CHAPTER 1

THE SECOND

1.1 Electron Microscopy Preparation

1.1.1 Tissue Processing

Tissue Processing
Dehydration to Plastic

Christopher Creveling, Graduate Student [*]

June 6th, 2018

Abstract

Tissue preparation in the electron microscopy lab to use TEM to look at the vitreo-retinal interface in eyes.

§1 Introduction

This document is intended to be used to process tissue from formalin to embedded plastic to be used on the transmission electron microscope (TEM) to identify the orientation of collagen fibers.

§1.1 Sorting

Begin first by sorting the tissue in two piles of tissue that was peeled and tissue that was adjacent to the peeled region. Then write down the identification ID # on the paper to keep the proper vial straight during the tissue process.

§1.1.1 Identification ID #

Sheep #, L/R, E/P, P/A

For example, *UL-15A-B Left Equator Peel* can be reduced to *UL15LEP*

§2 Dehydration

First place samples in glass vials. Use forceps if it is required to remove excess waste from the container. Properly label the samples from before section 1.1.1 and place the label on the vial. Before adhering the label to the vial, write down the number of specimens in the vial to ensure that the specimens don't get lost during the process. Use tape to ensure that the label will not be removed from the vial during the process.

[*]N. Chandler was with the Electron Microscopy Facility, University of Utah, Salt Lake City, UT, 84112 USA e-mail: (see http://www.bioscience.utah.edu/molecular-biology/core-facilities.php).

§2.1 Buffer Rinse

Remove the fixative from the existing vial using the micropipette. Be sure not to suck out the tissue. Then fill the vial with buffer - 0.1M Sodium Cacodylate buffer.

§2.1.1 Agitation

Put the sample vials in the rotating agitator for 5 minutes.

§2.2 Buffer Rinse

Remove the buffer from section 2.1 and replace with new buffer - 0.1M Sodium Cacodylate buffer.

§2.2.1 Agitation

Put the sample vials in the rotating agitator again for 5 minutes.

§2.3 Osmium dilution

During the previous agitation step in section 2.2.1 dilute the osmium tetroxide OsO_4 (4% in dH_2O) with 0.2 M Sodium Cacodylate buffer in a 1:1 mixture. Be sure to filter the Osmium tetroxide with a millipore filter to remove any excess particulate that would otherwise result in artifacts inside the tissue.

§2.4 Osmium rinse

Remove the 0.1M Sodium cacadylate buffer from the vials and replace with the diluted Osmium from section 2.3. Use just enough diluted Osmium to cover the tissue.

§2.4.1 Agitation

Put the sample vials back in the rotating agitator again for one hour.

§2.5 DI water rinse

Remove the diluted Osmium tetroxide from the vials and rinse with DI water. The DI water will be filtered [1]. This step is done to remove excess osmium.

§2.5.1 Agitation

Put the sample vials back in the rotating agitator again for 5 minutes.

§2.6 Uranyl Acetate rinse

Remove the DI water from the vials and replace with Saturated 4% Aqueous Uranyl Acetate. The Uranyl Acetate also needs to be filtered using a millipore filternote1 on a 10 ml syringe.

§2.6.1 Agitation

Put the sample vials back in the rotating agitator again for one hour.

[1]The millipore filter is used to remove any excess particulate that would otherwise result in artifacts inside the tissue.

§3 Final acetone dehydration step

The final step of the dehydration process is to replace all of the moisture in the tissue from H_2O to pure acetone. This is done with a series of rinses in various percentages of alcohol with the last set of rinses in acetone. **Note - if there is not enough alcohol mixtures in the hood then you will need to make more. When making the dilutions, use the graduated cylinder that is in the sink and mix the highest concentrations first to ensure that the percentages of alcohol is correct. Start with 95 then 70 then 50 etc. Also be sure that the ethanol containers are covered to prevent evaporation during each step of the dehydration process.

§3.1 50% Ethanol Alcohol

Remove the urinal acetate from the vial in section 2.6 to the appropriate container. Next use the micropipette and fill the vial with 50% Ethanol Alcohol; ensure that the tissue specimen is well covered.

§3.1.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.2 70% Ethanol Alcohol

Remove the 50% Ethanol Alcohol from the vial. Next use the micropipette and fill the vial with 70% Ethanol Alcohol; ensure that the tissue specimen is well covered.

§3.2.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.3 95% Ethanol Alcohol

Remove the 70% Ethanol Alcohol from the vial. Next use the micropipette and fill the vial with 95% Ethanol Alcohol; ensure that the tissue specimen is well covered.

§3.3.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.4 95% Ethanol Alcohol

Remove the 95% Ethanol Alcohol from the vial. Next use the micropipette and fill the vial with 95% Ethanol Alcohol; ensure that the tissue specimen is well covered.

§3.4.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.5 100% Ethanol Alcohol

Remove the 95% Ethanol Alcohol from the vial. Next use the micropipette and fill the vial with 100% Ethanol Alcohol; ensure that the tissue specimen is well covered.

§3.5.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.6 100% Ethanol Alcohol

Remove the 100% Ethanol Alcohol from the vial. Next use the micropipette and fill the vial with 100% Ethanol Alcohol; ensure that the tissue specimen is well covered.

§3.6.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.7 100% Ethanol Alcohol

Remove the 100% Ethanol Alcohol from the vial. Next use the micropipette and fill the vial with 100% Ethanol Alcohol; ensure that the tissue specimen is well covered.

§3.7.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.8 100% Ethanol Alcohol

Remove the 100% Ethanol Alcohol from the vial. Next use the micropipette and fill the vial with 100% Ethanol Alcohol; ensure that the tissue specimen is well covered.

§3.8.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.9 Acetone

Remove the 100% Ethanol Alcohol from the vial. Next use the micropipette and fill the vial with acetone; ensure that the tissue specimen is well covered.

§3.9.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.10 Acetone

Remove the acetone from the vial. Next use the micropipette and fill the vial with acetone; ensure that the tissue specimen is well covered.

§3.10.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.11 Acetone

Remove the acetone from the vial. Next use the micropipette and fill the vial with acetone; ensure that the tissue specimen is well covered.

§3.11.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§3.12 Acetone

Remove the acetone from the vial. Next use the micropipette and fill the vial with acetone; ensure that the tissue specimen is well covered.

§3.12.1 Agitation

Put the sample vials in the rotating agitator again for 10 minutes.

§4 Infiltration

Once the tissue samples have been completely dehydrated and all moisture in the sample has been replaced with acetone, the next step is to infiltrate with plastic. This will allow the tissue to be embedded and then cut using the Ultramicrotomes. This will also take a few steps that still incorporate various mixtures of acetone and plastic.

§4.1 Acetone & Plastic

The first step is to remove the acetone from the vial using a micropipette and replacing it with a 1:1 mixture of acetone and plastic. Again, as mentioned before, the vial does not need to be filled up to the brim, just enough to throughly allow plastic to infiltrate the tissue.

§4.1.1 Agitation

Put the sample vials in the rotating agitator again for one hour.

§4.2 Acetone & Plastic Overnight Option**

If you are to finish the process for the day and return the next, then perform the following option, if not skip to section 4.3. First remove the 1:1 mixture from section 4.1 and replace with a 3:1 mixture of plastic to acetone and let it sit overnight.

§4.3 Acetone & Plastic

If you are to finish the process the same day then skip section 4.2. First remove the 1:1 mixture from section 4.1 and replace with a 3:1 mixture of plastic to acetone.

§4.3.1 Agitation

Put the sample vials in the rotating agitator again for one hour.

§4.4 Pure Plastic

First remove the 3:1 mixture from either section 4.2 or 4.3 and replace with pure plastic.

§4.4.1 Agitation

Put the sample vials in the rotating agitator again for one hour.

§4.4.2 Vacuum

Place all of the vials with the lids removed inside the vacuum chamber. Turn the pump on to remove air from the chamber. This will remove all air from the samples that has been embedded inside the tissue and will allow the infiltration of plastic to fully take affect. Let the samples sit inside the vacuum chamber for one hour.

§4.5 Pure Plastic

Remove the pure plastic from section 4.4 and replace with pure plastic again.

§4.5.1 Agitation

Put the sample vials in the rotating agitator again for one hour.

§4.5.2 Vacuum

Place all of the vials with the lids removed inside the vacuum chamber. Turn the pump on to remove air from the chamber. This will remove all air from the samples that has been embedded inside the tissue and will allow the infiltration of plastic to fully take affect. Let the samples sit inside the vacuum chamber for one hour.

§5 Embedding

The next step is to embed the plasticized tissue into the mold. Before forgoing with this process, a list of all of the specimens will need to be created on Excel to print and cut out. For example if there are five specimens in the same vial, make a list of sample names with the specimen ID (A), specimen ID (B), ... specimen ID (E). Next, grab a razor blade and a wooden stir stick. Simply use the razor blade to shave away wood from the stir stick to make a flat surface. The flat surface will be used to transfer specimens from the vials to the mold. Place the printed out label inside the mold and set the mold inside the oven to let it bake the specimens to cure the plastic.

§6 Cutting

After the plastic has cured, remove the specimen to be cut and use the microtome to shave away thin layers to be used for TEM.

§7 Grid Staining

Once thin sections have been placed on grids from section 6 the grids will need to be stained to increase the contrast for TEM. Two chemicals will be Uranyl Acetate and Lead Citrate.

§7.1 Preparation

Using the square petri-dish and wax from the cupboard cut the wax to fit the inside the petri-dish. Clean the wax with alcohol and DI water to remove any impurities on the wax that would alter the grid samples. This will also prevent the drops from coagulating together on the wax. Simply rinse the wax to clean it off.

§7.2 Chemical Prep

After the wax has been cleaned and cut remove the saturated Uranyl Acetate and Reynold's Lead Citrate from the refrigerator. Grab two small 1 ml syringes from the drawer and fill up each syringe with either UA or Lead Citrate. Then place one small filter on the end of the syringe filled with UA and two filters on the syringe filled with Lead Citrate.

§7.3 UA Stain

Using the 1 ml syringe with a single filter place a droplet of UA for each grid that you need to stain evenly spaced on the wax pad. Use the forceps and remove the grids from the grid holder and place on top of the UA droplet. Be sure to place the grid shiny side down to allow the UA to stain the specimen.

§7.4 Timer - 18 minutes

Set the timer for 18 minutes. During this time fill up enough 30 ml syringes with DI water for rinsing both UA and Lead Citrate. You will need approximately 10 ml per sample per rinse. Place a large filter on the end of the syringe.

§7.5 Staging Area

Grab a small round petri dish and insert two filter papers to absorb the water following the rinse. Use a pen or pencil to mark the paper to help organize the order of specimens to prevent a mix up.

§7.6 First Rinse

After 18 minutes, pick up the grid with forceps and rinse with 10 ml of DI water. Hold the forceps at a 60° angle from the horizontal and drip the water down the curved section of the forceps. After the rinse, place the specimens inside the round petri dish to remove excess DI water. Once all of the specimens have been placed on the filter paper, a few sodium hydroxide crystals will need to be placed inside the square petri dish. The NaOH will help prevent any sort of moisture from interfering with the grid during the staining process. Next, use the other 1 ml syringe with Lead Citrate and place drops on the wax pad following the same procedure mentioned before in section 7.3.

§7.7 Lead Citrate Rinse

Using the forceps, grip the grid and place it on top of the Lead Citrate droplet with the shiny side down which allows the grid to be stained. Set the timer for eight minutes.

§7.8 Second Rinse

After eight minutes have passed, repeat the same step as in 7.6. Once the grids have completely dried, place them back in the grid holder and they are ready for the TEM.

§7.9 Cleanup

Dispose of the petri dish in the unwanted UA container.

§8 Transmission Electron Microscopy

Transmission electron microscopy (TEM) is a microscopy technique in which a beam of electrons is transmitted through an ultra-thin specimen, interacting with the specimen as it passes through it.

§8.1 TEM

Head over to the TEM and begin imaging!

#	Step	Instruction	Time	☑	☑	☑	☑	☑	☑	☑	☑
1	Dehydration	0.1 M Sodium Cacodylate buffer	*A** 5 minutes								
2	Dehydration	0.1 M Sodium Cacodylate buffer	*A** 5 minutes								
3	Fix	4% OsO_4 with 0.2 M Sodium Cacodylate buffer (1:1 filtered)	*A** 60 minutes								
4	Rinse	DI water rinse	*A** 5 minutes								
5	Stain	Saturated 4% Aqueous Uranyl Acetate (filtered)	*A** 60 minutes								
6	Dehydration	50% Ethanol	*A** 10 minutes								
7	Dehydration	70% Ethanol	*A** 10 minutes								
8	Dehydration	95% Ethanol	*A** 10 minutes								
9	Dehydration	95% Ethanol	*A** 10 minutes								
10	Dehydration	100% Ethanol	*A** 10 minutes								
11	Dehydration	100% Ethanol	*A** 10 minutes								
12	Dehydration	100% Ethanol	*A** 10 minutes								
13	Dehydration	100% Ethanol	*A** 10 minutes								
14	Dehydration	Acetone	*A** 10 minutes								
15	Dehydration	Acetone	*A** 10 minutes								
16	Dehydration	Acetone	*A** 10 minutes								
17	Dehydration	Acetone	*A** 10 minutes								
18	Infiltration	1:1 Plastic to Acetone	*A** 60 minutes								
19	Infiltration	3:1 Plastic to Acetone	*A** 60 minutes								
20	Infiltration	Pure Plastic	*A** 60 minutes								
21	Vacuum	Vacuum	*V** 60 minutes								
22	Infiltration	Pure Plastic	*A** 60 minutes								
23	Vacuum	Vacuum	*V** 60 minutes								
24	Embedding	Embedding	Limitless								

Table 1: Simplified instructions to check off the steps during the tissue processing by hand. *A** indicates Agitation, and *V** indicates Vacuum.

