**Automated analysis of scanning electron microscopic images for assessment of hair surface damage**

Fanny Chu1,2\*, Deon S. Anex1, A. Daniel Jones3, Bradley R. Hart1

1Lawrence Livermore National Laboratory, 7000 East Ave. Livermore, CA 94550

2Department of Chemistry, Michigan State University, 578 S Shaw Ln East Lansing, MI 48824

3Department of Biochemistry and Molecular Biology, 603 Wilson Rd. East Lansing, MI 48824

\***Corresponding Author**

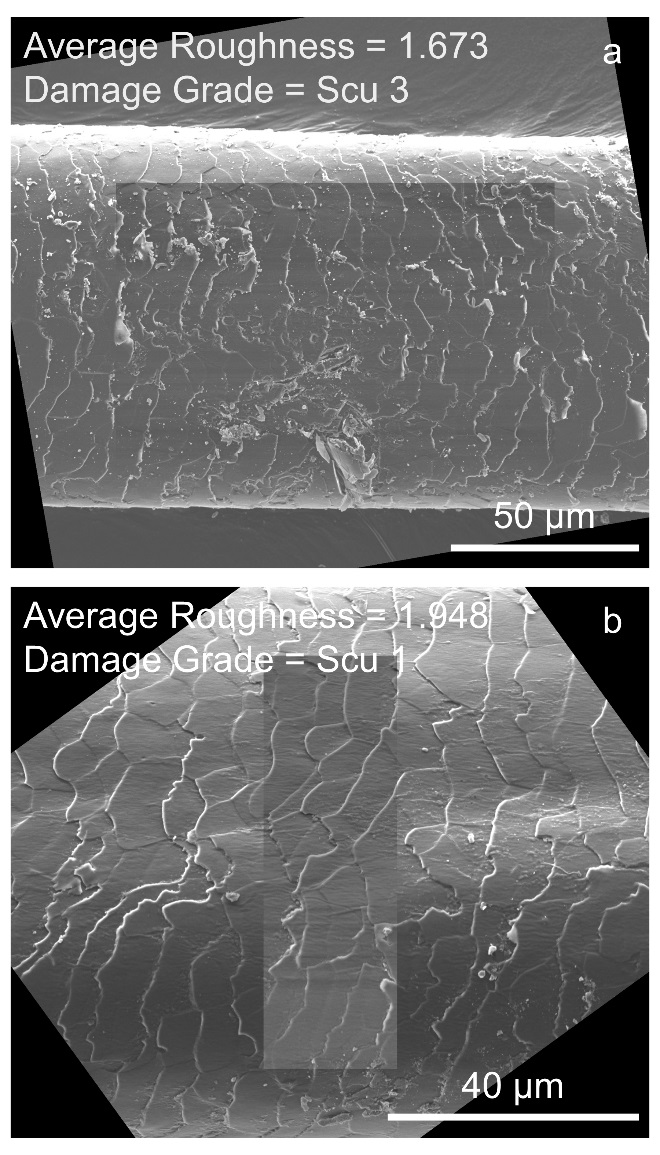
Fanny Chu

7000 East Ave. L-173

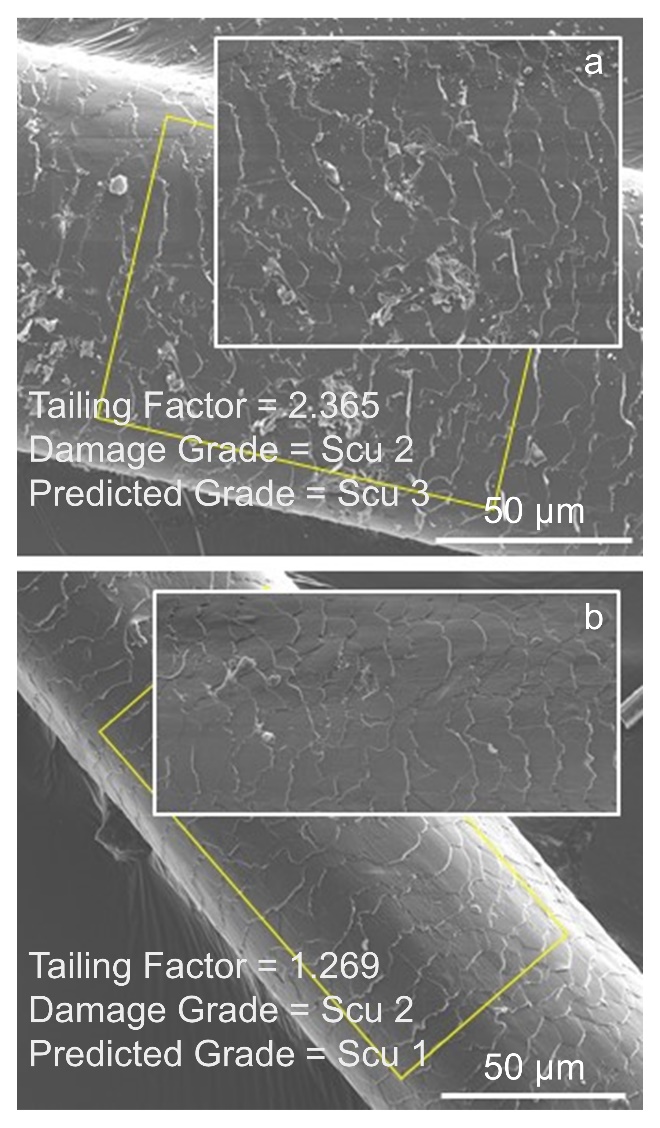
Livermore, CA 94550

925.423.0684

chu28@llnl.gov

****

**Figure S1.** Example rotated SEM images with overlays of normalized regions of interest from (a) Hair Sample 1 and (b) Hair Sample 4 that exhibit similar average image roughness but vastly different extents of damage, as assessed with the SEM damage grade system. Average image roughness fails to effectively characterize hair surface damage.



**Figure S2.** Example SEM images of hair segments with normalized regions of interest from (a) Hair Sample 1 (exploded) and (b) Hair Sample 5 (control) whose damage grades were incorrectly predicted by the kNN model.

**Appendix 1.** ImageJ macros (for version 1.52k) to normalize SEM images of hair fibers and quantify hair surface damage using tailing factor.

Hair Damage Quantification Menu Tool:

1. Duplicate and Crop Action Tool – duplicates raw image input (TIFF file) and crops out SEM image acquisition parameter label

2. Measure Angle and Rotate Action Tool – measures angle of hair segment with respect to transverse axis and rotates image so that hair segment lies along the transverse axis

3. Normalize Image Action Tool – defines region of interest (ROI) according to Eqs. 1 – 3 and normalizes ROI according to Eqs. 4 and 5

