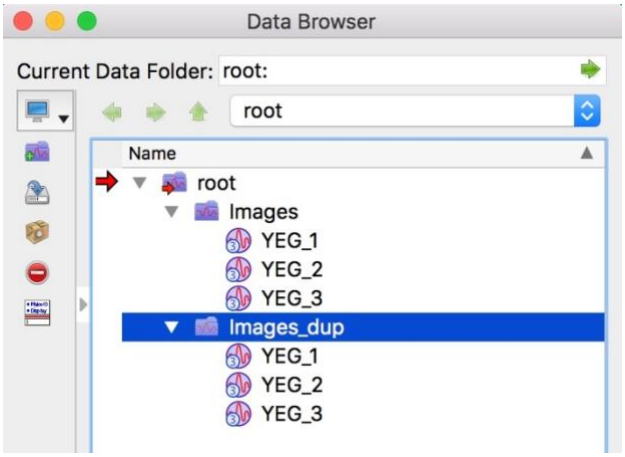


Figure 9.2: Highlighted data folder containing images to be preprocessed.

Then, we simply execute the BatchPreprocess function in the command line, with no inputs, as shown below.

```
BatchPreprocess()
```

You will be prompted for the preprocessing parameters. First is the number of standard deviations away from mean streakiness levels that are required to identify and remove a streak. We commonly use three, although what one should use (and if one should use streak removal at all) will depend upon the application – this kind of preprocessing is targeted mostly towards users of scanning probe microscopes. If you do not wish to use streak removal, enter 0. You will also be prompted for the order of the polynomial fitting for flattening. We commonly use second order flattening to account for the quadratic bowing effect seen in some atomic force

microscopy images, but again your ideal amount of flattening will be application dependent. If no flattening is desired, you may enter 0.

If you chose to perform flattening, the screen below will pop up for each image. Here you choose an appropriate height threshold to mask off particles which will interfere with the background fit. Any masked pixels will appear in blue and will be ignored in the fitting. The threshold may be adjusted via the slider bar to the right. After identifying a threshold which strikes a good balance between the features and background, hit the "Accept" button.

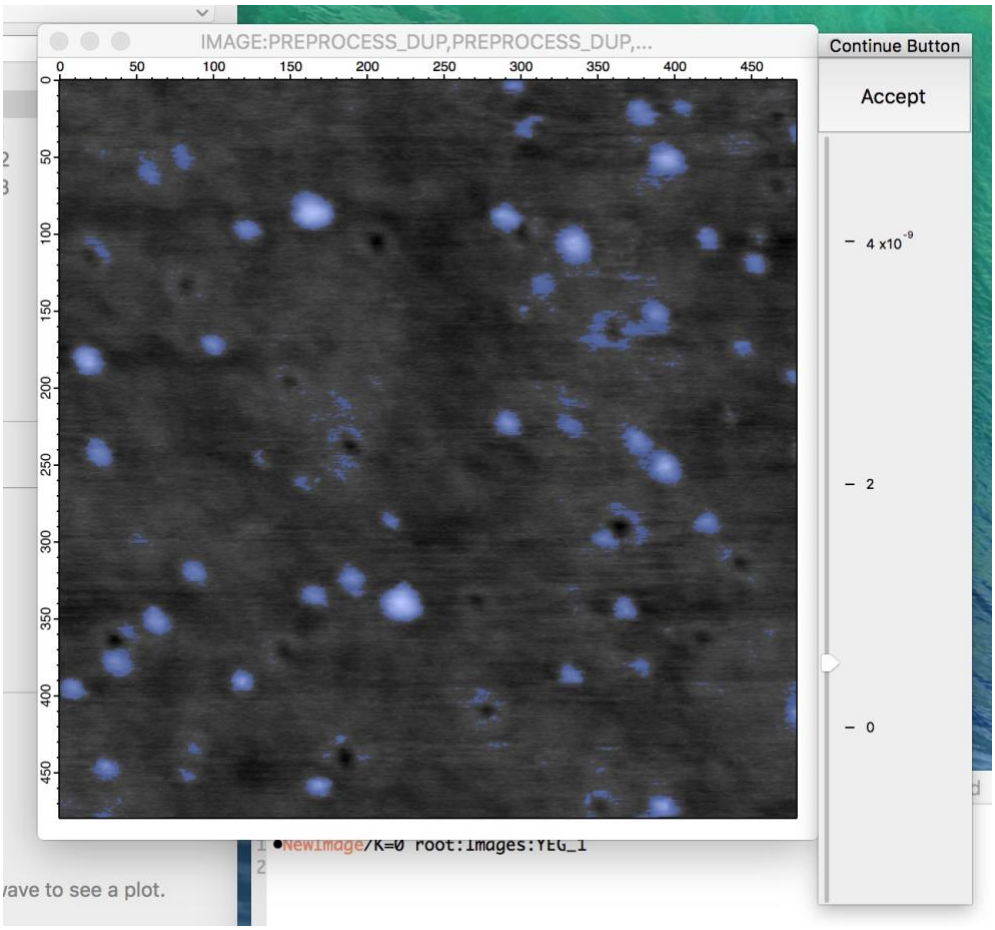


Figure 10: The flattening user interface. Here an appropriate height threshold is set for the image which masks off tall features which will interfere with the flattening process.

After the interface has been presented for each image, the preprocessing is complete. You may wish to view your images to note any changes.