Station #	Step	Instruction	Time	☑	☑	☑	☑
1	-	-	-	-	-	-	-
2	Dehydration	0.1 M Sodium Cacodylate buffer	*A** 10 minutes				
3	Fix	4% OsO_4 with 0.2 M Sodium Cacodylate buffer (1:1 filtered)	*A** 60 minutes				
4	Rinse	DI water rinse	*A** 10 minutes				
5	Stain	Saturated 4% Aqueous Uranyl Acetate (filtered)	*A** 60 minutes				
6	Dehydration	50% Ethanol	*A** 10 minutes				
7	Dehydration	70% Ethanol	*A** 10 minutes				
8	Dehydration	95% Ethanol	*A** 10 minutes				
9	Dehydration	95% Ethanol	*A** 10 minutes				
10	Dehydration	100% Ethanol	*A** 10 minutes				
11	Dehydration	100% Ethanol	*A** 10 minutes				
12	Dehydration	100% Ethanol	*A** 10 minutes				
13	Dehydration	100% Ethanol	*A** 10 minutes				
14	Dehydration	Acetone	*A** 10 minutes				
15	Dehydration	Acetone	*A** 10 minutes				
16	Dehydration	Acetone	*A** 10 minutes				
17	Dehydration	Acetone	*A** 10 minutes				
18	Infiltration	1:1 Plastic to Acetone	*A** 60 minutes				
19	Infiltration	3:1 Plastic to Acetone	*A** 60 minutes				
20	Infiltration	Pure Plastic	*A** & *V** 60 minutes				
21	Infiltration	Pure Plastic	*A** & *V** 60 minutes				
22	-	-	-	-	-	-	-
23	-	-	-	-	-	-	-
24	-	-	-	-	-	-	-

Table 2: Simplified instructions to check and make sure the automatic tissue processor is set up at the correct stations. Each vial should be filled with 20 ml when processing. Be sure to check the program on the automatic tissue processor; it should be marked by program #2. *A** indicates Agitation, and *V** indicates Vacuum.

Tissue Processing
Dehydration to Plastic

Christopher Creveling, Graduate Student [*]

June 8th, 2018

Abstract

Tissue preparation in the electron microscopy lab to prepare the EMbed 812 for tissue processing.

§1 Introduction

This document is intended to be used to process tissue from formalin to embedded plastic to be used on the transmission electron microscope (TEM) to identify the orientation of collagen fibers.

§2 Embed 812

§2.1 Personal Protective Equipment

Begin first by grabbing a lab coat and then use paper towels and acetone to clean off the scale used for measuring out the mass of various resin mixtures.

§2.2 Recipe for EMbed 812

Remove the four chemicals for the EMbed 812 resin from the cabinet by using the WPE-147. Where W.P.E. is the Weight per Epoxide Equivalent).

Ingredient	Unit
EMbed 812 Resin	51.80 g
DDSA	26.68 g
NMA	21.67 g
BDMA	2.5 ml

Table 1: Simplified instructions to check and make sure the automatic tissue processor is set up at the correct stations. Each vial should be filled with 20 ml when processing. Be sure to check the program on the automatic tissue processor; it should be marked by program #2.

[*]N. Chandler was with the Electron Microscopy Facility, University of Utah, Salt Lake City, UT, 84112 USA e-mail: (see http://www.bioscience.utah.edu/molecular-biology/core-facilities.php).

§3 Lab Equipment

Grab one (400 ml) Tripore container along with four clean pipettes.

§4 EMbed 812

Balance the scale with the Tripore container and pour in 51.80 g of EMbed 812 Resin. Clean the bottle and cap with Kimwipes and throw out the pipette. Grab a strip of Parafilm to stretch over the cap to ensure an air-tight seal.

§5 DDSA

Balance the scale after 51.80 g of EMbed 812 Resin has been added to the Tripore container. Add 26.68 g of DDSA to the container by first underpouring and then using a pipette to add the rest of the DDSA. Clean the bottle and cap with Kimwipes and throw out the pipette. Grab a strip of Parafilm to stretch over the cap to ensure an air-tight seal.

§6 NMA

Balance the scale after 26.68 g of DDSA has been added to the Tripore container. Add 21.67 g of NMA to the container by first underpouring and then using a pipette to add the rest of the NMA. Clean the bottle and cap with Kimwipes and throw out the pipette. Grab a strip of Parafilm to stretch over the cap to ensure an air-tight seal.

§7 Stir

Move the Tripore container inside the hood. Add a stirbar to the Tripore container containing EMbed 812 Resin, DDSA, and NMA to stir the mixture for 10 minutes in the hood.

§8 BDMA

In the hood, while the Tripore container is being stirred, use a graduated micropipette and obtain 2.5 ml of BDMA. The BDMA is used as the accelerant for polymerization.

§9 Stir

Stir the Tripore container containing EMbed 812 Resin, DDSA, NMA, and BDMA mixture for 10 minutes in the hood. The mixture should turn orange after the BDMA has been added.

§10 Parafilm

Using Parafilm wrap, ensure that each chemical lid has been wrapped to keep an air-tight seal.

§11 Syringe

Grab eight syringes and caps from the cupboards and prepare them for filling up with the resin. Place the newly filled resin syringes in the freezer.

§12 Clean-up

Clean the magnetic stirbars with acetone. Be sure to put vinyl liners inside the gloves.

§13 Embed

Embed the tissue samples!

1.2 MatLab Least Squares

Script 1: *Matlab script that performs a least squares regression calculation.*

```
function [A] = Least_Squares(A)
% Calculate the slope and y-intercept using matrix math
% x & y are the coordinates of points
x = A(:,1);
y = A(:,2);
Z = ones(length(x),2);
Z(:,2) = x;
% Calculate the matrix inverse for the constants of the regression
A = inv(Z'*Z)*(Z'*y);
return
end
```

1.3 Ridge Detection Input Parameters

Script 2: *Matlab script that determine ridge detection parameters using TEM images.*

```
% Sigma selection parameter
% Christopher Creveling

close all
clear
clc

[file_name_root, dirname] = uigetfile('*.tif');
info = imfinfo(file_name_root);
% Gathers the resolution from the image data
resolution = info.XResolution;

line_width = 0.026; % Micron length
U =  204; % Image upper intensity value (background)
P = 160; % Pixel intensity for the contrast value

% line_width = input('Max of  four line width measurements (Microns)\n');
fprintf('Resolution %f (pixels/micron)\n', resolution);

fprintf('Line width %f (microns)\n', line_width);
% resolution = 623.1429; % conversion between length and pixels

L = line_width*resolution; %Line width in pixels
fprintf('Line width %f (pixels)\n', L);
w = L/2; % width of a line in pixels
sigma = w/sqrt(3) + 0.4; % calculated sigma value
% sigma = 3.1
fprintf('Sigma = %f\n', sigma)

% sigma = 3.8; % approximate value

fprintf('U --- %d\n', U);
fprintf('P --- %d\n', P);
```

```matlab
% Contrast (difference between upper and selected pixel intensity values)
h = U - P;
%h = 42;
fprintf('h = %d\n', h)

% First derivative of the gaussian kernel [Equation 4] - 1D
g_p1Dx = @(x, sigma)-x/(sqrt(2*pi)*sigma^3)*exp(-(x^2)/(2*sigma^2));
% Second directional derivative approximation [Equation 8]
rb_pp1Dx = @(x) h*(g_p1Dx(x + w, sigma) - g_p1Dx(x - w, sigma));
% Evaluate the second order approximation at zero to find out the upper
% threshold value 1D
fprintf('1D upper threshold approximation is %f\n', abs(rb_pp1Dx(0)))

% First derivative of the 2D gaussian kernel [Equation 4]
% g_p2Dx = @(x, y, sigma)-x/(2*pi*sigma^4)*exp(-(x^2 + y^2)/(2*sigma^2));
% First derivative of the 2D gaussian kernel [Equation 4]
% g_p2Dy = @(x, y, sigma)-y/(2*pi*sigma^4)*exp(-(x^2 + y^2)/(2*sigma^2));
% Second directional derivative approximation [Equation 8]
% rb_pp2D = @(x, y) h*(g_p2Dx(x + w, y, sigma) - ...
%     g_p2Dx(x - w, y, sigma) + g_p2Dy(x, y + w, sigma) - ...
%     g_p2Dy(x, y - w, sigma));
% Evaluate the second order approximation at zero to find out the upper
% threshold value 1D
% fprintf('2D upper threshold approximation is %f\n', abs(rb_pp2D(0, 0)))
% s = 0.006:0.001:0.03; % Range of sigma values
%
% for i = 1:length(s)
%     H(i) = abs(h*(g_p1Dx(0 + w, s(i)) - g_p1Dx(0 - w, s(i))));
% end
% H';
```