4. Calculate Histogram Peak Tailing Action Tool – calculates peak lag tailing from brightness histogram of normalized ROI at 2% of peak height maximum according to Eqs. 10 – 12

//series of macros containing duplicate and crop (to remove SEM image label), measure angle and rotate (to orient

//hair length to lie along the transverse axis), normalize image, and calculate peak tailing factor (to quantify hair

//surface damage)

var sCmds = newMenu("Hair Damage Quantification Menu Tool", newArray("Duplicate and Crop Action Tool", "Measure Angle and Rotate Action Tool", "-", "Normalize Image Action Tool", "Calculate Histogram Peak Tailing Action Tool"));

//menu macro to run selected macro

macro "Hair Damage Quantification Menu Tool - Cf00T1a12HC2a0T4c12DC000T7g12Q" {

cmd = getArgument();

if (cmd!="-")

run(cmd);

}

//duplicate and crop macro

//requires raw image input to be TIFF file

//duplicates SEM image and crops label after identifying where label begins (a horizontal line composed entirely of //white pixels at 255)

macro "Duplicate and Crop Action Tool" {

orig\_name = getTitle();

//duplicates image

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

new\_name = replace(orig\_name, ".tif", "\_ImageJ.tif");

//renames duplicated image

rename(new\_name);

//gets dimensions of image in pixels

image\_length = getWidth(); //hair segment length along transverse axis

image\_height = getHeight(); //hair segment width along vertical axis

//scans image line by line horizontally to find start of label

for (j=0; j<image\_height; j++) {

makeLine(0, j, image\_length, j);

profile = getProfile();

max\_value = 0;

min\_value = 255;

for (i=0; i<profile.length; i++) {

if (max\_value<profile[i]) {

max\_value = profile[i];