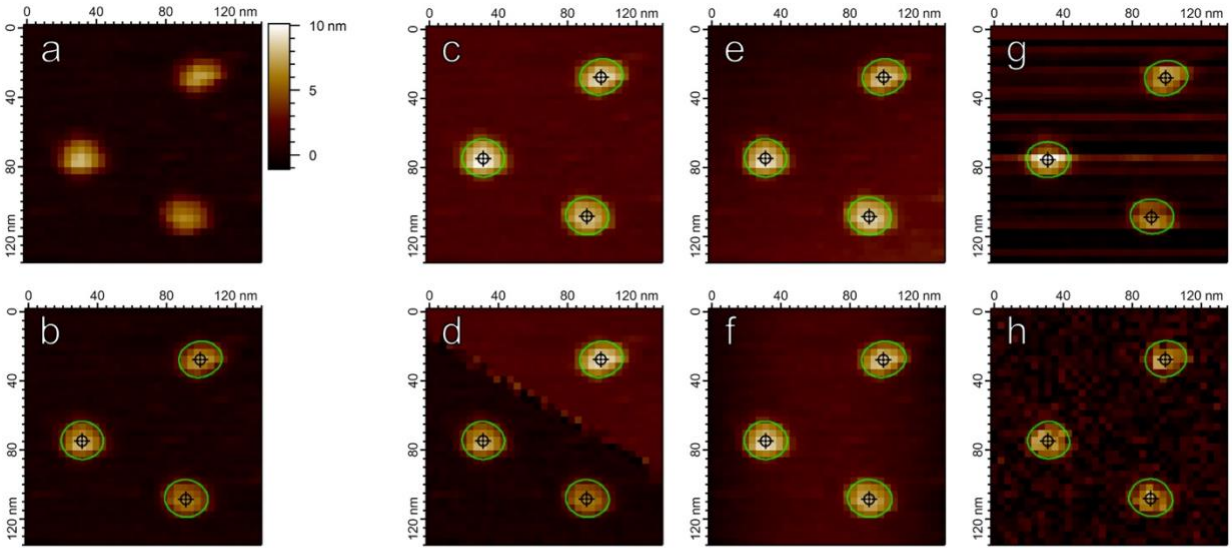
The flattening and streak removal routines may also be run individually on any image. To do so, use the following commands. Here imagePath is the full path to the image and polynomialOrder is the order of the polynomial to be fit to each row of pixels.

```
RemoveStreaks(imagePath)

Flatten(imagePath,polynomialOrder)
```

We have now covered how to apply flattening and streak removal preprocessing to a set of images, thus completing tutorial two. Congratulations once again!

Further preprocessing within Igor itself is a vast subject, and we direct the reader to the Igor manual if further processing is desired. However, note that only in drastic cases is heavy amounts of processing required. In fact, one of the key characteristics of the Hessian blob algorithm is that the algorithm is largely insensitive to common imaging artifacts, as demonstrated in the full paper. A figure from the paper is reproduced below demonstrating this point.



	c) Constant Offset	d) Discontinuity	e) Planar Offset	f) Parabolic Offset	g) Scanline Noise	h) Pixel Noise
Centerpoint Deviation	< 0.01 nm (< 0.01 pixel units)	0.03 nm (< 0.01 pixel units)	< 0.01 nm (< 0.01 pixel units)	0.01 nm (< 0.01 pixel units)	0.51 nm (0.13 pixel units)	0.46 nm (0.12 pixel units)
Perimeter Deviation	< 0.01 % (< 0.01 pixel units)	0.09 % (0.02 pixel units)	< 0.01 % (< 0.01 pixel units)	1.23 % (0.24 pixel units)	1.40 % (0.28 pixel units)	3.48 % (0.70 pixel units)
Area Deviation	< 0.01 % (< 0.01 pixel units^2)	0.04 % (0.01 pixel units^2)	0.03 % (< 0.01 pixel units^2)	2.30 % (0.64 pixel units^2)	4.35 % (1.21 pixel units^2)	6.90 % (1.94 pixel units^2)

Figure 11: Hessian blobs are stable under image defects commonly seen in AFM images. **a)** An unprocessed image of pGBP molecules. **b)** Hessian blobs discovered in (a) with minimum blob strength $1x10^{-18}$. Blob centers marked by crosshairs, boundaries calculated to 10 times pixel resolution and marked by green contours. Panels c-h demonstrate the discovered Hessian blobs after imposing image perturbations. **c) Constant Offset**: All pixels from (a) were given an addition of 2 nm. **d) Discontinuity**: A fault line was introduced, all pixels above the line were given an addition of 2 nm. **e) Tilt**: A planar tilt reaching 2 nm over the span of the image was added to (a). **f) Parabolic**: Each horizontal scan line of pixels was given a parabolic addition of up to 2 nm, mimicking the AFM bowing effect. **g) Scanline Noise**: Each scan line was given a random offset according to a Gaussian distribution of standard deviation 1 nm, mimicking scan line noise. **h) Gaussian Noise**: Gaussian noise of standard deviation 1 nm was applied to all pixels in (a). **Table)** Average changes in subpixel particle measurements after imposing image perturbations. Center points calculated to subpixel accuracy via second-order Taylor approximation. Particle perimeters and projected areas calculated via the polygon approximation as in the scanning probe microscopy analysis software Gwyddion.

III. Hessian Blob Particle Detection

In this tutorial, we apply the Hessian blob particle detection algorithm to a set of loaded images. To begin, load a set of images and place them in a data folder as covered in tutorial I. Optionally, one may then perform preprocessing as covered in tutorial II.

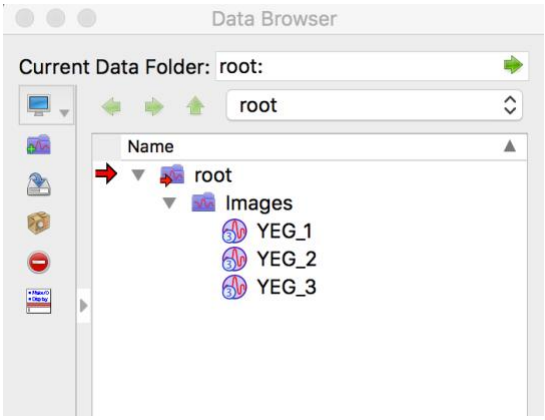


Figure 12: A set of loaded images in their own data folder.

We now run the Hessian blob algorithm on the images, but we must first identify for Igor the data folder containing our images. Highlight the data folder containing the images by single clicking on the data folder, as shown below.

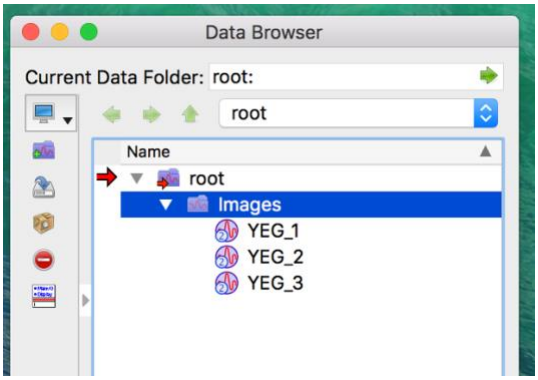


Figure 13: The highlighted data folder containing the images to be analyzed.

With the correct data folder highlighted, we now execute the BatchHessianBlobs command to run the algorithm on all of our images as follows.

```
BatchHessianBlobs()
```

Upon executing the command in the command line, the user should be prompted to input parameters for the analysis as depicted below.