1.4 Analyze Ridge Detection Output

Script 3: *Matlab script that analyzes ridge detection output from TEM images.*

```matlab
% Name:  Christopher Creveling
% Date:  11/13/18
% Title:  Image analysis Ridge Detection Interpretation

% Description:  After running a non-local means filter and further running
% a Ridge-Detection algorithm through Fiji I am trying to learn to extract
% what the output is giving me


%{
Output from Ridge-Detection
/** This class holds one extracted line.  The field num contains the number of
 points in the line.  The coordinates of the line points are given in the
 arrays row and col.  The array angle contains the direction of the normal
 to each line point, as measured from the row-axis.  Some people like to
 call the col-axis the x-axis and the row-axis the y-axis, and measure the
 angle from the x-axis.  To convert the angle into this convention, subtract
 PI/2 from the angle and normalize it to be in the interval [0, 2*PI).  The
 array response contains the response of the operator, i.e., the second
 directional derivative in the direction of angle, at each line point.  The
 arrays width_l and width_r contain the width information for each line point
 if the algorithm was requested to extract it; otherwise they are NULL.  If
 the line position and width correction was applied the contents of width_l
 and width_r will be identical.  The arrays asymmetry and contrast contain
 the true asymmetry and contrast of each line point if the algorithm was
 instructed to apply the width and position correction.  Otherwise, they are
 set to NULL.  If the asymmetry, i.e., the weaker gradient, is on the right
 side of the line, the asymmetry is set to a positive value, while if it is
 on the left side it is set to a negative value. */
%}

clear all;
close all force; % Force the message boxes to close
clear;
clc;

cd 'Z:\students\Yousef\TEM\Ridge detection\Fiji Output'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Real TEM Image Data
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Import the data from the CSV file
synthetic = false;

% file root name  _crop
file_name_root = 'H160993LPA-3_12_L4';
% File name extension  _h85_H191_L06_S44
file_name_extension = '_C41_U351_L02_S220_W0010';
% file_name_extension = '';

% Ridge Detection Results
table_1 = readtable(strcat(file_name_root, file_name_extension, '_RD.csv'));
% Ridge Detection Junction Results
% table_2 = readtable(strcat(file_name_root, file_name_extension, ...
% '_RD_J.csv'));
% Ridge Detection Summary Results
table_3 = readtable(strcat(file_name_root, file_name_extension, '_RD_S.csv'));
% Extract information from the original image
img = imread(strcat(file_name_root, '.tif'));
info = imfinfo(strcat(file_name_root, '.tif'));
x_scale = info.XResolution;
y_scale = info.YResolution;
val = 1;%input(prompt);

fiber_color_num = 11; % the number of fiber divisions for the visual output

height = size(img, 2);
width = size(img, 1);

% Set up the file for outputting data
fileID = fopen(strcat(file_name_root, file_name_extension, '.txt'), 'w');

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Identify how to properly shift the TEM image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
prompt = ['Are the collagen fibers on the top (1), right (2), ' ...
    'bottom (3), or left (4)? \n'];
% val = 2;%input(prompt);
```

```
if (val == 1)
    %     No need to shift pixels
    shift_x = 0;
    shift_y = 0;
elseif (val == 2)
    % shift pixels to the right
    shift_x = max(width) - max(table_1.X*x_scale);
    shift_y = 0;
elseif (val == 3)
    % shift pixels down
    shift_x = 0;
    shift_y = max(width) - max(table_1.X*y_scale);
elseif (val == 4)
    %     No need to shift pixels
    shift_x = 0;
    shift_y = 0;
else
    err = 'Invalid input';
    error(err);
end


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ask for collagen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
imshow(img);


answer = questdlg('Do Collagen fibers exist?');
switch answer
    case 'Yes'
        close

        %%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Plot the RD classification color for all of the fiber segments detected
        % by the algorithm
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %figure
        %imshow(img);
        %hold on
        %RD_classification = unique(table_1.Class);
        %RD_class_vals = []; % Empty array
        %C = hsv(length(RD_classification));
        % for i = 1:length(RD_classification)
        %     Ridge_Detection_Class{i} = RD_classification(i);
        %     RD_class_vals(i).XY = [table_1.X(categorical(table_1.Class) ==
↪   RD_classification{i}), ...
        %           table_1.Y(categorical(table_1.Class) ==
↪   RD_classification{i})]*x_scale;
        %     plot(RD_class_vals(i).XY(:, 1) + shift_x, RD_class_vals(i).XY(:, 2) +
↪   shift_y, '.', 'Color', C(i, :), 'markersize', 5, 'linewidth', 3);
        %     %     Legend(i) = num2str(RD_classification(i));
        % end
        % legend(RD_classification, 'location', 'best');

        Length_segment = table_1.Length; % extract line length
        Contour_ID = table_1.ContourID;

        %%
        % Ridge-Detection results

        figure;
        imshow(img);
        title('\bf Original Image')

       msgStr = ['Select two points that define the ILM (Right to Left' ...
                ' if Collagen Fibrils are above, Left to Right if'  ...
                ' Collagen Fibrils are below'];
        % Indicate the ILM used for angle calculations
        f = msgbox(msgStr, 'ILM');
        pause(3);
        [ILM.x, ILM.y] = ginput(2);
        % delete(f); % Delete the message box
        hold on
        plot(ILM.x, ILM.y, 'bo', 'linewidth', 2);
        % Sorts rows of the input to maintain correct order (ascending)
        % ILM.x = sortrows(ILM.x);
        ILM_slope = [];
        ILM_angle = [];
        for i = 1:length(ILM.x)-1
            numerator = (ILM.y(i+1) - ILM.y(i));
            denominator = (ILM.x(i+1) - ILM.x(i));
            % slope of the line
            ILM_slope(i) = numerator/denominator;
            % Angle of the ILM relative to the x-axis
            ILM_angle(i) = -atan(numerator/denominator)*180/pi;
        end
        fprintf('ILM slope = %f\n', ILM_slope);

        slope = mean(ILM_slope); % Mean slope between the points
        y_int = ILM.y(1) - slope*ILM.x(1); % Solve for the y-intercept



        %% Create Rectangle

        x1 = linspace(ILM.x(1), ILM.x(2));
        y1 = linspace(ILM.y(1), ILM.y(2));
        d = 1 * x_scale;     %distance in microns

        height = size(img, 2);
        width = size(img, 1);
        aLine = [-ILM_slope, 1, -y_int];

        fcn = @(x)ILM_slope*x + y_int; % Function handle
        fplot(fcn, [0, width], 'r');

        start_ = [ILM.x(1) ILM.y(1)];
        goal_ = [ILM.x(2) ILM.y(2)];

        n = 2;
        t = linspace(0, 1, n);
        v = goal_ - start_;
        x3 = start_(1) + t*v(1);
        y3 = start_(2) + t*v(2);
        v =  d* v / norm(v);

        for i=1:n
            line([x3(i) - v(2)], [y3(i) + v(1)]);
            plot([x3(i) - v(2)], [y3(i) + v(1)], 'ro', 'linewidth', 2);
        end

        x3f = x3 - v(2);
        y3f = y3 + v(1);

        % Coordinates of the region of interest within the 1 micron rectangle
        xv = [ILM.x(1), x3f(1), x3f(2), ILM.x(2)];
        yv = [ILM.y(1), y3f(1), y3f(2), ILM.y(2)];

        % Plots the 1 micron rectangle
        plot(xv, yv, 'r--', 'LineWidth', 1.5)

        Answer = questdlg('Is this correct?');

        switch Answer
            case 'Yes'
                In = inpolygon(table_1.X*x_scale, table_1.Y*y_scale, xv, yv);

                table_1.X = In .* table_1.X;
                table_1.Y = In .* table_1.Y;

                table_1(~table_1.X, :) = [];

            case 'No'
                fprintf('Please run code again')
                msgbox('Please run code again');

                return
        end


        %%%%%% End of create Rectangle

        %%
        % Define the input parameters for the line to border points (Ax+By+C=0)
        % A = slope
        % B = integer in front of y
        % C = y-intercept
        aLine = [-slope, 1, -y_int];


        % extrapolate the ILM line on the image as well as calculate the distance
        ILM_x_pts = linspace(0, width, 100);
        for i = 1:length(ILM_x_pts)
            ILM_line(i) = slope*ILM_x_pts(i) + y_int; % + ILM.x(end)
        end
        ILM_length = sqrt((ILM.x(2)-ILM.x(1))^2 + (ILM.y(2) - ILM.y(1))^2);
        ILM_length = ILM_length/x_scale;
        fprintf('ILM length = %f microns\n', ILM_length);
        ILM_angle = (mean(ILM_angle));
        fprintf(['ILM angle is %f degrees relative to the x-axis ' ...
            '(Unit circle)\n'], ILM_angle);

        fiber_min_length = 0.044962164;

        % Indicate the five points on the ILM used for thickness measurements
        figure
        imshow(img)
        for i = 1:5
            f = msgbox(['Select the first two points that define the ' ...
                'ILM thickness'], 'ILM');
            %     pause(1);
            [ILM_thick(i).x, ILM_thick(i).y] = ginput(2);
            hold on
            plot(ILM_thick(i).x, ILM_thick(i).y, 'g-o', 'linewidth', 1);
            % Pythogrean theorem
            ILM_thick(i).measurement = sqrt((ILM_thick(i).x(1) - ILM_thick(i).x(2))^2
↪   + ...
                (ILM_thick(i).y(1) - ILM_thick(i).y(2))^2);
            delete(f); % Delete the message box
        end
        for i = 1:5
            ILM_measurement(i) = ILM_thick(i).measurement;
        end
        L{4} = 'ILM thickness measurements';
        %legend(L, 'location', 'best');
        axis image;

        ILM_thickness = mean(ILM_measurement)/x_scale*1000;
        fprintf('Average ILM thickness is %f nanometers \n', ILM_thickness);





        %%
        %  Loop over all of the unique Contour ID's and identify the length of each
        %  one
        ID_num = unique(Contour_ID);
        for i = 1:length(ID_num)
            unique_ID_lengths(i) = mean(table_1.Length(table_1.ContourID ==
↪   ID_num(i)));
            unique_ID_widths(i) = mean(table_1.LineWidth(table_1.ContourID ==
↪   ID_num(i)));
            unique_ID_ang_of_norm(i) = mean(table_1.AngleOfNormal(table_1.ContourID
↪   == ID_num(i)));
        end


        %%
        % figure;
        % imshow(img);
        % hold on

        % fiber_color_num = 12; % the number of fiber divisions for the visual
        % output (chosen from up above)

        % Properly match the associated ContourID with the unique_ID number and the
        % specified fiber length

        fiber_length = linspace(min(Length_segment), ...
            max(Length_segment)*0.8, fiber_color_num); %
        C = hsv(length(fiber_length)); % Splits up the colormap into 11 unique values
        m_size = 5;

        % Loop over the unique fiber segment lengths to break them apart by lengths
        for i = 1:length(fiber_length)
            % if the length of the fibers is longer than the specified bin put them here
            if i == length(fiber_length)
                % extract X & Y coordinates of each point based on the criteria
                fiber(i).x = table_1.X(table_1.Length > fiber_length(i));
                % extract X & Y coordinates of each point based on the criteria
                fiber(i).y = table_1.Y(table_1.Length > fiber_length(i));
                % Calculate fiber area (LineLength *LineWidth)
                % fiber(i).area = datatbl.Length(datatbl.Length > fiber_length(i)).*datatbl.LineWidth(datatbl.Length > fiber_length(i));
                fiber(i).len = table_1.Length(table_1.Length > fiber_length(i));
                fiber(i).wid = table_1.LineWidth(table_1.Length > fiber_length(i));
                % Fiber area = length * width (pixels)
                fiber(i).area = fiber(i).len.*fiber(i).wid;
                % Calculates the angle of the fiber
                % fiber(i).angle = atan2(max(fiber(i).y) - min(fiber(i).y), max(fiber(i).x) - min(fiber(i).x))*180/pi;
            else
                % extract X & Y coordinates of each point based on the criteria
                fiber(i).x = table_1.X(table_1.Length > fiber_length(i) & ...
                    table_1.Length <= fiber_length(i+1));
                % extract X & Y coordinates of each point based on the criteria
                fiber(i).y = table_1.Y(table_1.Length > fiber_length(i) & ...
                    table_1.Length <= fiber_length(i+1));
                % Calculate fiber area (LineLength *LineWidth)
                % fiber(i).area = datatbl.Length(datatbl.Length > fiber_length(i) & ...
                %   datatbl.Length <= fiber_length(i+1)).*datatbl.LineWidth(datatbl.Length > fiber_length(i) & ...
                %   datatbl.Length <= fiber_length(i+1));
                fiber(i).len = table_1.Length(table_1.Length > fiber_length(i) & ...
                    table_1.Length <= fiber_length(i+1));
                fiber(i).wid = table_1.LineWidth(table_1.Length > fiber_length(i) & ...
                    table_1.Length <= fiber_length(i+1));
                fiber(i).area = fiber(i).len.*fiber(i).wid; % Fiber area = length * width (pixels)
                % Calculates the angle of the fiber
                % fiber(i).angle = atan2(max(fiber(i).y) - min(fiber(i).y), ...
                %   max(fiber(i).x) - min(fiber(i).x))*180/pi;
            end
            tot_fiber_area(i) = sum(fiber(i).area); % sum up fiber area
        end

        % fiber_area = sum(tot_fiber_area); % fiber area
        % fprintf('Area of fiber segments [pixels]) %f\n', fiber_area);
        %  fprintf(['Collagen fiber segment density (Area of fibers ' ...
        %    '[pixels]/ILM length (nanometers)) %f\n'], ...
        %    fiber_area/ILM_length);