}

if (min\_value>profile[i]) {

min\_value = profile[i];

}

}

//stops scanning when finds start of label

if (max\_value==255 && min\_value==255) {

break;

}

}

//crops image to exclude label

makeRectangle(0, 0, image\_length, j);

run("Crop");

selectWindow(new\_name);

}

var param\_table = "Image Parameters";

//measure angle and rotate macro; adapted from “Measure\_Angle\_And\_Length.txt” macro

//requires line input from user - use Line Tool to draw line segment along edge of hair shaft

//measures angle of user-defined line segment with transverse axis and rotates image so hair shaft length lies along

//transverse axis

macro "Measure Angle and Rotate Action Tool" {

Roi.remove;

//pops up dialog box to direct user

waitForUser("Draw line along edge of hair shaft. Click ok when done");

//extracts line coordinates

getLine(x1, y1, x2, y2, lineWidth);

//if Line Tool not selected or no line input, exits out of macro

if (selectionType!=5 || x1==-1)

exit("Straight line selection along edge of hair shaft required");

getPixelSize(unit, length, height, depth);

x1\*=length; y1\*=height; x2\*=length; y2\*=height;

//finds angle of line with transverse axis using formula below

angle = getAngle(x1, y1, x2, y2);

length = sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1));

//creates image parameters table

Table.create("Image Parameters");

param\_table = Table.title;

//outputs angle and line length measurements

Table.set("Parameter", Table.size, "Image Name", param\_table);

Table.set("Parameter Value", Table.size-1, getTitle(), param\_table);

Table.set("Parameter", Table.size, "Angle", param\_table);

Table.set("Parameter Value", Table.size-1, angle, param\_table);

Table.set("Parameter", Table.size, "Line Length", param\_table);

Table.set("Parameter Value", Table.size-1, length, param\_table);

dx = x2-x1;

dy = y1-y2;

//calculates angle of rotation

if (dx<0)

angle\_rot=angle-180.0;

else if (dx>=0 && dy>=0)

angle\_rot=angle;

else

angle\_rot=angle-360.0;

Table.set("Parameter", Table.size, "Angle of Rotation", param\_table);

Table.set("Parameter Value", Table.size-1, angle\_rot, param\_table);

//rotates image based on angle of rotation

run("Rotate...", "angle=angle\_rot grid=1 interpolation=Bilinear stack");

Table.update;

}

//formula to calculate angle (from “Measure\_Angle\_And\_Length.txt” macro)

function getAngle(x1, y1, x2, y2) {

q1=0; q2orq3=2; q4=3; //quadrant

dx = x2-x1;

dy = y1-y2;

if (dx!=0)

angle = atan(dy/dx);

else {

if (dy>=0)

angle = PI/2;

else

angle = -PI/2;

}

angle = (180/PI)\*angle;

if (dx>=0 && dy>=0)

quadrant = q1;

else if (dx<0)

quadrant = q2orq3;

else

quadrant = q4;

if (quadrant==q2orq3)

angle = angle+180.0;

else if (quadrant==q4)

angle = angle+360.0;

return angle;

}

//normalize image macro

//requires line input from user - use Line Tool to draw diagonal line spanning two corners of the hair segment

//defines region of interest, centered using 75% of length and width of hair segment as defined by diagonal line (Eqs.

//1 - 3)

//normalizes each pixel to average brightness along hair length and rescales average brightness to 109 (Eq. 4)

//rescales normalized pixel to be bounded between 109 and 255 if normalized value is greater than 109 (Eq. 5)

var param\_table = Table.title;

macro "Normalize Image Action Tool" {

Roi.remove;

updateDisplay();

//pops up dialog box to direct user

waitForUser("Draw diagonal line from edge of one end of hair shaft to edge of other end. Click ok when done");

//extracts line coordinates

getLine(x1, y1, x2, y2, lineWidth);

//exits out of macro if Line Tool not selected or no line input

if(selectionType!=5 || x1==-1)

exit("Straight line selection along hair shaft diagonal required");

image\_id = getImageID();