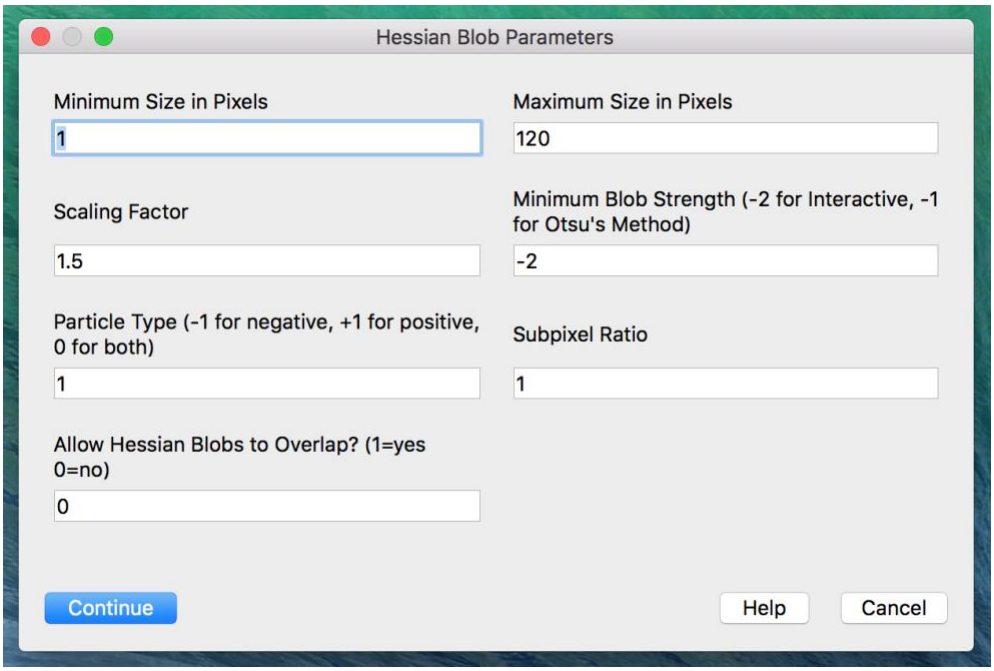


Figure 14: The parameter input screen for the Hessian blob algorithm.

Here we provide descriptions of the analysis parameters and practical direction on setting them.

Hessian Blob Parameters

1. **Minimum Size:** This parameter corresponds to the minimum size of particles that will be considered in the analysis. If the user want's to consider particles of all sizes, leave this parameter at 1 pixel. If the user only wishes to consider particles whose rough radius is at least r_{min} pixels, use r_{min} pixels as the parameter.

2. **Maximum Size:** Similarly, this parameter controls the maximum size of particles detected according to their rough radius. If the user wishes to only consider particles up to a maximum radius of r_{max} pixels, provide r_{max} as the parameter. The default setting is to consider particles up to one quarter of the size of the image in radius. Note that setting r_{max} smaller will speed up the run time.

3. **Scaling Factor:** This parameter determines how finely the scale-space representation is computed. It must be greater than 1, and the closer it is to 1 the more precise the scale-space representation is, at the cost of increased computation time. In our experience, there is little to be gained by setting the factor any lower than 1.2, but setting the factor much higher than 2 will lead to coarse grained results. The default value of 1.5 should be appropriate in most cases.

4. **Minimum Blob Strength:** This parameter provides a threshold blob strength which must be attained by a candidate Hessian blob to be considered in the analysis. Though distinct, the blob strength is related to the particle height. The user can directly enter a positive number as the minimum blob strength, provide -1 to use Otsu's method (https://en.wikipedia.org/wiki/Otsu%27s_method) to automatically determine a

threshold, or -2 to interactively set the threshold in a new dialog. Using -2 to interactively set the blob strength is the best place to start if the user is unsure.

5. **Particle Type:** Hessian blobs can detect both positive blobs (like mounds) and negative blobs (like potholes). If you wish to consider just one kind of blob, use +1 for positive blobs or -1 for negative blobs, or provide 0 to consider both kinds of blobs.

6. **Subpixel Ratio:** Hessian blobs can also be computed to subpixel precision. The subpixel ratio provides the subpixel precision as a ratio to the single pixel accuracy, e.g. a subpixel ratio of $R = 2$ corresponds to one half pixel precision, $R = 10$ to one-tenth pixel precision, and $R = 1$ as the default corresponding to single pixel precision. Note that large subpixel ratios can slow down computation time.

7. **Overlap Flag:** Hessian blobs of different sizes may overlap with one another. For most analyses, especially those where it is important not to double count particles, overlapping particles should not be permitted. By default, this parameter is set to 0 to *not* allow overlapping particles – the particle with the strong blob strength is retained and the other discarded. Otherwise, setting the parameter to 1 allows Hessian blobs to overlap.

After setting the parameters and clicking "Continue", the user will be prompted with the following dialog.

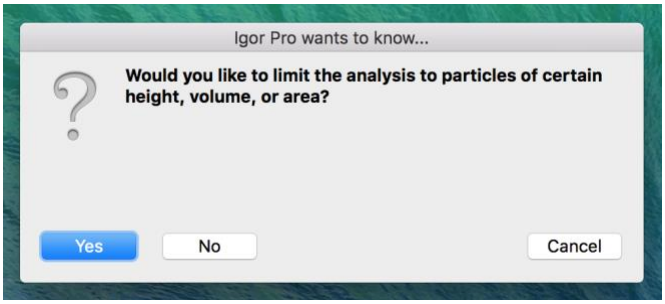


Figure 15: The contstraints dialog.

This dialog optionally allows the user to constrain the analysis to particles which fall within user-set bounds of particle height, area, or volume. This can be useful if the image is known to contain particles of certain size which the user wishes to discard from the analysis. Choosing yes opens the following dialog.

