Validation of a Device for High-Throughput Phenotyping of Wheat Stem Flexural Rigidity

Presented in Partial Fulfillment of the Requirements for the Degree of

Masers of Science

With a Major in

Mechanical Engineering

in the

College of Graduate Studies

University of Idaho

by

Clayton Bennett

Major Professor

Daniel J. Robertson, Ph.D.

Committee

Dan Cordon, Ph.D.

Robert Stephens, Ph.D.

Christopher J. Stubbs, Ph.D.

Department Administrator

Gabriel Potirniche, Ph.D.

December 2022

Abstract

Wheat breeders develop new crop varieties each year with a goal to improve yields and to improve farmer success. Farmer success is threatened by stalk lodging, failure of crop stems due to high winds. Selective breeding can be used to mitigate stalk lodging by introducing varieties with enhanced bending strength characteristics. To aid breeders in identifying structurally robust stalks, a high-throughput device known as SOCEM (Strength of Crops Extrapolation Machine) can be used to identify which genetic varieties of wheat are stalk lodging resistant.

Data gathered with the field-deploying SOCEM device was compared to three-point bending test results of wheat stems conducted on a universal testing machine. SOCEM results were also compared to historical reports of actual lodging percentages. In both cases the SOCEM produced accurate assessments of the structural robustness of wheat varieties. A key advantage of the SOCEM is that the data collection is faster and cheaper compared to conducting three-point bending tests or assessing lodging percentages. Data gathered with the SOCEM device could potentially supplant lodging percentage values published in variety trials and yield reports in the future and become the standard by which lodging is assessed. Data from the SOCEM provides increased numerical granularity compared to lodging percentage values and is not directly confounded by uncontrolled weather events.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under Grant OIA- 1826715.

Dedication

To the good people of Moscow, Idaho.

To my Moscow mentors, Dr. Daniel Robertson, Dr. Chris Stubbs, Suvia Judd, and T-Jay Clevenger.

To my friends.

To my family.

Table of Contents

List of Tables

List of Figures

Chapter 1: Background

Global Wheat Production

The stability of modern civilization relies on the stability of our crop supply. Wheat (*Triticum aestivum* L.) is arguably one of the world's most important crops, providing about 20% of dietary calories and proteins worldwide [1]. In 2020, wheat was the world's 64th most traded product [2]. An estimated 781 million tons of wheat were produced in 2021, with a total consumption of 778 million tons of wheat [3].

Argentina, Australia, China, India, Russia, and the United States are the world's largest producers of wheat [4]. Figure 1 shows the concentrations of wheat grown in each of these nations.

Global wheat production currently meets global wheat consumption (see Figure 2), while demand for wheat continues to grow each year [3] and the cost of food continues to rise.

It's important to recognize that the amount of wheat planted each year must exceed projected demand, to overcome losses due to crop failure. In the United States, in the last decade, roughly