//extracts length and width of hair segment

image\_length = abs(x2-x1);

image\_height = abs(y2-y1);

//finds center of image

image\_xcenter =(x1+x2)/2;

image\_ycenter = (y1+y2)/2;

if (x1<x2) {

x\_min = x1;

x\_max = x2;

} else {

x\_min = x2;

x\_max = x1;

}

if (y1<y2) {

y\_min = y1;

y\_max = y2;

} else {

y\_min = y2;

y\_max = y1;

}

//outputs hair segment parameters

selectWindow(param\_table);

Table.set("Parameter", Table.size, "Image Center\_x", param\_table);

Table.set("Parameter Value", Table.size-1, image\_xcenter, param\_table);

Table.set("Parameter", Table.size, "Image Center\_y", param\_table);

Table.set("Parameter Value", Table.size-1, image\_ycenter, param\_table);

Table.set("Parameter", Table.size, "Segment Length", param\_table);

Table.set("Parameter Value", Table.size-1, image\_length, param\_table);

Table.set("Parameter", Table.size, "Segment Height", param\_table);

Table.set("Parameter Value", Table.size-1, image\_height, param\_table);

//defines upper lefthand corner, length, and width of ROI (Eqs. 1 - 3)

rect\_x = round(((x\_min)/2)+((x\_min)/4)+(x\_min+x\_max)/8);

rect\_y = round(((y\_min)/2)+((y\_min)/4)+(y\_max+y\_min)/8);

rect\_length = round((((x\_max)/2)+((x\_max)/4)+(x\_min+x\_max)/8))-rect\_x;

rect\_height = round((((y\_max)/2)+((y\_max)/4)+(y\_max+y\_min)/8))-rect\_y;

Table.set("Parameter", Table.size, "ROI Length", param\_table);

Table.set("Parameter Value", Table.size-1, rect\_length, param\_table);

Table.set("Parameter", Table.size, "ROI Height", param\_table);

Table.set("Parameter Value", Table.size-1, rect\_height, param\_table);

Table.set("Parameter", Table.size, "ROI Upper LH\_x", param\_table);

Table.set("Parameter Value", Table.size-1, rect\_x, param\_table);

Table.set("Parameter", Table.size, "ROI Upper LH\_y", param\_table);

Table.set("Parameter Value", Table.size-1, rect\_y, param\_table);

//scans through raw ROI to obtain average brightness per line

j = rect\_y;

m = rect\_x;

k = 0;

profile\_avg = newArray(rect\_height+1);

while (j<=(rect\_y+rect\_height)) {

makeLine(rect\_x, j, rect\_x+rect\_length, j);

profile = getProfile();

profile\_sum = 0;

m = rect\_x;

while (m<(rect\_x+rect\_length)) {

m=m+1;

profile\_sum += profile[m-rect\_x];

}

profile\_avg[k] = profile\_sum/profile.length;

profile\_sum\_ROI += profile\_sum;

j=j+1;

k=k+1;

}

//scans through raw ROI to obtain maximum brightness value after normalizing to average line brightness

//and rescaling average to 109 (Eq. 4)

j = rect\_y;

k = 0; //row\_index

i = 0; //col\_index

max\_value = 0;

profile\_normval = newArray(rect\_length+1);

while (j<=(rect\_y+rect\_height)) {

profile\_normval = newArray(rect\_length+1);

makeLine(rect\_x, j, rect\_x+rect\_length, j);

profile = getProfile();

m = rect\_x;

i = 0;

while (m<(rect\_x+rect\_length)) {

profile\_normval[i] = (profile[i]/profile\_avg[k])\*109;

if (profile\_normval[i]>max\_value) {

max\_value=profile\_normval[i];

}

m=m+1;

i=i+1;

}

j=j+1;

k=k+1;

}

//scans through raw ROI to apply normalization and rescaling to raw values (Eq. 4 and 5)

run("Clear Results");

j = rect\_y;

k = 0; //row\_index

i = 0; //col\_index

profile\_newval = newArray(rect\_length+1);

profile\_newval\_avg = newArray(rect\_height+1);

while (j<=(rect\_y+rect\_height)) {

profile\_newval = newArray(rect\_length+1);

makeLine(rect\_x, j, rect\_x+rect\_length, j);

profile = getProfile();

m = rect\_x;

i = 0;

profile\_newval\_sum = 0;

while (m<(rect\_x+rect\_length)) {

if ((profile[i]/profile\_avg[k])\*109>109) {

profile\_newval[i] = 109+(146\*(((profile[i]/profile\_avg[k])\*109)-109)/(max\_value-109));