```

```matlab
        % Plot the fibers
        % for i = 1:length(fiber_length)
        %     plot(fiber(i).x*x_scale + shift_x, fiber(i).y*y_scale + shift_y, '.',
↪ 'color', C(i, :), 'markersize', m_size);
        % end
        %
        % title('\bf Scatter Plot of Collagen fiber segments with corresponding
↪ lengths');
        %
        % % Create the legend based upon the length in the fiber array
        % for i = 1:length(fiber_length)
        %     if i == length(fiber_length)
        %         Legend{i} = strcat('L \geq', num2str(fiber_length(i)), '\mu', 'm');
        %     else
        %         Legend{i} = strcat(num2str(fiber_length(i)), ...
        %             ' < L \leq', num2str(fiber_length(i+1)), '\mu', 'm');
        %     end
        % end
        %
        % [h, ~] = legend(Legend);
        % %// children of legend of type line
        % ch = findobj(get(h, 'children'), 'type', 'line');
        % set(ch, 'Markersize', 24); %// set value as desired
        % set(h, 'Interpreter', 'latex', 'location', 'best');
        % axis image;
        % set(gca, 'DataAspectRatio', [1 1 1]) % Adjust the aspect ratio for printing


        %%

        % %%
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % % Data from the Ridge Detection Junction Results CSV file
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % T2_x = table_2.X;
        % T2_y = table_2.Y;
        % T2_ID1 = table_2.ContourID1;
        % T2_ID2 = table_2.ContourID2;
        %
        % figure
        % imshow(img);
        % hold on;
        % C = hsv(length(T2_x));
        % for i = 1:length(T2_x)
        %     %     plot(All_Fibers(i).XYRes(:, 1), All_Fibers(i).XYRes(:, 2), '.',
↪ 'color', C(i, :), 'linewidth', 2);
        %     plot(T2_x(i)*x_scale + shift_x, T2_y(i)*y_scale + shift_y, 'o',
↪ 'linewidth', 3, 'markersize', 8, 'color', C(i, :));
        %     %     hold on;
        % end
        % axis image


        %%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Data from the Ridge Detection Summary Results CSV file
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Length_T3 = table_3.Length;
        % Width_T3 = table_3.MeanLineWidth;
        % ContourID_T3 = table_3.ContourID;
        %
        % figure
        % subplot(1, 2, 1);
        % hist(Length_T3, fiber_color_num); % histogram of the lengths from the summary results file
        % xlabel('\bf Lengths from summary results file');
        %
        % subplot(1, 2, 2);
        % hist(Width_T3, fiber_color_num); % histogram of the lengths from the summary results file
        % xlabel('\bf Mean line width from summary results file');

        % % Plots all of the junction points from the Fiji Output
        % figure;
        % imshow(img);
        % hold on
        % plot(c_X + shift_x, c_Y + shift_y, '.', bj_X + shift_x, bj_Y + shift_y, '.', ej_X + shift_x, ej_Y + shift_y, '.', sj_X + shift_x, sj_Y + shift_y, '.', nj_X + shift_x, nj_Y + shift_y, '.', 'markersize', 5)
        % title('\bf Ridge-Detection Results')
        % Legend_1 = legend({'Closed Points', 'Both Junction', 'End Junction', 'Start Junction', 'No Junction'}, 'location', 'best');
        % axis image
        % [h, ~] = legend(Legend_1);
        % ch = findobj(get(h, 'children'), 'type', 'line'); %// children of legend of type line
        % set(ch, 'Markersize', 24); %// set value as desired
        % set(h, 'Interpreter', 'latex', 'location', 'best');
        % axis image;
        % set(gca, 'DataAspectRatio', [1 1 1]) % Adjust the aspect ratio for printing

        %%

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Identify the fiber segments that are greater than the threshold and
        % identify whether or not they overlap and combine them into a single fiber
        % if they do
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % close all force;
        % clc;

        % extract the ContourID & Length in an array
        ID_Length = unique([table_1.ContourID, table_1.Length], 'rows');
        % Identify fiber segments that are greater than the minimum length
        segments = ID_Length(ID_Length(:, 2) >= fiber_min_length);
        % Identify fiber segments that are less than the minimum length
        short_segments = ID_Length(ID_Length(:, 2) < fiber_min_length);


        % Loop over all of the unique segments to identify which ones are contained
        % in the longer fibers by looking at all combinations.  i.e. if two fiber
        % segments have matching coordinates/slope they would be combined into a
        % single fiber and the list of potential fibers would decrease

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Go over the matching fibers and further eliminate duplicates
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Initalize the arrays
        tic
        atol = 0.02; % relative tolerance
        rtol = 0.01; % absolute tolerance

        c1 = 1; % while loop 1 counter
        c3 = 1; % fiber match counter
        count = 1; % iteration counter
        Lib = [];
        fiber_union = []; % Fiber unions
        fiber_segment = []; % initialize the array to be zero
        condition_segment = []; % Initialize the array to be zero
        lone_fibers = [];
        check_1 = false; % Initialize the while loop statements

        while (check_1 == false)
            check_2 = false; % Initialize the while loop statements
            c2 = 2; % while loop 2 counter
            while (check_2 == false)

                % X-coordinates
                A1 = table_1.X(table_1.ContourID == segments(c1));
                % Y-coordinates
                A2 = table_1.Y(table_1.ContourID == segments(c1));

                % X-coordinates
                B1 = table_1.X(table_1.ContourID == segments(c2));
                % Y-coordinates
                B2 = table_1.Y(table_1.ContourID == segments(c2));

                A = [A1, A2]; % [X, Y] coordinates from contour ID (i)
                B = [B1, B2]; % [X, Y] coordinates from contour ID (j)

                % Find the number of matches between array A and B and store them
                % every iteration
                % compares the two arrays to find matches (:, 1:2)(:, 1:2)
                Lib.logical = double(ismember(A, B, 'rows'));
                 % finds the mean value of the comparison array
                Lib.mean = mean(Lib.logical);
                % finds the mode value of the comparison array
                Lib.mode = mode(Lib.logical);
                % Sums the zeros
                Lib.num_zero = sum(Lib.logical == 0);
                % sums the ones
                Lib.num_one = sum(Lib.logical == 1);
                % Identifies the combination of contour ID#s
                Lib.IDs = [segments(c1), segments(c2)];

                % Consider looking at the slope of each line segment
                % Pass in an array of coordinates to find the slope & y-intercept [
↪   a_0 + a_1*x]
                MA = Least_Squares(A);
                % Pass in an array of coordinates to find the slope & y-intercept [
↪   a_0 + a_1*x]
                MB = Least_Squares(B);

                Ax = A(:, 1);
                Ay = A(:, 2);
                Bx = B(:, 1);
                By = B(:, 2);

                % Find the distance between the segments
                C_A = [mean(Ax), mean(Ay)]; % Center of mass for A
                C_B = [mean(Bx), mean(By)]; % Center of mass for B

                % Distance between fiber centers
                D_AB = sqrt((C_A(2) - C_B(2))^2 + (C_A(1) - C_B(1))^2);

                % Local extrema of each fiber segment
                A_E(1) = min(Ax);
                A_E(2) = max(Ax);
                A_E(3) = min(Ay);
                A_E(4) = max(Ay);
                B_E(1) = min(Bx);
                B_E(2) = max(Bx);
                B_E(3) = min(By);
                B_E(4) = max(By);

                % Distance between local extrema for each fiber segment assuming
                % they are linear
                % Distance between fiber centers
                L_A = sqrt((A_E(2) - A_E(1))^2 + (A_E(4) - A_E(3))^2);
                % Distance between fiber centers
                L_B = sqrt((B_E(2) - B_E(1))^2 + (B_E(4) - B_E(3))^2);

                % Three conditions need to be satisfied
                % Looks at the mode of the overlap values if there are any
                condition_1 = (Lib.mode == 1);
                % Compares how close the two slopes of similar segments are
                condition_2 = (all(abs(MA(2) - MB(2)) <= atol + rtol*abs(MB(2))));
                % Compares how close the two y-intercepts are
                condition_3 = (all(abs(MA(1) - MB(1)) <= atol + rtol*abs(MB(1))));
                % Is the distance between the fiber centers less than the length of
↪   the fiber segment
                condition_4 = ((D_AB < L_A) || (D_AB < L_B));
                condition_5 = (c1 ~= c2); % checks to see if A & B are duplicates

                % Used for debugging
                [condition_1, condition_2, condition_3, condition_4, ...
                    condition_5, segments(c1), segments(c2), count, ...
                    (max(table_1.ContourID) + 1)];

                % Five conditions need to be satisfied
                if [condition_1 && condition_5 || condition_2 && ...
                        condition_3 && condition_4 && condition_5]

                    A3 = table_1.Length(table_1.ContourID == segments(c1));
                    A4 = table_1.Contrast(table_1.ContourID == segments(c1));
                    A5 = table_1.Asymmetry(table_1.ContourID == segments(c1));
                    A6 = table_1.LineWidth(table_1.ContourID == segments(c1));
                    A7 = table_1.AngleOfNormal(table_1.ContourID == segments(c1));

                    B3 = table_1.Length(table_1.ContourID == segments(c2));
                    B4 = table_1.Contrast(table_1.ContourID == segments(c2));
                    B5 = table_1.Asymmetry(table_1.ContourID == segments(c2));
                    B6 = table_1.LineWidth(table_1.ContourID == segments(c2));
                    B7 = table_1.AngleOfNormal(table_1.ContourID == segments(c2));

                    A = [A, A3, A4, A5, A6, A7]; % Combine A with A3:A7
                    B = [B, B3, B4, B5, B6, B7]; % Combine B with B3:B7

                    fiber_pair = [segments(c1), segments(c2)];

                    % Update the vertical array of matching fiber segment overlaps
                    fiber_segment = vertcat(fiber_segment, fiber_pair);

                    % write down which conditions were satisified per segment
                    condition_quad = [condition_1, condition_2, ...
                                condition_3, condition_4, condition_5];
                    condition_segment = vertcat(condition_segment, ...
                                                condition_quad);

                    % merge the two contourID's (X&Y) coordinates together without
↪   duplicating points
                    fiber_union(c3).XY = [union(A, B, 'rows', 'stable')];
                    f_len = length(fiber_union(c3).XY); % length of the matched fiber
↪   segment

                    % Length of the new segments is going to be a mixture of the
                    % two fiber segments
                    L_A = unique(table_1.Length(table_1.ContourID == segments(c1)));
                    L_B = unique(table_1.Length(table_1.ContourID == segments(c2)));

                    % Fiber A contains all of fiber B
                    case_1 = (Lib.mode == 1) && (Lib.num_zero == 0);
                    % Fiber A contains the majority of fiber B
                    case_2 = (Lib.mode == 1) && (condition_2 == 1) && ...
                        (condition_3 == 1) && (condition_4 == 1);
                    % Fiber A contains the minority of fiber B
                    case_3 = (Lib.mode == 0) && (condition_2 == 1) && ...
                        (condition_3 == 1) && (condition_4 == 1);
                    % Fiber A does not contain fiber B
                    case_4 = (Lib.num_one == 0) && (condition_2 == 1) && ...
                        (condition_3 == 1) && (condition_4 == 1);

                    if case_1 == 1
                        % Max of the two fiber segments length
                        new_fiber_len = max([A3;B3]);
                    elseif case_2 == 1
                        overlap = Lib.num_one;
                        % If the majority of the points overlap, find the percentage
                        new_fiber_len = (L_A + L_B - ...
                            (overlap/length(A)*L_A + ...
                            overlap/length(B)*L_B)/2);
                    elseif case_3 == 1
                        overlap = Lib.num_one;
                        % If the majority of the points overlap, find the percentage
                        new_fiber_len = L_A + L_B - ...
                            (overlap/length(A)*L_A + ...
                            overlap/length(B)*L_B)/2;
                    elseif case_4 == 1
                    % if the two fibers don't overlap
                    new_fiber_len = L_A + L_B;
                else
                    % Average the two lengths
                    new_fiber_len = 0.5*(L_A + L_B);
                end

                % store the matching contourID with the coordinates
                fiber_union(c3).segment_match = fiber_pair;
                % adds a new ContourID number (max(ContourID) + 1)
                fiber_union(c3).New_ContourID = ones(f_len, 1) * (max(table_1.ContourID) + 1);
                if strcmp(table_1.Properties.VariableNames{1}, 'Var1')
                    % update the number Var1 number.  Some of the outputs have this.  If not, comment out
                    fiber_union(c3).Var1 = ones(f_len, 1).*table_1.Var1(end):table_1.Var1(end) + f_len - 1;
                end
                fiber_union(c3).Frame = ones(f_len, 1);
                fiber_union(c3).Pos_ = 1:f_len;
                fiber_union(c3).X = fiber_union(c3).XY(:, 1);
                fiber_union(c3).Y = fiber_union(c3).XY(:, 2);
                %; % Update new fiber length
                fiber_union(c3).Length = ones(f_len, 1)*new_fiber_len;
                fiber_union(c3).Contrast = fiber_union(c3).XY(:, 4);
                fiber_union(c3).Asymmetry = fiber_union(c3).XY(:, 5);
                fiber_union(c3).LineWidth = fiber_union(c3).XY(:, 6);
                fiber_union(c3).AngleOfNormal = fiber_union(c3).XY(:, 7);
                fiber_union(c3).Class(1:f_len) = {'new_fiber'};
                fiber_union(c3).Class = fiber_union(c3).Class(1:f_len)';

                % create a shortcut for the list
                fu = fiber_union(c3);
                % transpose the position
                fu.Pos_ = fu.Pos_';
                % If the attribute is in the CSV file add the info
                if strcmp(table_1.Properties.VariableNames{1}, 'Var1')
                    fu.Var1 = fu.Var1'; % Transpose the column
                    % new matching segment info
                    table_1_new_fiber_segment = table(fu.Var1, ...
                        fu.Frame, fu.New_ContourID, fu.Pos_, fu.X, ...
                        fu.Y, fu.Length, fu.Contrast, fu.Asymmetry, ...
                        fu.LineWidth, fu.AngleOfNormal, fu.Class);
                else
                    % If the attribute is not in the CSV file, move on without it
                    % new matching segment info
                    table_1_new_fiber_segment = table(fu.Frame, ...
                        fu.New_ContourID, fu.Pos_, fu.X, fu.Y, ...
                        fu.Length, fu.Contrast, fu.Asymmetry, ...
                        fu.LineWidth, fu.AngleOfNormal, fu.Class);
                end
                % stores the variable names to the new table for merging
                table_1_new_fiber_segment.Properties.VariableNames = table_1.Properties.VariableNames;
                % append new matching segment info to table1
                table_1 = [table_1;table_1_new_fiber_segment];

                % % Plot both segments that are being eliminated
```

```
                    % figure;
                    % imshow(img);
                    % hold on
                    % plot(Ax*x_scale + shift_x, Ay*y_scale + shift_y, 'r.', 'markersize', 5);
                    % plot(Bx*x_scale + shift_x, By*y_scale + shift_y, 'bo', 'markersize', 5);
                    % txt = {'\leftarrow A -s#', num2str(segments(c1)), '\leftarrow B -s#', num2str(segments(c2))};
                    % text(mean(Ax*x_scale) + shift_x, mean(Ay*x_scale) + shift_y, strcat(txt{1}, txt{2}));
                    % text(mean(Bx*x_scale) + shift_x, mean(By*x_scale) + shift_y, strcat(txt{3}, txt{4}));
                    %
                    % % Used for debugging
                    % fprintf('A ----- %f, B ----- %f, New Fiber #%d ----- %f\n', ...
                    % L_A, L_B, unique(fiber_union(c3).New_ContourID), ...
                    % new_fiber_len);


                    % If the length of segment_A is longer than segment_B get rid
                    % of the smaller segment (segment_B)
                    if(length(A) > length(B))
                        % Update the table with the new ContourID #
                        segments(c1) = max(table_1.ContourID);
                        % Delete the ID number from list 'B'
                        segments(c2) = [];
                        % start from the top of the list
                        % c2 = 1;

                    % If the two segments are identical
                    elseif (segments(c1) ~= segments(c2))
                        % Update the table with the new ContourID #
                        segments(c2) = max(table_1.ContourID);
                        % Delete the ID number from list 'A'
                        segments(c1) = [];
                        % start from the top of the list
                        % c1 = 1;
                    end
                    % restart from the top of the list
                    c1 = 1;
                    % c2 = 2;
                    % Update the matched pairs counter
                    c3 = c3 + 1;
                end

                % If the length of segments is 1 or 0, or the last iteration of the loop
                if (length(segments) <= 1) || (length(segments) == c2)
                    % If there are no more matches after the end of looping through the
                    % it is considered a 'lone fiber'
                    lone_fibers = [lone_fibers;segments(c1)];
                    % Delete the ID number from list 'A'
                    segments(c1) = [];
                    % restart from the top of the list
                    c1 = 1;
                    fprintf(['Segment # %.0f removed from the list ' ...
                        'of potential segments (%d)\n'], ...
                        lone_fibers(end), length(segments));
                    % If there are no more combinations that can be ...
                    % checked then all unique fibers have been ...
                    % identified and concatenated
                    check_2 = true;
                    if (length(segments) == 0) || (length(segments) == 1) % (c1 ==
↪   length(segments)) || (c1 > length(segments))
                        % If there are no more combinations that can
                        % be checked then all unique fibers have been
                        % identified and concatenated
                        check_1 = true;
                    end
                end

                if (condition_1 && condition_5 || ...
                        condition_2 && condition_3 && ...
                        condition_4 && condition_5)
                    % restart from the top of the list if a segment was removed
                    c2 = 2;