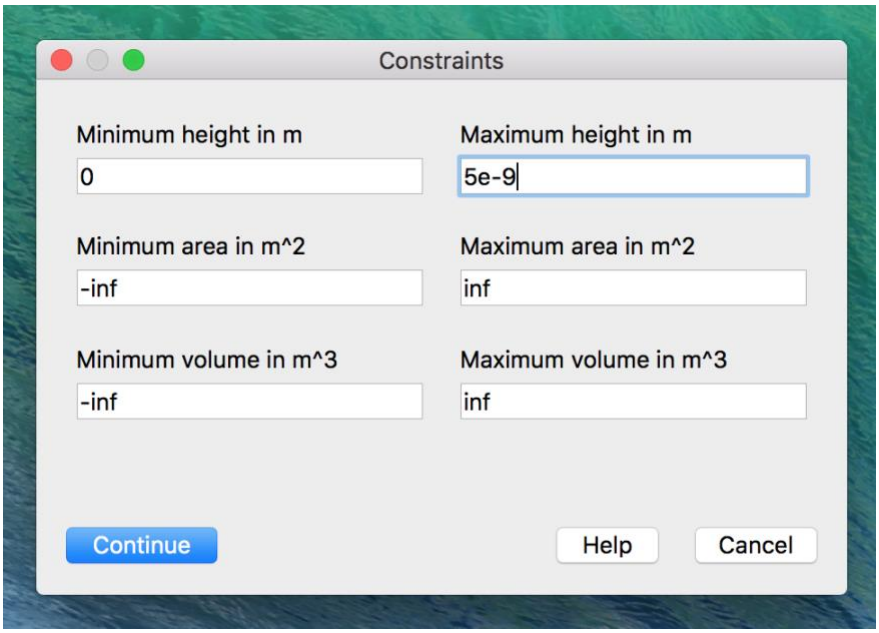


Figure 16: The optional particle constraints dialog. The default values, minimum values as -inf (minus infinity) or maximum values as inf (positive infinity), puts no constraint on the particles.

After clicking continue or skipping the particle constraints dialog, the Hessian blob algorithm will begin to run, first computing the scale-space representation of the image followed by "blob detection" images used to detect Hessian blobs.

If the blob strength parameter was set to -2 to interactively set the blob strength, the following interactive window will appear after computing the scale-space representation. This window allows the user to manually select the blob strength threshold and visually observe which blobs are considered at any given threshold. Once an appropriate blob strength threshold has been found, press the "Accept" button.

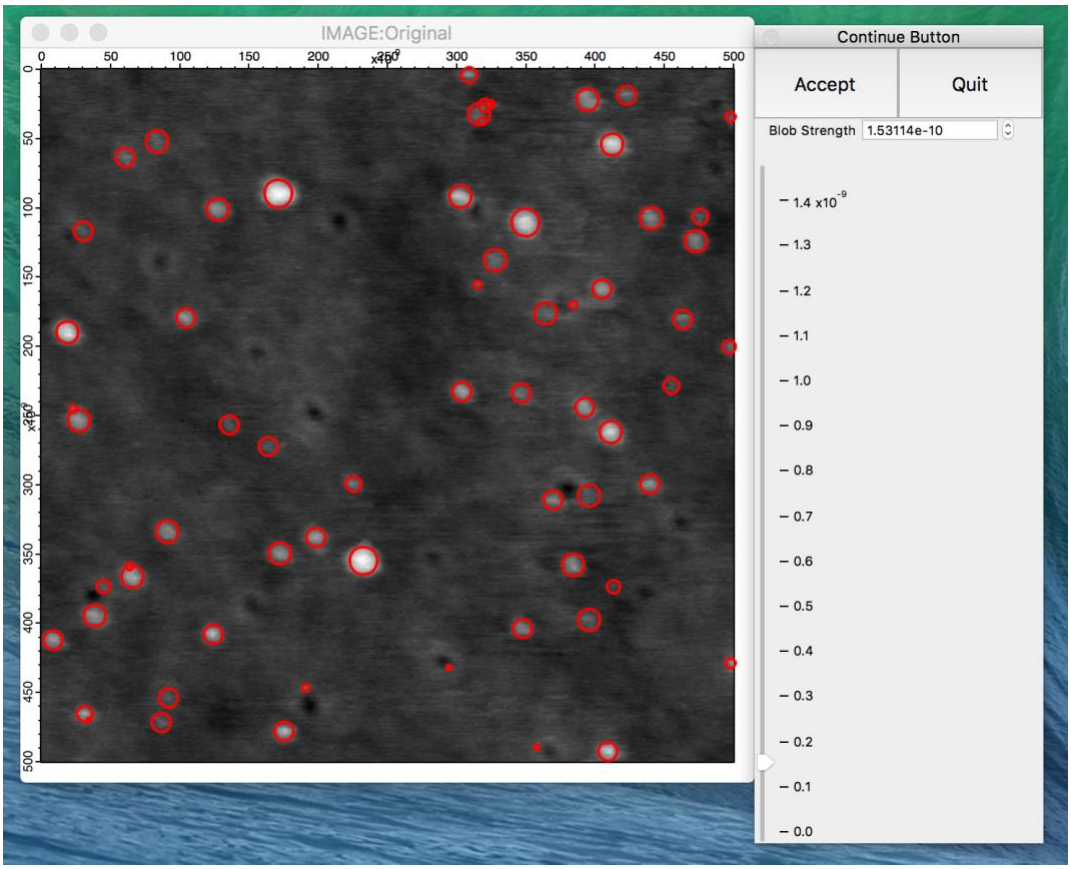


Figure 17: The interactive blob strength window. Use the slide bar to gradually change the blob strength threshold or the input text box to manually change the blob strength threshold. As the threshold is changed, the image will update and place red circles which roughly encapsulate the blobs which meet the minimum blob strength threshold.

After setting the blob strength (if the user chose to do so manually), the software will proceed to detect, isolate, and store the Hessian blobs in the image. This will be repeated for each image in the data folder. Upon completion, all of the images will be opened with the detected blobs highlighted in red.