} else {

profile\_newval[i] = (profile[i]/profile\_avg[k])\*109;

}

//generates array of normalized brightness values in Results Table (separate from

//param\_table)

setResult(m, k, profile\_newval[i]);

profile\_newval\_sum += profile\_newval[i];

m=m+1;

i=i+1;

}

profile\_newval\_avg[k] = profile\_newval\_sum/profile.length;

profile\_newval\_sum\_ROI += profile\_newval\_sum;

j=j+1;

k=k+1;

}

updateResults();

//sets brightness bounds between 0 and 255 in normalized ROI by temporarily converting 2 pixels to 0 and

//255, respectively

black\_pix\_x = rect\_x;

white\_pix\_x = rect\_x+1;

black\_pix\_orig = getResult(black\_pix\_x, 0);

white\_pix\_orig = getResult(white\_pix\_x, 0);

setResult(black\_pix\_x, 0, 0);

setResult(white\_pix\_x, 0, 255);

updateResults();

selectWindow("Results");

//converts array of normalized brightness values to 32-bit grayscale image

run("Results to Image");

//resets 2 pixels to normalized brightness values

setPixel(black\_pix\_x-rect\_x, 0, black\_pix\_orig);

setPixel(white\_pix\_x-rect\_x, 0, white\_pix\_orig);

updateDisplay();

setResult(black\_pix\_x, 0, black\_pix\_orig);

setResult(white\_pix\_x, 0, white\_pix\_orig);

updateResults();

IJ.renameResults("Normalized ROI Array");

run("Clear Results");

//calculates raw and normalized ROI average % brightness

profile\_sum\_ROI\_avg = profile\_sum\_ROI/(profile.length\*profile\_avg.length);

profile\_sum\_ROI\_avg\_percent = (profile\_sum\_ROI\_avg/255)\*100;

profile\_newval\_sum\_ROI\_avg = profile\_newval\_sum\_ROI/(profile\_newval\_avg.length\*profile.length);

profile\_newval\_sum\_ROI\_avg\_percent = (profile\_newval\_sum\_ROI\_avg/255)\*100;

//outputs raw and normalized ROI average % brightness

selectWindow(param\_table);

Table.set("Parameter", Table.size, "Raw Average % Brightness", param\_table);

Table.set("Parameter Value", Table.size-1, profile\_sum\_ROI\_avg\_percent, param\_table);

Table.set("Parameter", Table.size, "Normalized Average % Brightness", param\_table);

Table.set("Parameter Value", Table.size-1, profile\_newval\_sum\_ROI\_avg\_percent, param\_table);

Table.update;

close("Normalized ROI Array");

selectWindow("Results Table");

rename("Normalized ROI");

}

//calculate histogram peak tailing macro

//defines peak apex to reduce histogram skew in event of multiple peaks (Eqs. 10 and 11)

//calculates tailing factor at 2% of peak height maximum (Eq. 12)

macro "Calculate Histogram Peak Tailing Action Tool" {

selectWindow("Normalized ROI");

//generates histogram of normalized ROI to find peak height maximum H and average brightness value

q = 0;

nBins = 256;

peak\_height\_max = 0;

average\_brightness\_sum = 0;

pixel\_count = 0;

getHistogram(values, counts, nBins, 0, 255);

for (q=0; q<nBins; q++) {

if (counts[q]>peak\_height\_max) {

peak\_height\_max=counts[q];

brightness\_max=q;

}

average\_brightness\_sum+=(counts[q]\*q);

pixel\_count+=counts[q];

}

average\_brightness = average\_brightness\_sum/pixel\_count;

//calculates peak lead and lag brightness values at full width half maximum for application to Eq. 10

//in event that brightness values are not integers, exact values are calculated using equation of a line from