                else
                    % Update the iteration for while loop #2
                    c2 = c2 + 1;
                end
                count = count + 1; % Update the number of iterations
            end
            % c1 = c1 + 1; % Update the iteration for while loop #1 % We don't need
            % to update this because we are eliminating the c1 point if there are
            % not matches after each c2 iteration through all of the segments.  We
            % should probably eliminate the first while loop because it is
            % unnecessary to increment now in this 2.0 version of the code by
            % eliminating the c1 point.
        end
        toc

        %%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Plot the fiber segments that matched from the previous step
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % sort the fibers from the previous loop to color code by length
        combined_and_lone_fibers = [segments;lone_fibers];
        fiber_len_array = []; % zero array
        for i = 1:length(combined_and_lone_fibers)
            fiber_len = [combined_and_lone_fibers(i), ...
                mean(table_1.Length(table_1.ContourID == ...
                combined_and_lone_fibers(i)))];
            fiber_len_array = vertcat(fiber_len_array, fiber_len);
        end

        % Sort the fibers based on their length
        combined_and_lone_fibers = sortrows(fiber_len_array, 2);
        C = parula(length(combined_and_lone_fibers));
        % Overlay of the fibers and the original image
        h = figure;
        imshow(img);
        hold on
        for i = 1:length(combined_and_lone_fibers)
```

```
            % figure;
            % imshow(img);
            % hold on
            x1 = table_1.X(table_1.ContourID == combined_and_lone_fibers(i));
            y1 = table_1.Y(table_1.ContourID == combined_and_lone_fibers(i));
            % Plot dots instead of connected lines
            plot(x1*x_scale + shift_x, y1*y_scale + shift_y, '.', 'markersize', 5, 'color', C(i, :));
            % i % Plot the ID # i
            % txt = {'\leftarrow #', num2str(combined_and_lone_fibers(i))};
            % text(mean(x1*x_scale) + shift_x, mean(y1*x_scale) + shift_y, strcat(txt{1}, txt{2}));
            title('\bf True Fibers');
        end
        plot(xv, yv, 'r--', 'LineWidth', 1.5)
        plot(ILM.x, ILM.y, 'r--', 'LineWidth', 1.5)
        title('\bf True Fibers!');
        % Saves the figure as a Tif
        saveas(h, strcat(file_name_root, file_name_extension, '.tif'));

        % % Look at the matching fibers that were used to construct the complete
        % % fiber
        % for i = 1:length(fiber_segment)
        %     figure;
        %     imshow(img);
        %     hold on
        %     x1 = table_1.X(ContourID == fiber_segment(i, 1));
        %     y1 = table_1.Y(ContourID == fiber_segment(i, 1));
        %     x2 = table_1.X(ContourID == fiber_segment(i, 2));
        %     y2 = table_1.Y(ContourID == fiber_segment(i, 2));
        %     plot(x1*x_scale, y1*y_scale, 'r.', 'markersize', 5);
        %     plot(x2*x_scale, y2*y_scale, 'bo', 'markersize', 10);
        % end

        % filtered out contour ID's that were too small
        % ID_eliminated = unique(table_1.ContourID(table_1.Length < length_threshold));
        % filtered out contour ID's that were too small
        % ID_eliminated = unique(table_1.ContourID((table_1.Length < fiber_min_length)));

        %%
        filtered_fibers = length(short_segments);
        fprintf('Filtered out %d fiber segments\n', filtered_fibers);
        fprintf('Remaining eligible fibers = %d fibers\n', ...
            length(segments));
        fprintf('Total unique fibers = %d fibers\n', ...
            length(combined_and_lone_fibers));



        % Loop over all the current IDs that satisfy the criteria
        for i = 1:length(combined_and_lone_fibers)
            cur_x = table_1.X(find(table_1.ContourID == ...
                combined_and_lone_fibers(i)));
            cur_y = table_1.Y(find(table_1.ContourID == ...
                combined_and_lone_fibers(i)));
            %     cur_xRes = cur_x*x_scale + shift_x;
            %     cur_yRes = cur_y*y_scale + shift_y;
            Filt_Fibers_XY = [cur_x, cur_y];
            %     Filt_Fibers_XYRes = [cur_xRes, cur_yRes];
            Filt_Fibers(i).Length = unique(table_1.Length(table_1.ContourID == ...
                combined_and_lone_fibers(i)));
            Filt_Fibers(i).Width = table_1.LineWidth(table_1.ContourID == ...
                combined_and_lone_fibers(i));
            Filt_Fibers(i).ID = combined_and_lone_fibers(i);
            Filt_Fibers(i).Area = Filt_Fibers(i).Length.*Filt_Fibers(i).Width; % Area
↪   of fibers

%              sort_cur_x = sort(table_1.X(combined_and_lone_fibers(i)));
%              sort_cur_y = sort(table_1.Y(combined_and_lone_fibers(i)));
%              angle = []; % clears the array during each loop
%              slope = []; % array of slopes
%              for j = 1:length(cur_x)-1
%                  % Consider using the polyfit
%                  numerator = (cur_y(j+1) - cur_y(j));
%                  denominator = (cur_x(j+1) - cur_x(j));
%                  % Calculates the fiber angle for each successive point in the fiber
%                  %  angle(j) = atan(numerator/denominator)*180/pi;
%                  % Calculates the fiber angle for each successive point in the fiber
%                  angle_calc = atan(numerator/denominator)*180/pi;
%                  %          if isnan(angle_calc)
%                  %              j
%                  %              fprintf('isnan\n');
%                  %              continue % bypass the angle that doesnt
%                  %          elseif (numerator == 0 && denominator == 0)
%                  if (numerator == 0 && denominator ==0)
%                      continue % bypass the angle that doesn't exist
%                  elseif (denominator == 0)
%                      %angle(j) = 90; % perpendicular line segments
%                      angle = [angle;90];
%                      continue
%                  else
%                      %slope(j) = numerator/denominator;
%                      slope = [slope;numerator/denominator];
%                      if slope(end) < 0 % slope(j) < 0
%                          % slopes are negative so add 180 degrees
%                          %angle(j) = angle(j) + 180;
%                          angle = [angle;angle_calc + 180];
%                      end
%                  end
%              end

            % Calculate slope & y-intercept from linear fit
            [F] = Least_Squares(Filt_Fibers_XY);
            %Slope
            Filt_Fibers(i).slope = F(2);
            % inverse tangent of the slope
            Filt_Fibers(i).Angle = -atan(F(2))*180/pi;
            % Clear the dataset from the array for the next iteration
            Filt_Fibers_XY = [];

            % % average the slope for each individual contour ID
            % Filt_Fibers(i).slope = mean(slope);
            % %ILM_angle - ...
            % Filt_Fibers(i).Angle = angle;
            % % Mean angle of each countour ID
            % Filt_Fibers(i).mean_Angle = mean(angle);
        end

        for i = 1:length(combined_and_lone_fibers)
            % Puts each mean angle into an array
            filt_ang(i) = Filt_Fibers(i).Angle;
            % Calculates mean fiber length
            filt_len(i) = Filt_Fibers(i).Length;
            % Average width of the fiber and puts it into an array
            filt_wid(i) = mean(Filt_Fibers(i).Width);
            % Calculates the average fiber area (length*width of pixels)
            filt_area(i) = mean(Filt_Fibers(i).Area);
            % Number of points in each contour ID# and puts it into an array
            filt_num(i) = length(Filt_Fibers(i).Width);
            % Average slope of each contour ID#
            filt_slo(i) = Filt_Fibers(i).slope;
        end

        for i = 1:length(short_segments)
            cur_x = table_1.X(find(table_1.ContourID == ...
                short_segments(i)));
            cur_y = table_1.Y(find(table_1.ContourID == ...
                short_segments(i)));
            %     cur_xRes = cur_x*x_scale + shift_x;
            %     cur_yRes = cur_y*y_scale + shift_y;
            No_Filt_Fibers_XY = [cur_x, cur_y];
            %     No_Filt_Fibers(i).XYRes = [cur_xRes, cur_yRes];
            No_Filt_Fibers(i).Length = unique(table_1.Length(table_1.ContourID == ...
                short_segments(i)));
            No_Filt_Fibers(i).Width = table_1.LineWidth(table_1.ContourID == ...
                short_segments(i));
            No_Filt_Fibers(i).ID = short_segments(i);

            % Calculate slope & y-intercept from linear fit
            [F] = Least_Squares(No_Filt_Fibers_XY);
            %Slope
            No_Filt_Fibers(i).slope = F(2);
            % inverse tangent of the slope
            No_Filt_Fibers(i).Angle = atan(F(2))*180/pi;
            % Clear the dataset from the array for the next iteration
            No_Filt_Fibers_XY = [];

            % % average the slope for each individual contour ID
            % No_Filt_Fibers(i).slope = mean(slope);
            % %ILM_angle - ...
            % No_Filt_Fibers(i).Angle = angle;
            % % Mean angle of each countour ID
            % % No_Filt_Fibers(i).mean_Angle = mean(angle);
        end

        for i = 1:length(short_segments)
            % Puts each mean angle into an array
            No_filt_ang(i) = No_Filt_Fibers(i).Angle;
            % Puts each fiber length into an array
            No_filt_len(i) = No_Filt_Fibers(i).Length;
            % Average width of the fiber and puts it into an array
            No_filt_wid(i) = mean(No_Filt_Fibers(i).Width);
            % Number of points in each contour ID# and puts it into an array
            No_filt_num(i) = length(No_Filt_Fibers(i).Width);
            % Average slope of each contour ID#
            No_filt_slo(i) = No_Filt_Fibers(i).slope;
        end

        % %%
        % % Plot individual fibers on a single sheet
        % %
        % % Do not run this on a real image
        % %
        % C = hsv(length(segments)); % Color array for the fibers
        % for i = 1:length(segments)
        %     figure
        %     imshow(img);
        %     hold on
        %     plot(Filt_Fibers(i).XYRes(:, 1), Filt_Fibers(i).XYRes(:, 2), '.',
↪  'Color', C(i, :));
        % end
        % title('\bf Filterd image', 'fontsize', 18);
        % %%
        % [~, index] = sortrows([Filt_Fibers.Length].');
        % Filt_Fibers = Filt_Fibers(index);
        % clear index; % Sort the Filt_Fibers by Length
        %
        %
        % % Plot individual fibers on the same sheet just pausing for half a second
        % C = hsv(length(segments)); % Color array for the fibers
        % figure
        % imshow(img);
        % hold on
        % for i = 1:length(segments)
        %     waitbar(i/length(segments));
        %     plot(Filt_Fibers(i).XYRes(:, 1), Filt_Fibers(i).XYRes(:, 2), '.',
↪  'Color', C(i, :));
        %     %     pause(0.01)
        % end
        % title('\bf Filterd image', 'fontsize', 18);
        %
        % %%
        % [~, index] = sortrows([No_Filt_Fibers.Length].');
        % No_Filt_Fibers = No_Filt_Fibers(index);
        % clear index; % Sort the Filt_Fibers by Length
        %
        %
        % % Plot individual fibers on the same sheet just pausing for half a second
        % C = hsv(length(short_segments)); % Color array for the fibers
        % figure
        % imshow(img);
        % hold on
        % for i = 1:length(short_segments)
        %     waitbar(i/length(short_segments));
        %     plot(No_Filt_Fibers(i).XYRes(:, 1), No_Filt_Fibers(i).XYRes(:, 2), '.',
↪  'Color', C(i, :));
        %     %     pause(0.01)
        % end
        % title('\bf Non-Filterd image', 'fontsize', 18);


        % The combined_and_lone_fibers list needs to be sorted by fiber length
        % before calculating attributes such as slope, and angle
        for i = 1:length(combined_and_lone_fibers)
            % unique length of the connected fibers *1000 for nanometers
            len = unique(table_1.Length(table_1.ContourID ==
↪   combined_and_lone_fibers(i)));
            % converted average angle from y-axis to the x-axis -pi/2
            angle = (mean(table_1.AngleOfNormal(table_1.ContourID == ...
                combined_and_lone_fibers(i)))-pi)*180/pi;
            % angle from calculating the inverse tangent of the slope
            calc_ang = filt_ang(i);
            %difference in angle
            difference = angle - calc_ang;
            % density of collagen fibers / ilm length
            density(i) = filt_area(i)/ILM_length;
            fprintf(['Fiber # %d -- length = %.4f nanometers, -- ' ...
                'avg. angle RD = %.2f degrees, -- angle Calc = ' ...
                '%.2f degrees, -- angle diff %.2f\n'], ...
                combined_and_lone_fibers(i), len, angle, calc_ang, ...
                difference);
        end
        % density of collagen fibers / ilm length
        fprintf('Collagen fiber density = %f microns\n', sum(density));

        % Plots the histogram of the calculated angles
        % figure
        % hist(filt_ang);
        % title('\bf Calculation of fiber angles');
        % fprintf(['Collagen fiber angle is %f \n ' ...
        %     '(relative to the x-axis)\n'], mean(filt_ang));

        % Plots the angle vs. fiber segment length
        %figure
        %plot(filt_ang, filt_len, '.');
        %set(gca, 'XDir', 'reverse');
        %xlabel('\bf Fiber Angle');
        %ylabel('\bf Fiber Length');
        %title('\bf Fiber Angle vs. Length');

        % Plots the angle vs. fiber segment length on a polar grid
        %figure
        % plot(ang, len, '.');
        % pax = gca; % 2018a
        % pax.ThetaAxisUnits = 'radians'; % 2018a
        %polarplot(filt_ang*pi/180, filt_len, '.')
        % xlabel('\bf Fiber Angle'); % 2018a
        % ylabel('\bf Fiber Length'); % 2018a
        % axis([min(ang), max(ang), min(len), max(len)]);
        %title('\bf Fiber Angle vs. Length');

        % Plot each unique fiber with a different color
        % figure
        % imshow(img);
        % hold on
        % C = hsv(length(unique(combined_and_lone_fibers)));
        % for i = 1:length(combined_and_lone_fibers)
```

```matlab
        %     plot(All_Fibers(i).XYRes(:, 1), All_Fibers(i).XYRes(:, 2), '.', 'color', C(i, :), 'linewidth', 2);
        %     hold on;
        % end
        % axis image;
        % title('\bf Unique ContourID fiber identification');

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Plot the histrogram of the image
        % figure
        % if synthetic == true
        %     img2 = rgb2gray(img); % Converts the RGB image to grayscale
        %     % [counts, grayLevels] = imhist(img, 256);
        %     imhist(img2); % Looks at the histogram of pixel intensitites
        % else
        %     % [counts, grayLevels] = imhist(img, 256);
        %     imhist(img); % Looks at the histogram of pixel intensitites
        % end
        % title('\bf Histogram of TEM image pixel intensities');

        % Plot the contour map for the image overlayed with the detected fibers
        % h = figure;
        % image(img)
        % hold on
        % contourf(img, 10)
        % axis image
        % colormap gray
        % for i = 1:length(combined_and_lone_fibers)
        %       %  figure;
        %       %  imshow(img);
        %          hold on
        %     x1 = table_1.X(table_1.ContourID == combined_and_lone_fibers(i));
        %     y1 = table_1.Y(table_1.ContourID == combined_and_lone_fibers(i));
        %     plot(x1*x_scale + shift_x, y1*y_scale + shift_y, '.', 'markersize', 5, 'color', C(i, :)); % Plot dots instead of connected lines
        %  %   txt = {'\leftarrow #', num2str(combined_and_lone_fibers(i))}; % i % Plot the ID # i
        %   %  text(mean(x1*x_scale) + shift_x, mean(y1*x_scale) + shift_y, strcat(txt{1}, txt{2}));
        % %     title('\bf True Fibers');
        % end
        %title('\bf True Fibers overlayed on a contour filled plot!');
        %saveas(h, strcat(file_name_root, file_name_extension, '_contour.tif')); % Saves the figure as a Tif

        fprintf(fileID, 'Total unique fibers = %d fibers\n', ...
            length(combined_and_lone_fibers));
        fprintf(fileID, ...
            'Width of the rectangle ILM measurement = %d microns\n', ...
            ILM_length);
        fprintf(fileID, ...
            'ILM angle is %f degrees \n (relative to the x-axis)\n', ...
            ILM_angle);
        fprintf(fileID, ...
            'Average ILM thickness is %f nanometers \n', ...
            ILM_thickness);
        fprintf(fileID, 'Collagen fiber count density = %f \n', ...
            length(combined_and_lone_fibers)/ILM_length);
        fprintf(fileID, ...
            ['Abs Mean Collagen fiber angle is %f \n ' ...
            '(relative to the x-axis)\n'], ...
            nanmean(abs(filt_ang)));
        fprintf(fileID, ...
            ['Abs Median Collagen fiber angle is %f \n ' ...
            '(relative to the x-axis)\n'], ...
            nanmedian(abs(filt_ang)));
        fprintf(fileID, ...
            ['Abs Mean Collagen fiber angle is %f \n ' ...
            '(relative to the ILM)\n'], ...
            nanmean(abs(filt_ang-ILM_angle)));
        fprintf(fileID, ...
            ['Abs Median Collagen fiber angle is %f \n ' ...
            '(relative to the ILM)\n'], ...
            nanmedian(abs(filt_ang-ILM_angle)));

        %fprintf(fileID, 'ILM slope = %f\n', ILM_slope);
        %fprintf(fileID, 'ILM length = %f microns\n', ILM_length);

        %fprintf(fileID, 'Mimimum fiber length is %f microns\n', fiber_min_length);

        %fprintf(fileID, 'Filtered out %d fiber segments\n', filtered_fibers);
        %fprintf(fileID, 'Remaining eligible fibers = %d fibers\n',
↪   length(segments));

        % for i = 1:length(combined_and_lone_fibers)
        %     fprintf(fileID, 'Fiber # %d -- length = %.4f nanometers, -- avg. angle
↪   RD = %.2f degrees, -- angle Calc = %.2f degrees, -- angle diff %.2f\n',
↪   combined_and_lone_fibers(i), len, angle, calc_ang, difference);
        % end
        %fprintf(fileID, 'Collagen fiber density = %f microns\n', sum(density)); %
↪   density of collagen fibers / ilm length

        fprintf(fileID, 'Average collagen fiber length = %f microns\n', ...
            mean(filt_len));
        fclose(fileID); % close the txt file for the output information

        % Saves the new table with Original Fibril & New Fibril data
        writetable(table_1, strcat(file_name_root, file_name_extension, ...
            '_Original_and_New_FibrilData', '.csv'))


    case 'No'
        %Calculate only ILM thickness if no collagen
        figure
        imshow(img);
        % Indicate the five points on the ILM used for thickness measurements
        for i = 1:5
            f = msgbox(['Select the first two points that define ' ...
                'the ILM thickness'], 'ILM');
            %     pause(1);
            [ILM_thick(i).x, ILM_thick(i).y] = ginput(2);
            hold on
            plot(ILM_thick(i).x, ILM_thick(i).y, 'g-o', 'linewidth', 1);
            % Pythogrean theorem
            ILM_thick(i).measurement = sqrt((ILM_thick(i).x(1) - ILM_thick(i).x(2))^2
↪   + ...
```

```
                (ILM_thick(i).y(1) - ILM_thick(i).y(2))^2);
            delete(f); % Delete the message box
        end
        for i = 1:5
            ILM_measurement(i) = ILM_thick(i).measurement;
        end
        L{4} = 'ILM thickness measurements';
        axis image;
        ILM_thickness = mean(ILM_measurement)/x_scale*1000;
        fprintf('Average ILM thickness is %f nanometers \n', ILM_thickness);
end
```