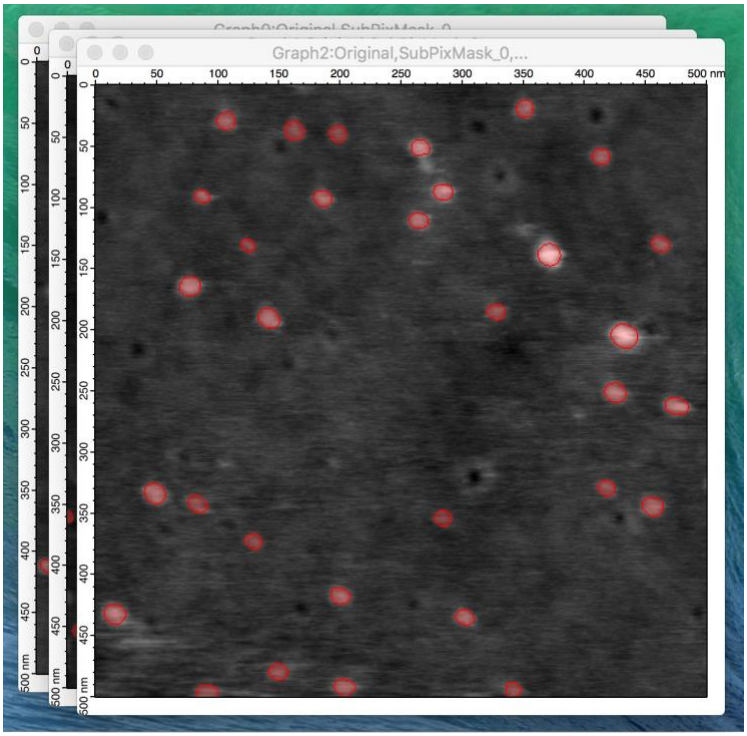


Figure 18: After the Hessian blob algorithm has run, all of the images will be displayed with detected Hessian blobs outlined in a red mask.

After completion, all individual particles, measurements, and much more information will be stored in a data folder in the data browser labelled “Series_X”, where X will be some unique number beginning at 0.

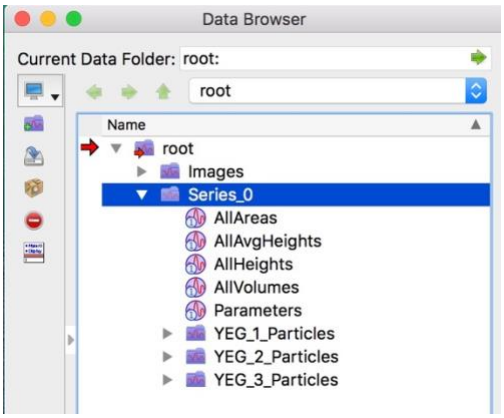


Figure 19: The Series_X data folder containing all of the results of the Hessian blobs analysis.

Inside the Series_X data folder, you will find waves containing the accumulated heights / volumes / areas of particles from all of the images. In the next tutorial we will use these waves to construct histograms summarizing the results of the analysis. Also in the Series_X data folder is a Parameters wave which contains the parameters used in the analysis. The parameters are stored as follows:

Parameters[0]	Minimum Size
Parameters[1]	Maximum Size
Parameters[2]	Scaling Factor
Parameters[3]	Blob Strength Threshold
Parameters[4]	Particle Type
Parameters[5]	Subpixel Ratio
Parameters[6]	Overlap Flag
Parameters[7]	Min Height
Parameters[8]	Max Height
Parameters[9]	Min Area
Parameters[10]	Max Area
Parameters[11]	Min Volume
Parameters[12]	Max Volume

Also located in the Series_X data folder is a data folder for each of the analyzed images, named of the form (ImageName)_Particles, which contains the results of the analysis for each individual image.

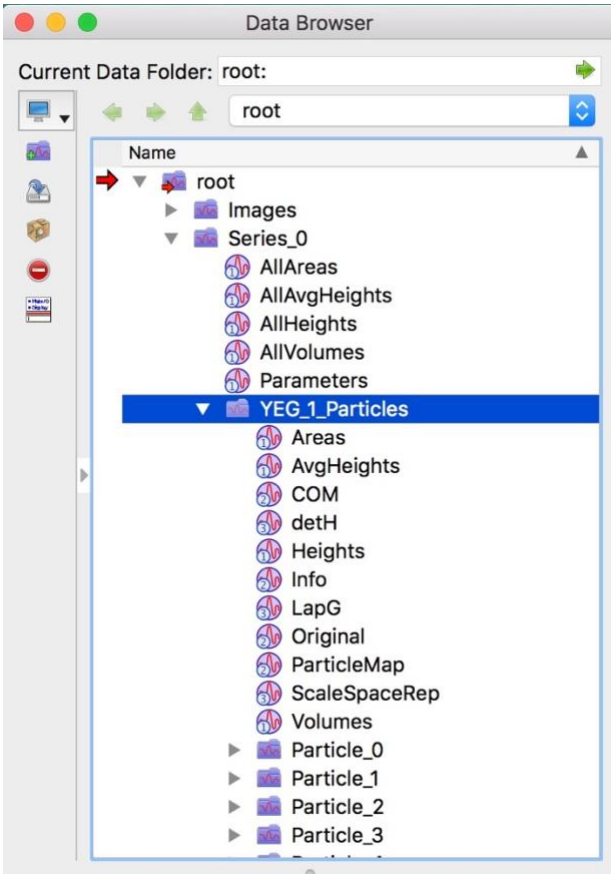


Figure 20: Looking inside an image folder within a Series_X folder.

Within an image folder, you will find waves containing the heights / areas / volumes of particles in that image, the original image, the scale-space representation of the image, a "ParticleMap" showing where particles are located, and other data.

Moreover, there is also a folder for every individual particle found within the image. For each identified particle in the image, there is an associated "Particle_X" folder, where each particle is given a unique number X.

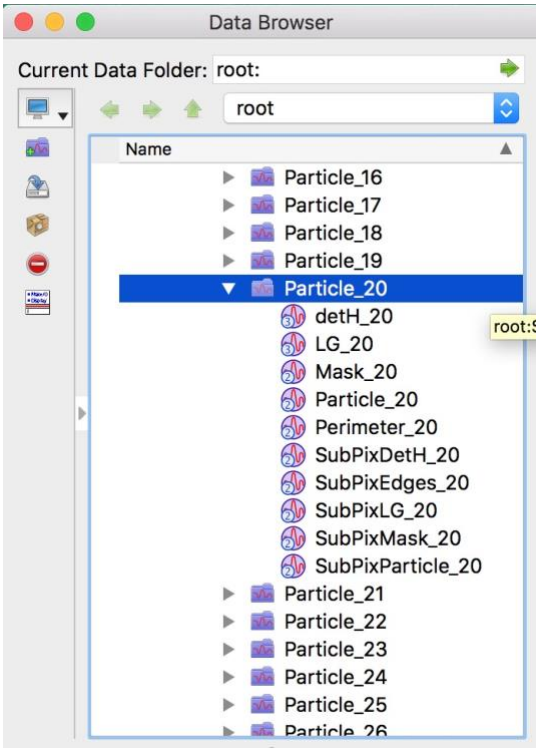


Figure 21: Looking inside a particle folder.