//consecutive brightness values that bound the height at half maximum

height\_halfmax = 0.5\*peak\_height\_max;

for (r=255; r>brightness\_max; r--) {

if ((counts[r]<=height\_halfmax) && (counts[r-1]>=height\_halfmax)) {

halfmax\_lag\_brightness = r;

halfmax\_lag\_brightness2 = r-1;

break;

}

}

slope = (counts[halfmax\_lag\_brightness]-counts[halfmax\_lag\_brightness2])/(1);

intercept = counts[halfmax\_lag\_brightness]-(slope\*(halfmax\_lag\_brightness));

halfmax\_lag\_brightness\_exact = (height\_halfmax-intercept)/slope;

for (r=0; r<brightness\_max; r++) {

if ((counts[r]<=height\_halfmax) && (counts[r+1]>=height\_halfmax)) {

halfmax\_lead\_brightness = r;

halfmax\_lead\_brightness2 = r+1;

break;

}

}

slope = (counts[halfmax\_lead\_brightness2]-counts[halfmax\_lead\_brightness])/(1);

intercept = counts[halfmax\_lead\_brightness2]-(slope\*(halfmax\_lead\_brightness2));

halfmax\_lead\_brightness\_exact = (height\_halfmax-intercept)/slope;

//calculates averge brightness value between peak lead and lag at full width half maximum (Eq. 10)

k = round(halfmax\_lead\_brightness\_exact);

average\_brightness\_halfmax\_sum = 0;

pixel\_count\_halfmax\_width = 0;

while (k<=round(halfmax\_lag\_brightness\_exact)) {

average\_brightness\_halfmax\_sum+=(counts[k]\*k);

pixel\_count\_halfmax\_width+=counts[k];

k=k+1;

}

average\_brightness\_halfmax = average\_brightness\_halfmax\_sum/pixel\_count\_halfmax\_width;

//redefines peak apex by comparing brightness at peak height maximum to average brightness when

//calculated between full width half maximum (Eq. 11)

selectWindow(param\_table);

if (brightness\_max<=(average\_brightness\_halfmax+3) && brightness\_max>=(average\_brightness\_halfmax-3)) {

Table.set("Parameter", Table.size, "Brightness\_Peak Height Maximum", param\_table);

Table.set("Parameter Value", Table.size-1, brightness\_max, param\_table);

Table.set("Parameter", Table.size, "Brightness\_Peak Apex", param\_table);

Table.set("Parameter Value", Table.size-1, brightness\_max, param\_table);

} else {

Table.set("Parameter", Table.size, "Brightness\_Peak Height Maximum", param\_table);

Table.set("Parameter Value", Table.size-1, brightness\_max, param\_table);

brightness\_max = round(average\_brightness\_halfmax);

Table.set("Parameter", Table.size, "Brightness\_Peak Apex", param\_table);

Table.set("Parameter Value", Table.size-1, brightness\_max, param\_table);

}

//calculates peak lead and lag brightness values at 0.02H for application to Eq. 12

//in event that brightness values are not integers, exact values are calculated using equation of a line from

//consecutive brightness values that bound height at 0.02H

height\_tailing = 0.02\*peak\_height\_max;

for (r=255; r>brightness\_max; r--) {

if ((counts[r]<=height\_tailing) && (counts[r-1]>=height\_tailing)) {

tailing\_lag\_brightness = r;

tailing\_lag\_brightness2 = r-1;

break;

}

}

slope = (counts[tailing\_lag\_brightness]-counts[tailing\_lag\_brightness2])/(1);

intercept = counts[tailing\_lag\_brightness]-(slope\*(tailing\_lag\_brightness));

tailing\_lag\_brightness\_exact = (height\_tailing-intercept)/slope;

for (r=0; r<brightness\_max; r++) {

if ((counts[r]<=height\_tailing) && (counts[r+1]>=height\_tailing)) {

tailing\_lead\_brightness = r;

tailing\_lead\_brightness2 = r+1;

break;

}

}

slope = (counts[tailing\_lead\_brightness2]-counts[tailing\_lead\_brightness])/(1);

intercept = counts[tailing\_lead\_brightness2]-(slope\*(tailing\_lead\_brightness2));

tailing\_lead\_brightness\_exact = (height\_tailing-intercept)/slope;