1.5 Human Data Analysis

Script 4: *Python script analyzes human data, performs statistics, and creates figures.*

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 23 21:48:15 2020

@author: Kiffer Creveling
"""

import pandas as pd
import os
import numpy as np
import seaborn as sns
from statannot import add_stat_annotation
import matplotlib.pyplot as plt
from matplotlib.patches import PathPatch
plt.rcParams['figure.figsize'] = [16, 10]
from scipy import stats
import pdb

# In[Functions]

# fcn for plotting
def yfit(x):
    return slope*x + intercept

# In[Read values from Database]
""" Read from the database """

df = pd.read_csv('JMP_Data.csv') # Data from JMP
df = pd.read_excel('Human Data Paper 2 TEM only (Updated Jul 10 2020).xlsx',
                   engine='openpyxl')
df = pd.read_excel('Human Data Paper 2 TEM only (Updated April 17 2021).xlsx',
                   engine='openpyxl')

""" Simplification of code """
SF = 'StatisticsFigures' # Figure directory
TMD = 'TEM Mean Density'
TMA = 'TEM Mean Angle'
TAA = 'TEM Angle ABS'
```

```python
ILM = 'ILM Thickness (nm)'
FL = 'Fiber Length (um)'
MPF = 'Maximum peel force (mN)'
mpf_mN = 'Max peel force (mN)'
R = 'Region'
Eq = 'Equator'
Po = 'Posterior'
AG = 'AgeGroup'
A60 = 'Age60'
Aleq60 = r'Age $\leq$ 60'
Ag60 = 'Age $>$ 60'
A = 'Age'
MN = 'Max [N]'
MmN = 'Max [mN]'
SSN = 'SS [N]'
SSmN = 'SS [mN]'

# Plot attributes (labels, etc)
A_yrs = 'Age (yr.)'
A_G = 'Age Group (yr.)'
DensityUnit = (r'Collagen Fibril Density
↪  $\left(\frac{\mathrm{\#~of~fibrils}}{\mathrm{ILM~length~(nm)}}\right)$')
FibrilLengthUnit = r'Collagen Fibril length ($\mu$m)'
OrientationUnit = r'Collagen Fibril Angle Relative to the ILM $(^{\circ})$'

# convert from N to mN
df[mpf_mN] = df[MN]*1000
df[SSmN] = df[SSN]*1000

# Exclude the cells that have duplicates or have been exculded due to
# video analysis
df = df[df['Excluded'] != 'yes']

# In[Create AgeGroup bins]
bins = [30, 40, 50, 60, 70, 80, 90]
labels = ['30-39', '40-49', '50-59', '60-69','70-79', '80-89']
# Create binned AgeGroups

df[AG] = pd.cut(df[A], bins, labels=labels, right=False)

bins = [0, 60, 90]
labels = [Aleq60, Ag60]
# Create binned AgeGroups
df[A60] = pd.cut(df[A], bins, labels=labels, right=True)

# In[Pivot Table]
# Simplify pivot table output

pvtOut = {'count', np.median, np.mean, np.std} # pivot table outputs

# In[Plots]

standardError = 68 # Used for confidence intervals

sns.set_theme(context='paper', style='darkgrid', palette="Paired",
              font_scale=2)
custom_style = {'axes.facecolor': 'white',
                 'axes.edgecolor': 'black',
                 'axes.grid': False,
                 'axes.axisbelow': True,
                 'axes.labelcolor': 'black',
                 'figure.facecolor': 'white',
                 'grid.color': '.8',
                 'grid.linestyle': '-',
                 'text.color': 'black',
                 'xtick.color': 'black',
                 'ytick.color': 'black',
                 'xtick.direction': 'out',
                 'ytick.direction': 'out',
                 'lines.solid_capstyle': 'round',
                 'patch.edgecolor': 'w',
                 'patch.force_edgecolor': True,
                 'image.cmap': 'rocket',
                 'font.family': ['sans-serif'],
                 'font.sans-serif': ['Arial', 'DejaVu Sans', 'Liberation Sans',
                                     'Bitstream Vera Sans', 'sans-serif'],
                 'xtick.bottom': True,
                 'xtick.top': False,
                 'ytick.left': True,
                 'ytick.right': False,
                 'axes.spines.left': True,
                 'axes.spines.bottom': True,
                 'axes.spines.right': False,
                 'axes.spines.top': False}
# White background with ticks and black border lines, Turns grid off
ax = sns.set_style(rc=custom_style)

def boxPlotBlackBorder(ax):
    # iterate over boxes in the plot to make each line black
    for i,box in enumerate(ax.artists):
        box.set_edgecolor('black')
        # box.set_facecolor('white')