Inside a particle folder, there is again much data. The cropped particle itself is the Particle_X image. The Mask_X wave is a binary 0 or 1 image identifying which pixels belong to the particle and which pixels are background. The Perimeter_X wave is another binary image highlighting just the perimeter of the particle. The above three waves also have subpixel resolution equivalents, prefixed by SubPix.

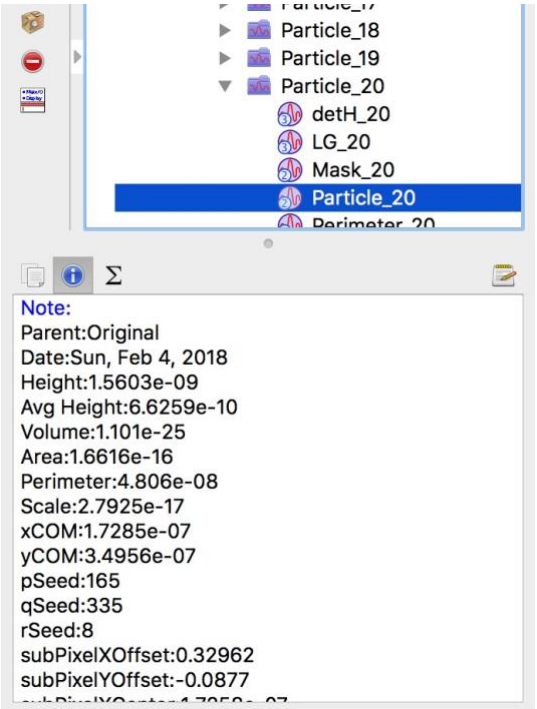


Figure 22: Additional particle information in the wave note.

Moreover, the Particle_X image contains much more information about the particle in its "wave note", which is visible in the data browser after clicking on the Particle_X image.

Included in the wave note are all of the particle's measurements, namely height, area, and volume, as well as others. The note also includes the center of mass (COM) of the particle in the image, as well as some other technical information.

A convenient method to view particles is the `ViewParticles()` function. To use it, highlight the image data folder whose particles you wish to view and run the `ViewParticles()` function in the command line.

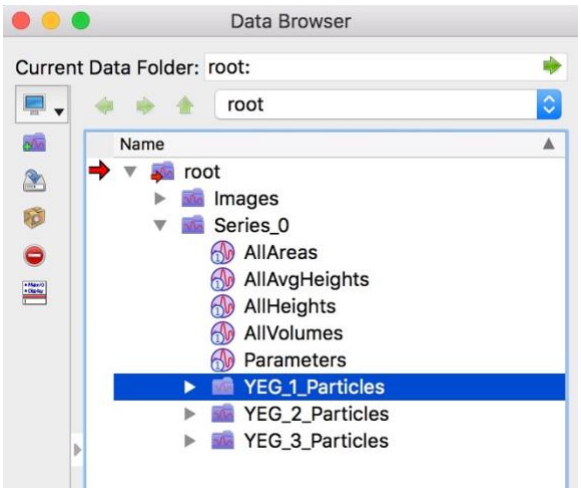


Figure 23: To view the particles in the "YEG_1" image, highlight the YEG_1_Particles data folder as shown then run the `ViewParticles()` function in the command line.

The Particle Viewer window will open, displaying the particle, some of its measurements, and various controls enabling the user to change the color scale, view the perimeter, and other options. When the "Controls" subwindow is the topmost window, the user should be able to use the arrow keys left and right to quickly navigate between particles, and use space to quickly delete a bad particle.

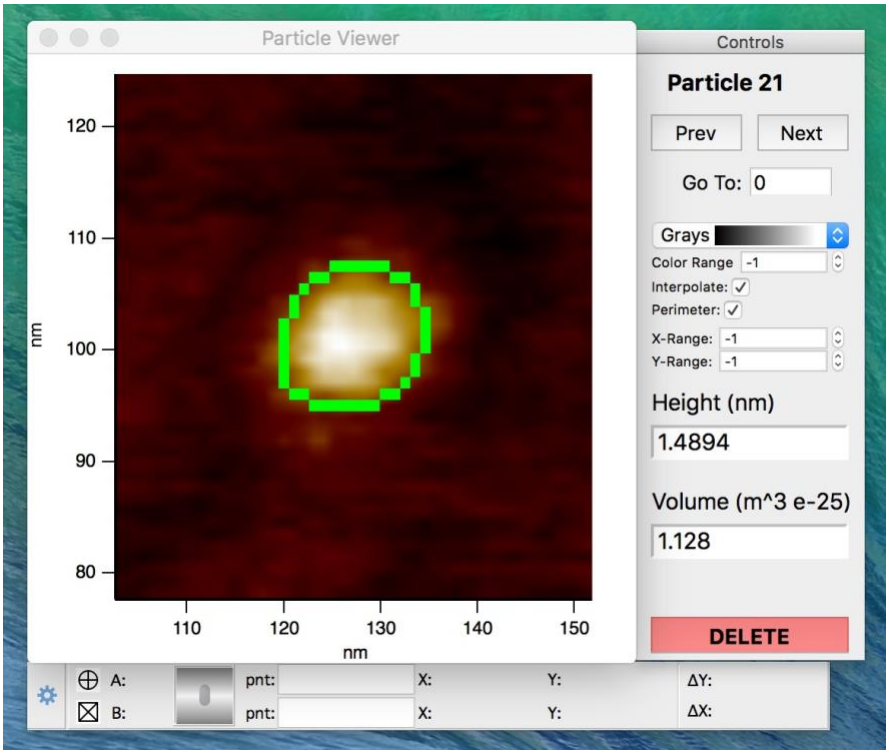


Figure 24: The Particle Viewer window.

Running the algorithm again will generate a new series data folder, so there's no need to worry about overwriting previous analyses.

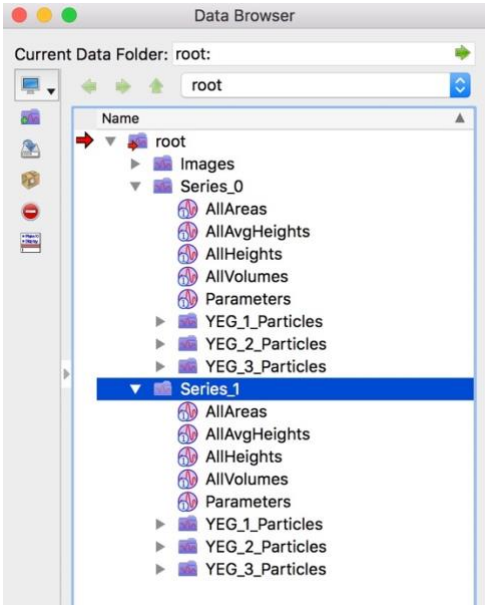


Figure 25: Rerunning the `BatchHessianBlobs()` algorithm generates a new Series_X data folder, this time Series_1.