//calculates tailing factor at 0.02H (Eq. 12)

tailing\_factor = (tailing\_lag\_brightness-tailing\_lead\_brightness)/(2\*(brightness\_max-tailing\_lead\_brightness));

//outputs peak lag tailing parameters

selectWindow(param\_table);

Table.set("Parameter", Table.size, "Peak Height Maximum", param\_table);

Table.set("Parameter Value", Table.size-1, peak\_height\_max, param\_table);

Table.set("Parameter", Table.size, "Tailing\_Peak Lead Brightness", param\_table);

Table.set("Parameter Value", Table.size-1, tailing\_lead\_brightness\_exact, param\_table);

Table.set("Parameter", Table.size, "Tailing\_Peak Lag Brightness", param\_table);

Table.set("Parameter Value", Table.size-1, tailing\_lag\_brightness\_exact, param\_table);

Table.set("Parameter", Table.size, "Tailing Factor\_0.02H", param\_table);

Table.set("Parameter Value", Table.size-1, tailing\_factor, param\_table);

Table.update;

}

**Appendix 2.** ImageJ code (for version 1.52k) to quantify hair surface damage using image roughness (in lieu of macro 4 in Appendix 1).

//finds average roughness of image where roughness across the normalized ROI is binned into pixel-by-pixel, 10, 20,

//50, or 100 sections

//performs calculations after normalizing ROI (in lieu of calculating tailing factor)

var param\_table = "Image Parameters";

selectWindow("Normalized ROI");

rect\_length = getWidth();

rect\_height = getHeight()-1;

//determines number of pixels binned to each section (section width) according to Eq. 7

sect10\_length = floor(rect\_length/10)+1;

sect20\_length = floor(rect\_length/20)+1;

sect50\_length = floor(rect\_length/50)+1;

sect100\_length = floor(rect\_length/100)+1;

j = 0; //height index

k = 0; //row\_index

p = 0; //col\_index

roughness\_pixpix\_array = newArray(rect\_height+1);

roughness\_pixpix\_sum = 0;

roughness\_sect100\_array = newArray(rect\_height+1);

roughness\_sect100\_sum = 0;

roughness\_sect50\_array = newArray(rect\_height+1);

roughness\_sect50\_sum = 0;

roughness\_sect20\_array = newArray(rect\_height+1);

roughness\_sect20\_sum = 0;

roughness\_sect10\_array = newArray(rect\_height+1);

roughness\_sect10\_sum = 0;

//calculates roughness across each row of the image according to Eq. 9

while (j<=(rect\_height)) {

makeLine(0, j, rect\_length, j);

profile = getProfile();

m = 0; //length index

p = 0;

r = 0; //index for distance\_pixpix\_array

u = 1; //index for distance\_sect100\_array

v = 1; //index for distance\_sect50\_array

w = 1; //index for distance\_sect20\_array

z = 1; //index for distance\_sect10\_array

distance\_pixpix\_sum = 0;

distance\_pixpix\_array = newArray(rect\_length);

distance\_sect10\_sum = 0;

distance\_sect10\_array = newArray((floor(rect\_length/sect10\_length)+1));

distance\_sect20\_sum = 0;

distance\_sect20\_array = newArray((floor(rect\_length/sect20\_length)+1));

distance\_sect50\_sum = 0;

distance\_sect50\_array = newArray((floor(rect\_length/sect50\_length)+1));

distance\_sect100\_sum = 0;

distance\_sect100\_array = newArray((floor(rect\_length/sect100\_length)+1));

while (m<(rect\_length)) {

if (p>0) {

distance\_pixpix\_array[r] = sqrt(((profile[p]-profile[p-1])\*(profile[p]-profile[p-1]))+1);

distance\_pixpix\_sum += distance\_pixpix\_array[r];

r=r+1;

}

if (p == (u\*sect100\_length) || m == (rect\_length - 1)) {

pix\_distance\_sq\_sect100 = (p-((u-1)\*sect100\_length))\*(p-((u-1)\*sect100\_length));

distance\_sect100\_array[u-1] = sqrt(((profile[p]-profile[(u-1)\*sect100\_length])\*(profile[p]-profile[(u-1)\*sect100\_length]))+pix\_distance\_sq\_sect100);

distance\_sect100\_sum += distance\_sect100\_array[(u-1)];

u=u+1;