        # iterate over whiskers and median lines
        for j in range(6*i, 6*(i+1)):
            ax.lines[j].set_color('black')

def smartPlot(data=None, x=None, y=None, hue=None, hue_order=None,
              addBoxPair=None, ci=None, errcolor=None, capsize=None,
              plot=None, test=None, sigLoc=None, text_format=None,
              line_offset=None, line_offset_to_box=None, line_height=None,
              fontsize=None, legLoc=None, verbose=None, xlabel=None,
              ylabel=None, legendTitle=None, figName=None, folderName=None,
              dataPoints=None):

    # barplot
    f, ax = plt.subplots()

    if plot == 'barplot':
        ax = sns.barplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order,
                         ci=ci, errcolor=errcolor, capsize=capsize)

    elif plot == 'boxplot':
        ax = sns.boxplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order)

    # Statistical test for differences
    x_grps = list(data[x].unique()) # List of groups
    if hue != None:
        # Create combinations to compare
        box_pairs_1 = [((x_grps_i, hue_order[0]),
                        (x_grps_i, hue_order[1]))
                       for x_grps_i in x_grps]
        box_pairs = box_pairs_1

        if addBoxPair != None:
            # Additional box pairs
            box_pairs =  box_pairs_1 + addBoxPair

    elif hue_order != None:
        box_pairs = [(hue_order[0], hue_order[1])]

    #Stats results and significant differences (SR)
    SR = add_stat_annotation(ax, plot=plot, data=data, x=x, y=y, hue=hue,
                             hue_order=hue_order, box_pairs=box_pairs,
                             test=test, loc=sigLoc, text_format=text_format,
                             verbose=verbose, comparisons_correction=None,
                             line_offset=line_offset,
                             line_offset_to_box=line_offset_to_box,
                             line_height= line_height,
                             fontsize=fontsize) # 'bonferroni'

    if plot == 'boxplot':
        boxPlotBlackBorder(ax) # Make borders black


    if dataPoints == True:
        # Add data points to the box plot
        sns.stripplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order,
                      color='.5', size=5, linewidth=1, dodge=True)

        # gather plot attributes for legends
        handles, labels = ax.get_legend_handles_labels()

        if hue != None:
            l = plt.legend(handles[0:2], labels[0:2], title=legendTitle)

    else:
        if hue != None:
            ax.legend(loc=legLoc).set_title(legendTitle)

    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax = sns.despine() # takes the lines off on the right and top of the graph

    if folderName != None:
        # If a new folder name is given, put the files there

        # New file path
        NP = os.path.join(SF, folderName)

        # Create folder if it doesn't exist
        os.makedirs(NP, exist_ok=True)

    else:
        # Put the file in the same folder
        NP = SF

    f.savefig(os.path.join(NP, '{}.pdf'.format(figName)),
              bbox_inches='tight')
    plt.close()

# Special spacing

def adjust_box_widths(g, fac):
    """
    Adjust the withs of a seaborn-generated boxplot.
    """

    # iterating through Axes instances
    for ax in g.axes:

        # iterating through axes artists:
        for c in ax.get_children():

            # searching for PathPatches
            if isinstance(c, PathPatch):
                # getting current width of box:
                p = c.get_path()
                verts = p.vertices
                verts_sub = verts[:-1]
                xmin = np.min(verts_sub[:, 0])
                xmax = np.max(verts_sub[:, 0])
                xmid = 0.5*(xmin + xmax)
                xhalf = 0.5*(xmax - xmin)

                # setting new width of box
                xmin_new = xmid - fac*xhalf
                xmax_new = xmid + fac*xhalf
                verts_sub[verts_sub[:, 0] == xmin, 0] = xmin_new
                verts_sub[verts_sub[:, 0] == xmax, 0] = xmax_new

                # setting new width of median line
                for l in ax.lines:
                    if np.all(l.get_xdata() == [xmin, xmax]):
                        l.set_xdata([xmin_new, xmax_new])

# In[TEM mean density by age +/- 60 and region]

""" TEM mean density by age +/- 60 and region """

pivotTEM_MeanDensityAgeGroup60 = pd.pivot_table(df, values=TMD,
                                                index=[A60, R],
                                                aggfunc=pvtOut)

print('pivotTEM_MeanDensityAgeGroup60')
print(pivotTEM_MeanDensityAgeGroup60)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_MeanDensityAgeGroup60.to_latex(index=True, na_rep='-',
                                              escape=False,
                                              float_format="{:0.3f}".format))

Folder = 'Density_Age60Region'
```

```python

# Barplot
smartPlot(data=df, x=A60, y=TMD, hue=R, hue_order=[Eq, Po], ci='sd',
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=DensityUnit, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=A60, y=TMD, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', text_format='star', sigLoc='outside',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=DensityUnit,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=A60, y=TMD, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=DensityUnit,
          legendTitle=R, figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)


# In[TEM mean density grouped by region]

""" TEM mean density """

pivotTEM_MeanDensityRegion = pd.pivot_table(df, values=TMD, index=[R],
                                            aggfunc=pvtOut)

print('pivotTEM_MeanDensityRegion')
print(pivotTEM_MeanDensityRegion)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_MeanDensityRegion.to_latex(index=True, na_rep='-',
                                          escape=False,
                                          float_format="{:0.3f}".format))

Folder = 'Density_Region'

# Barplot
smartPlot(data=df, x=R, y=TMD, hue=None, hue_order=[Eq, Po], ci='sd',
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=R, ylabel=DensityUnit, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=R, y=TMD, hue=None, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=R, ylabel=DensityUnit,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=R, y=TMD, hue=None, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=R, ylabel=DensityUnit,
          legendTitle=R, figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)

# matched_pairs student's t-test
dfTMD = df[df[TMD].notna()]

dfMP = dfTMD[dfTMD.duplicated(['MatchingID'], keep=False)]
f, p = stats.ttest_rel(dfMP[TMD][dfMP[R] == Eq],
                       dfMP[TMD][dfMP[R] == Po])

print(f, p, "Matched Pairs Student's t-test")

f, p = stats.ttest_ind(dfTMD[TMD][dfTMD[R] == Eq],
                       dfTMD[TMD][dfTMD[R] == Po])

print(f, p, "Student's t-test")

# In[TEM mean density grouped by age group decade and region]

pivotTEM_MeanDensity = pd.pivot_table(df, values=TMD, index=[R, AG],
                                      aggfunc=pvtOut)

print('pivotTEM_MeanDensity')
print(pivotTEM_MeanDensity)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_MeanDensity.to_latex(index=True, na_rep='-', escape=False,
                                    float_format="{:0.3f}".format))

Folder = 'Density_AgeDecadeRegion'

# Barplot
smartPlot(data=df, x=AG, y=TMD, hue=R, hue_order=[Eq, Po], ci='sd',
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=DensityUnit, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=AG, y=TMD, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=DensityUnit,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=AG, y=TMD, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=DensityUnit,
          legendTitle=R,
          figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)

# In[ILM thickness vs region age +/- 60]

""" TEM ILM thickness vs region age +/- 60 """

pivotTEM_ILM_ThicknessAge60 = pd.pivot_table(df, values=ILM, index=[A60, R],
                                             aggfunc=pvtOut)

print('pivotTEM_ILM_ThicknessAge60')
print(pivotTEM_ILM_ThicknessAge60)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_ILM_ThicknessAge60.to_latex(index=True, na_rep='-',
                                           escape=False,
                                           float_format="{:0.3f}".format))

Folder = 'ILM_Age60Region'

# Barplot
smartPlot(data=df, x=A60, y=ILM, hue=R, hue_order=[Eq, Po], ci='sd',
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=ILM, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=A60, y=ILM, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=ILM,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=A60, y=ILM, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=ILM,
          legendTitle=R, figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)


# In[ILM thickness vs region age group]

""" ILM thickness vs region and age group """

pivotTEM_ILM_Thickness = pd.pivot_table(df, values=ILM, index=[AG, R],
                                        aggfunc=pvtOut)

```

```python
print('pivotTEM_ILM_Thickness')
print(pivotTEM_ILM_Thickness)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_ILM_Thickness.to_latex(index=True, na_rep='-',
                                      escape=False,
                                      float_format="{:0.3f}".format))

Folder = 'ILM_Region'

# Barplot
smartPlot(data=df, x=AG, y=ILM, hue=R, hue_order=[Eq, Po], ci=68,
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=ILM, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=AG, y=ILM, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=ILM,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=AG, y=ILM, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=ILM,
          legendTitle=R,
          figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)


# In[ILM fiber length vs region age group decade]

""" TEM ILM fiber length """

pivotTEM_FiberLength = pd.pivot_table(df, values=FL, index=[AG, R],
                                      aggfunc=pvtOut)

print('pivotTEM_FiberLength')
print(pivotTEM_FiberLength)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_FiberLength.to_latex(index=True, na_rep='-',
                                    escape=False,
                                    float_format="{:0.3f}".format))

Folder = 'FibrilLength_AgeDecadeRegion'

# Barplot
smartPlot(data=df, x=AG, y=FL, hue=R, hue_order=[Eq, Po], ci=68,
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=FibrilLengthUnit, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=AG, y=FL, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=FibrilLengthUnit,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=AG, y=FL, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=FibrilLengthUnit,
          legendTitle=R,
          figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)


# In[ILM fiber length vs region age group +/- 60]

""" TEM ILM fiber length """

pivotTEM_FiberLengthAge60 = pd.pivot_table(df, values=FL, index=[A60, R],
                                           aggfunc=pvtOut)

print('pivotTEM_FiberLengthAge60')
print(pivotTEM_FiberLengthAge60)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_FiberLengthAge60.to_latex(index=True, na_rep='-',
                                         escape=False,
                                         float_format="{:0.3f}".format))

Folder = 'FibrilLength_Age60Region'

# Barplot
smartPlot(data=df, x=A60, y=FL, hue=R, hue_order=[Eq, Po], ci=68,
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=FibrilLengthUnit, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=A60, y=FL, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=FibrilLengthUnit,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=A60, y=FL, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=FibrilLengthUnit,
          legendTitle=R,
          figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)

# In[TEM Absolute Angle by age +/- 60 and region]

""" TEM Absolute Angle """

pivotTEM_MeanAngleABSAgeGroup60 = pd.pivot_table(df, values=TAA,
                                                 index=[A60, R],
                                                 aggfunc=pvtOut)

print('pivotTEM_MeanAngleABSAgeGroup60')
print(pivotTEM_MeanAngleABSAgeGroup60)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_MeanAngleABSAgeGroup60.to_latex(index=True, na_rep='-',
                                               escape=False,
                                               float_format="{:0.3f}".format))

Folder = 'ABSAngle_Age60Region'

# Barplot
smartPlot(data=df, x=A60, y=TAA, hue=R, hue_order=[Eq, Po], ci=68,
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnit, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=A60, y=TAA, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnit,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=A60, y=TAA, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnit,
          legendTitle=R,
          figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)


# In[TEM angle]

pivotTEM_MeanAngle = pd.pivot_table(df, values=TMA, index=[R, AG],
                                    aggfunc=pvtOut)

```

```python
print('pivotTEM_MeanAngle')
print(pivotTEM_MeanAngle)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_MeanAngle.to_latex(index=True, na_rep='-',
                                  escape=False,
                                  float_format="{:0.3f}".format))

OrientationUnitNoAbs = r'ILM angle $(^{\circ})$'
Folder = 'Angle_AgeRegion'

# Barplot
smartPlot(data=df, x=AG, y=TMA, hue=R, hue_order=[Eq, Po], ci=68,
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnitNoAbs, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=AG, y=TMA, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnitNoAbs,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=AG, y=TMA, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnitNoAbs,
          legendTitle=R,
          figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)

# In[TEM ABS angle by age decade group and region]

pivotTEM_MeanAngleABS = pd.pivot_table(df, values=TAA, index=[R, AG],
                                       aggfunc=pvtOut)
print('pivotTEM_MeanAngleABS')
print(pivotTEM_MeanAngleABS)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_MeanAngleABS.to_latex(index=True, na_rep='-',
                                     escape=False,
                                     float_format="{:0.3f}".format))

Folder = 'ABSAngle_AgeDecadeRegion'

# Barplot
smartPlot(data=df, x=AG, y=TAA, hue=R, hue_order=[Eq, Po], ci=68,
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnit, legendTitle=R,
          figName='BarPlot', folderName=Folder)
```

```python

# Boxplot
smartPlot(data=df, x=AG, y=TAA, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnit,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=AG, y=TAA, hue=R, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=A_G, ylabel=OrientationUnit,
          legendTitle=R,
          figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)