The Hessian blob algorithm may also be executed on any single image instead of a data folder containing many images. To use, simply use the `HessianBlobs(pathToImage)` function. Now, the user must directly specify the path to the image to be analyzed as an input to the function. For example, to run the Hessian blobs algorithm on just the YEG_1 image that we loaded and placed in the Images data folder, one would execute the following command in the command line.

```
HessianBlobs(root:Images:YEG_1)
```

The user will then go through the same procedure as outlined in this tutorial, but just for a single image. Instead of a Series_X data folder being produced, only a single new image data folder will be created containing the analysis results.

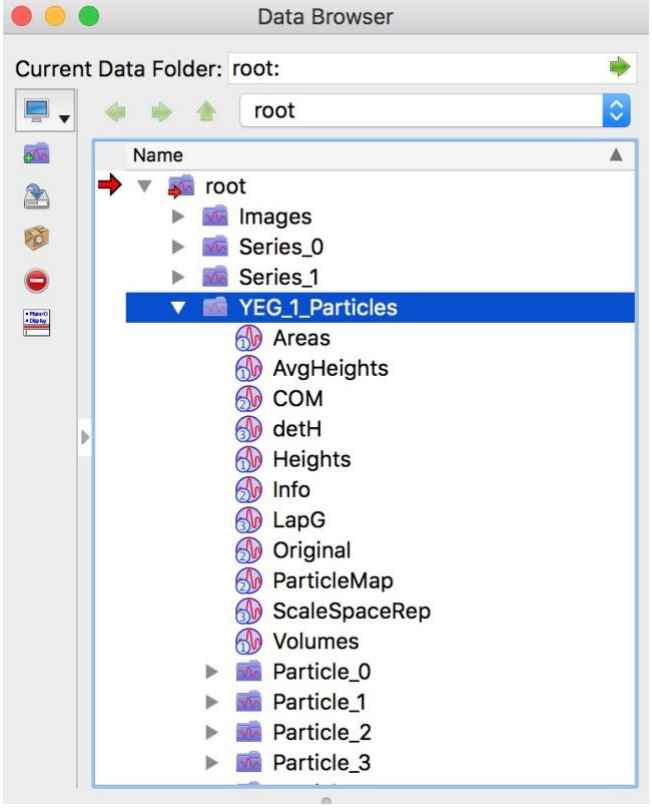


Figure 26: Running the `HessianBlobs()` algorithm on a single image, producing a new Image_Particles data folder but no new Series_X data folder.

All of the contents of the Image_Particles data folder is the same, containing the heights / areas / volumes of the particles in the image each in their own waves, as well as data folders for each particle. Using the `ViewParticles()` is the same as before, simply highlight the Image_Particles data folder in the data browser and execute the `ViewParticles()` command.

IV. Data Analysis

In this tutorial, we will perform basic data analysis on the particles detected by the Hessian blob algorithm. The end result will be histograms depicting the distributions of particle heights / areas / volumes.

To begin, run the algorithm on either a set of images or single image as in the previous tutorial. This will generate a Series_X data folder or Image_Particles data folder containing the results of the analysis.

Any wave containing the heights / areas / volumes of particles can be thought of as simply a table full of entries containing the heights / areas / volumes for each particle in the analysis. The simplest form of analysis we can do on such waves is find their average value and their standard deviation.

The easiest way to compute basic statistics of a wave in Igor is to use the `WaveStats` command. This command computes many such statistics about a given wave and prints the results in the command line. To use the `WaveStats` command, use the following syntax in the command line:

```
WaveStats (pathToWave)
```

Where `(pathToWave)` is the full data folder path to the wave you wish to compute statistics on. For example, let's try to use `WaveStats` to compute the statistics for the particle heights in the three images used in the previous tutorial. After running `BatchHessianBlobs()` on the images, the particle heights are stored in the wave AllHeights in the Series_X data folder as shown below.

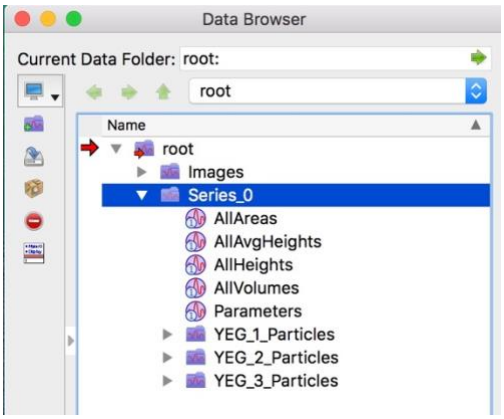


Figure 27: The path to the AllHeights wave we wish to analyze is root:Series_0:AllHeights

To use WaveStats on AllHeights, we thus use the command

```
WaveStats root:Series_0:AllHeights
```

The result in the command line after executing the above command is as follows.

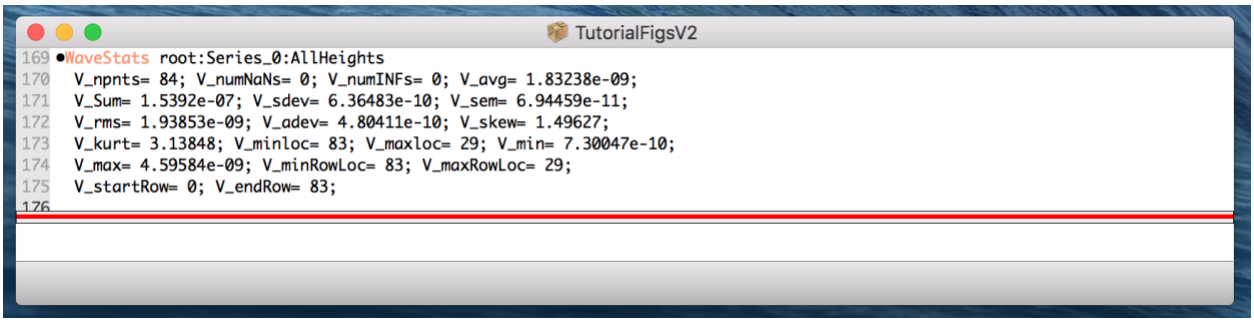


Figure 28: The results of WaveStats. The V_avg gives the average value and V_sdev gives the standard deviation.