}

if (p == (v\*sect50\_length) || m == (rect\_length - 1)) {

pix\_distance\_sq\_sect50 = (p-((v-1)\*sect50\_length))\*(p-((v-1)\*sect50\_length));

distance\_sect50\_array[v-1] = sqrt(((profile[p]-profile[(v-1)\*sect50\_length])\*(profile[p]-profile[(v-1)\*sect50\_length]))+pix\_distance\_sq\_sect50);

distance\_sect50\_sum += distance\_sect50\_array[v-1];

v=v+1;

}

if (p == (w\*sect20\_length) || m == (rect\_length - 1)) {

pix\_distance\_sq\_sect20 = (p-((w-1)\*sect20\_length))\*(p-((w-1)\*sect20\_length));

distance\_sect20\_array[w-1] = sqrt(((profile[p]-profile[(w-1)\*sect20\_length])\*(profile[p]-profile[(w-1)\*sect20\_length]))+pix\_distance\_sq\_sect20);

distance\_sect20\_sum += distance\_sect20\_array[w-1];

w=w+1;

}

if (p == (z\*sect10\_length) || m == (rect\_length - 1)) {

pix\_distance\_sq\_sect10 = (p-((z-1)\*sect10\_length))\*(p-((z-1)\*sect10\_length));

distance\_sect10\_array[z-1] = sqrt(((profile[p]-profile[(z-1)\*sect10\_length])\*(profile[p]-profile[(z-1)\*sect10\_length]))+pix\_distance\_sq\_sect10);

distance\_sect10\_sum += distance\_sect10\_array[z-1];

z=z+1;

}

m=m+1;

p=p+1;

}

roughness\_pixpix\_array[k] = distance\_pixpix\_sum/distance\_pixpix\_array.length;

roughness\_pixpix\_sum += roughness\_pixpix\_array[k];

roughness\_sect100\_array[k] = distance\_sect100\_sum/rect\_length;

roughness\_sect100\_sum += roughness\_sect100\_array[k];

roughness\_sect50\_array[k] = distance\_sect50\_sum/rect\_length;

roughness\_sect50\_sum += roughness\_sect50\_array[k];

roughness\_sect20\_array[k] = distance\_sect20\_sum/rect\_length;

roughness\_sect20\_sum += roughness\_sect20\_array[k];

roughness\_sect10\_array[k] = distance\_sect10\_sum/rect\_length;

roughness\_sect10\_sum += roughness\_sect10\_array[k];

j=j+1;

k=k+1;

}

//Calculates average image roughness in ROI

roughness\_pixpix\_avg = roughness\_pixpix\_sum/roughness\_pixpix\_array.length;

roughness\_sect100\_avg = roughness\_sect100\_sum/roughness\_sect100\_array.length;

roughness\_sect50\_avg = roughness\_sect50\_sum/roughness\_sect50\_array.length;

roughness\_sect20\_avg = roughness\_sect20\_sum/roughness\_sect20\_array.length;

roughness\_sect10\_avg = roughness\_sect10\_sum/roughness\_sect10\_array.length;

selectWindow(param\_table);

Table.set("Parameter", Table.size, "Average Roughness Pixel-by-Pixel", param\_table);

Table.set("Parameter Value", Table.size-1, roughness\_pixpix\_avg, param\_table);

Table.set("Parameter", Table.size, "Average Roughness 10 Sections", param\_table);

Table.set("Parameter Value", Table.size-1, roughness\_sect10\_avg, param\_table);

Table.set("Parameter", Table.size, "Average Roughness 20 Sections", param\_table);

Table.set("Parameter Value", Table.size-1, roughness\_sect20\_avg, param\_table);

Table.set("Parameter", Table.size, "Average Roughness 50 Sections", param\_table);

Table.set("Parameter Value", Table.size-1, roughness\_sect50\_avg, param\_table);

Table.set("Parameter", Table.size, "Average Roughness 100 Sections", param\_table);

Table.set("Parameter Value", Table.size-1, roughness\_sect100\_avg, param\_table);

Table.update;