# In[TEM absolute angle by region]

pivotTEM_MeanAngleABSRegion = pd.pivot_table(df, values=TAA, index=[R],
                                             aggfunc=pvtOut)

print('pivotTEM_MeanAngleABSRegion')
print(pivotTEM_MeanAngleABSRegion)
# Add the index groups and convert NaN's to "-"'s
print(pivotTEM_MeanAngleABSRegion.to_latex(index=True, na_rep='-',
                                           escape=False,
                                           float_format="{:0.3f}".format))

Folder = 'ABSAngle_Region'

# Barplot
smartPlot(data=df, x=R, y=TAA, hue=None, hue_order=[Eq, Po], ci=68,
          errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
          sigLoc='outside', text_format='star', line_offset=0.0,
          line_offset_to_box=0.0, line_height=0.015, fontsize='small',
          legLoc='best', verbose=2,
          xlabel=R, ylabel=OrientationUnit, legendTitle=R,
          figName='BarPlot', folderName=Folder)

# Boxplot
smartPlot(data=df, x=R, y=TAA, hue=None, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=R, ylabel=OrientationUnit,
          legendTitle=R, figName='BoxPlot', folderName=Folder)

# Boxplot with data
smartPlot(data=df, x=R, y=TAA, hue=None, hue_order=[Eq, Po], plot='boxplot',
          test='t-test_ind', sigLoc='outside', text_format='star',
          line_offset=0.0, line_offset_to_box=0.0, line_height=0.015,
          fontsize='small', legLoc='best', verbose=2,
          xlabel=R, ylabel=OrientationUnit,
          legendTitle=R,
          figName='BoxPlotWithData', folderName=Folder,
          dataPoints=True)

# In[ILM thickness vs age regression]

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
# dict(Equator="r", Posterior="b") , 'color':'black', 'color':'blue'
ax = sns.lmplot(x=A, y=ILM, hue=R, markers=["o", "x"], data=df,
                legend_out=False, fit_reg=True, height=5, aspect=1.6,
                palette="Set1", truncate=False, ci=95, line_kws={'lw':0})
ax.set(ylabel=ILM, xlabel=A_yrs)

# Remove all NaN's from the data for regressions

# remove nans from ILM thickness
df_no_Nan = df.dropna(subset=[ILM])

# linear regressions for fitting
x = df_no_Nan[A][df_no_Nan[R] == Eq]
y = df_no_Nan[ILM][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
plt.text(80, yfit(80) + 20, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# linear regressions for fitting
x = df_no_Nan[A][df_no_Nan[R] == Po]
y = df_no_Nan[ILM][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
plt.text(75, yfit(75) + 20, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# Axis limits
ax.set(ylim=(0, None))
ax.set(xlim=(None, None))

# New path
NP = os.path.join(SF, 'ILM_vs_Age')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

ax.savefig(os.path.join(NP, 'Regression.pdf'), bbox_inches='tight')
plt.close()

# In[Max peel force vs ILM thickness]

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                              "axes.labelsize":12})
ax = sns.lmplot(x=ILM, y=mpf_mN, hue=R, markers=["o", "x"], data=df,
                legend_out=False, fit_reg=True, height=5, aspect=1.6,
                palette="Set1", truncate=True, ci=95, line_kws={'lw':0})
ax.set(xlabel=ILM, ylabel=MPF)

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = df.dropna(subset=[ILM, mpf_mN])

# linear regressions for fitting
x = df_no_Nan[ILM][df_no_Nan[R] == Eq]
# Convert to N
y = df_no_Nan[mpf_mN][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
plt.text(500, yfit(500) + 4, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# linear regressions for fitting
x = df_no_Nan[ILM][df_no_Nan[R] == Po]
y = df_no_Nan[mpf_mN][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
plt.text(1500, yfit(1500) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# Axis limits
ax.set(ylim=(0, 18))
ax.set(xlim=(0, max(x)*1.1))

# New path
NP = os.path.join(SF, 'ILM_vs_MaxPeel')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

ax.savefig(os.path.join(NP, 'Regression.pdf'), bbox_inches='tight')
plt.close()

# In[Max peel force vs ILM thickness by age group]

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                              "axes.labelsize":12})
ax = sns.lmplot(x=ILM, y=mpf_mN, hue=A60, markers=["o", "x"], data=df,
                legend_out=False, fit_reg=True, height=5, aspect=1.6,
                palette="Set1", truncate=True, ci=95, line_kws={'lw':0})
ax.set(xlabel=ILM, ylabel=MPF)

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
```

```python
df_no_Nan = df.dropna(subset=[ILM, mpf_mN])

# linear regressions for fitting
x = df_no_Nan[ILM][df_no_Nan[A60] == Aleq60]
y = df_no_Nan[mpf_mN][df_no_Nan[A60] == Aleq60] # MmN

x_plot = np.linspace(min(x), max(x), 100)

# linear regression
slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)

# Linear regression line
plt.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1)
plt.text(1250, yfit(1250) + 0.75, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# linear regressions for fitting
x = df_no_Nan[ILM][df_no_Nan[A60] == Ag60]
y = df_no_Nan[mpf_mN][df_no_Nan[A60] == Ag60] # MmN

x_plot = np.linspace(min(x), max(x), 100)
# linear regression
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)

plt.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1) # linear regression
plt.text(1000, yfit(1000) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# Legend
plt.legend(loc='best').set_title(A_G) # legend

# axis limits
ax.set(ylim=(0, 18))
ax.set(xlim=(0, 2200))

# New path
NP = os.path.join(SF, 'ILM_vs_MaxPeel_Age60')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

ax.savefig(os.path.join(NP, 'Regression.pdf'), bbox_inches='tight')
plt.close()


# In[Max peel force vs ILM thickness in the Equator]

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(x=ILM, y=mpf_mN, hue=A60, markers=["o", "x"],
                data=df[df[R] == Eq], legend_out=False, fit_reg=True, height=5,
                aspect=1.6, palette="Set1", truncate=False, ci=95,
                line_kws={'lw':0})
ax.set(xlabel=ILM, ylabel=MPF)

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = df.dropna(subset=[ILM, mpf_mN])

# linear regressions for fitting
x = df_no_Nan[ILM][(df_no_Nan[A60] == Aleq60) & (df[R] == Eq)]
y = df_no_Nan[mpf_mN][(df_no_Nan[A60] == Aleq60) & (df[R] == Eq)] # MmN

x_plot = np.linspace(min(x), max(x), 100)

# linear regression
slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)

# Linear regression line
plt.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1)
plt.text(500, yfit(500) + 1, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# linear regressions for fitting
x = df_no_Nan[ILM][(df_no_Nan[A60] == Ag60) & (df[R] == Eq)]
y = df_no_Nan[mpf_mN][(df_no_Nan[A60] == Ag60) & (df[R] == Eq)] # MmN

x_plot = np.linspace(min(x), max(x), 100)
# linear regression
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)

plt.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1) # linear regression
plt.text(500, yfit(500) - 1, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# Legend
plt.legend(loc='best').set_title("Equator Age group (yr.)") # legend

# axis limits
ax.set(ylim=(0, 20))
# ax.set(xlim=(0, None))

# New path
NP = os.path.join(SF, 'ILM_vs_MaxPeel_Age60_Equator')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

ax.savefig(os.path.join(NP, 'Regression.pdf'), bbox_inches='tight')
plt.close()

# In[Steady state peel force vs ILM density]

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(x=TMD, y=SSmN, hue=R, markers=["o", "x"], data=df,
                legend_out=False, fit_reg=True, height=5, aspect=1.6,
                palette="Set1", truncate=True, ci=95, line_kws={'lw':0})
ax.set(xlabel=DensityUnit, ylabel='Steady state peel force (mN)')

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
```

```python
df_no_Nan = df.dropna(subset=[TMD, SSmN])
# figure out why zero's aren't being eliminiated

# linear regressions for fitting
x = df_no_Nan[TMD][df_no_Nan[R] == Eq]
y = df_no_Nan[SSmN][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
plt.text(85, yfit(85) + 0.2, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between Steady-state and Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan[TMD][df_no_Nan[R] == Po]
y = df_no_Nan[SSmN][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
plt.text(70, yfit(70) + 0.3, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between Steady-state and Posterior\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# axis limits
ax.set(ylim=(0, None))
ax.set(xlim=(0, max(df_no_Nan[TMD])*1.05))

# New path
NP = os.path.join(SF, 'ILM_vs_SteadyStatePeel_Region')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

ax.savefig(os.path.join(NP, 'Regression.pdf'), bbox_inches='tight')
plt.close()

# In[Maximum peel force vs ILM density]

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(x=TMD, y=mpf_mN, hue=R, markers=["o", "x"], data=df,
                legend_out=False, fit_reg=True, height=5, aspect=1.6,
                palette="Set1", truncate=False, ci=95, line_kws={'lw':0})
ax.set(xlabel=DensityUnit, ylabel='Maximum peel force (mN)')

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = df.dropna(subset=[TMD, mpf_mN])
# figure out why zero's aren't being eliminiated

# linear regressions for fitting
x = df_no_Nan[TMD][df_no_Nan[R] == Eq]
y = df_no_Nan[mpf_mN][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
plt.text(85, yfit(85) + 0.1, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# linear regressions for fitting
x = df_no_Nan[TMD][df_no_Nan[R] == Po]
y = df_no_Nan[mpf_mN][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
plt.text(70, yfit(70) + 0.1, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# axis limits
ax.set(ylim=(0, None))
# ax.set(xlim=(0, None))

# New path
NP = os.path.join(SF, 'ILM_vs_MaxPeel_Region')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

ax.savefig(os.path.join(NP, 'Regression.pdf'), bbox_inches='tight')
plt.close()

# In[Collagen fibril density vs age correlation (regression)]

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
# dict(Equator="r", Posterior="b") , 'color':'black', 'color':'blue'
ax = sns.lmplot(x=A, y=TMD, hue=R, markers=["o", "x"], data=df,
                legend_out=False, fit_reg=True,height=5, aspect=1.6,
                palette="Set1", truncate=False, ci=95, line_kws={'lw':0})
ax.set(ylabel=DensityUnit, xlabel=A_yrs)

# Remove all NaN's from the data for regressions

# remove nans from ILM thickness
df_no_Nan = df.dropna(subset=[TMD])

# linear regressions for fitting
x = df_no_Nan[A][df_no_Nan[R] == Eq]
y = df_no_Nan[TMD][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

```

```
slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
plt.text(80, yfit(80) + 5, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# linear regressions for fitting
x = df_no_Nan[A][df_no_Nan[R] == Po]
y = df_no_Nan[TMD][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
plt.text(75, yfit(75) + 5, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

# Axis limits
ax.set(ylim=(0, None))
ax.set(xlim=(None, None))

# New path
NP = os.path.join(SF, 'Density_vs_Age')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

ax.savefig(os.path.join(NP, 'Regression.pdf'), bbox_inches='tight')
plt.close()


# In[Collagen fibril Orientation vs age correlation (regression)]

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
# dict(Equator="r", Posterior="b") , 'color':'black', 'color':'blue'
ax = sns.lmplot(x=A, y=TAA, hue=R, markers=["o", "x"], data=df,
                legend_out=False, fit_reg=True,height=5, aspect=1.6,
                palette="Set1", truncate=False, ci=95, line_kws={'lw':0})
ax.set(ylabel=OrientationUnit, xlabel=A_yrs)

# Remove all NaN's from the data for regressions

# remove nans from ILM thickness
df_no_Nan = df.dropna(subset=[TAA])

# linear regressions for fitting
x = df_no_Nan[A][df_no_Nan[R] == Eq]
y = df_no_Nan[TAA][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
plt.text(80, yfit(80) + 2, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Collagen fibril Equator orientation\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan[A][df_no_Nan[R] == Po]
y = df_no_Nan[TAA][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
plt.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
plt.text(75, yfit(75) + 2, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Collagen fibril Posterior orientation\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# Axis limits
ax.set(ylim=(0, None))
ax.set(xlim=(None, None))

# New path
NP = os.path.join(SF, 'Angle_vs_Age')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

ax.savefig(os.path.join(NP, 'Regression.pdf'), bbox_inches='tight')
plt.close()

# In[Collagen fibril orientation distributions]

# remove nans from ILM thickness
df_no_Nan = df.dropna(subset=[TAA])

# Normal distribution plots
f, ax = plt.subplots(figsize=(9.6, 6))
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})

ax = sns.kdeplot(data=df_no_Nan, x=TAA, hue=R, hue_order=[Eq, Po], fill=True,
                 legend=False, palette='Paired', multiple='layer',
                 cut=0, bw_adjust=0.7, alpha=0.3)

ax.set(xlabel=OrientationUnit, ylabel='Kernel Density Estimation')

# Legend
plt.legend(labels=[Eq, Po], loc='best').set_title(R)

# Axis limits
# ax.set(ylim=(0, None))
# ax.set(xlim=(0, 90))

# New path
NP = os.path.join(SF, 'Angle')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'Distribution.pdf'), bbox_inches='tight')
```