The `WaveStats` command can of course be used for any wave, including the areas or volumes of the particles in the Series_X folder, or any of the above measurements for the particles in a single image by using `WaveStats` on the waves in the Image_Particles folders.

A more graphical way to represent the distributions of heights / areas / volumes is via a histogram. Igor has built-in histogram generating tools, available in the Analysis tab as Analysis->Histogram.

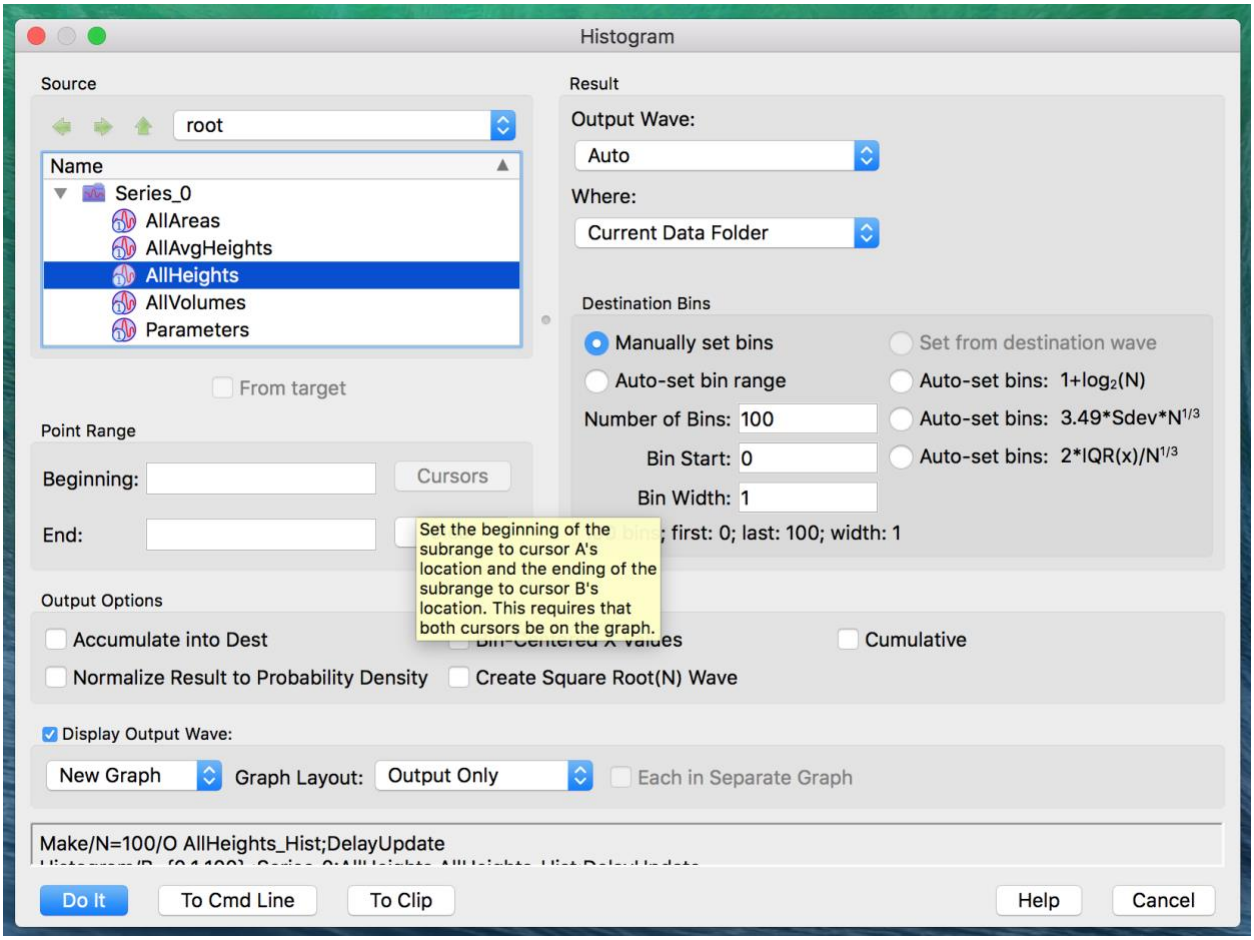


Figure 29: Igor's built-in histogram tool. We will defer to the Igor Pro manual for the specifics of the histogram tool: http://www.wavemetrics.net/doc/igorman/IgorMan.pdf

Using ten bins and a bin width of 0.5nm, we construct the following histogram of heights in the loaded images. The histogram will be constructed as a new wave in the data browser, with a name of the type `(waveName)_Hist`.

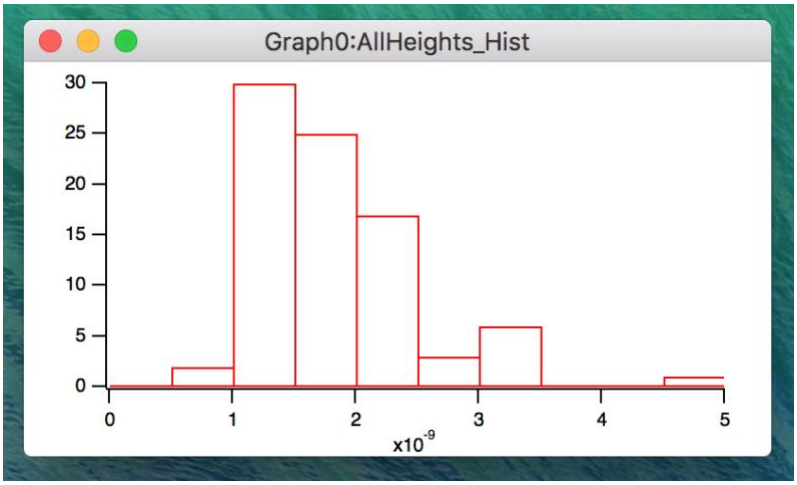


Figure 30: A histogram demonstrating the distribution of particle heights (in nanometers) detected in the loaded atomic force microscopy images.

A histogram can of course be generated for any of the particle measurements, providing a nice visual way of presenting the results of the analysis.