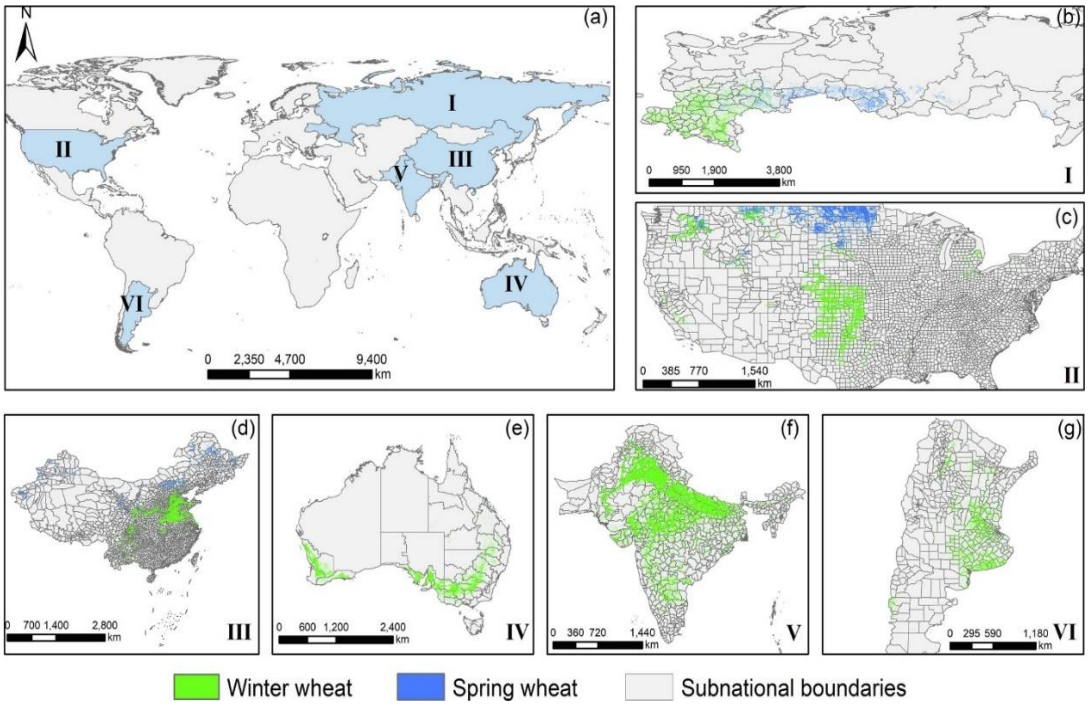

Figure 1: Spatial distribution of wheat crop agriculture in nations with leading contributions to global wheat production. Figure 1a provides an overview of which nations are included. Figure 1b shows the distribution of wheat in the Russian Federatio n and the Ukraine. Figure 1c shows the distribution of wheat grown in the United States of America. Figure 1d shows the distribution of wheat grown in China. Figure 1e shows the distribution of wheat grown in Australia. Figure 1f shows the distribution of wheat grown in India. Figure 1g shows the distribution of wheat grown in Argentina. Luo, et al, 2022 (Fig. 1, [4]).

20% of wheat planted has not been harvested (see Figure 3 [5]). Harvest is not performed when it is no longer economically advantageous for the grower (i.e., harvest cost outweighs selling price). Stalk lodging, discussed in the next section, can significantly increase the cost of harvest and reduce grain quality.

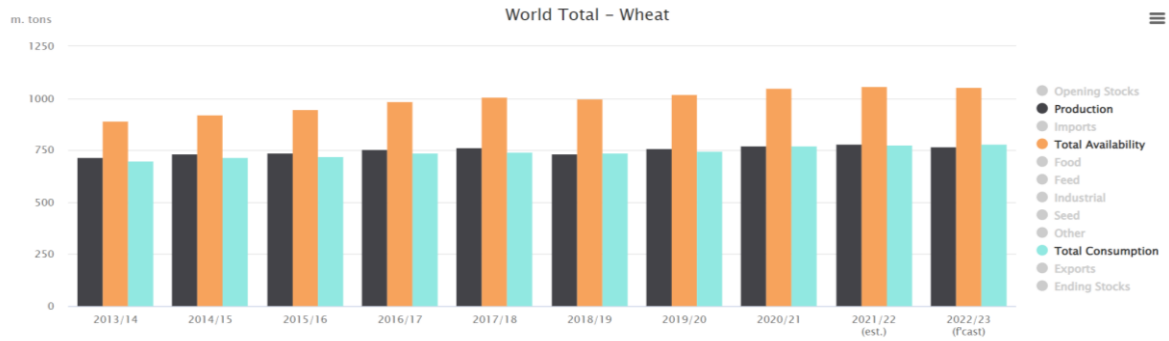

Figure 2: Wheat production, consumption, and storage amounts. International Grains Council, 2022 [3].

When deciding which varieties of wheat to plant each year, farmers consider many factors with the goal of mitigating the risk of crop failure, while maximizing potential yield of high quality grain [6]. Figure 57 in the Appendix displays a sales brochure for a particular genetic variety of wheat seed [7], and the details offered provide insight into the many considerations that farmers make when making planting decisions.

Wheat varieties are generally produced and marketed to be grown in the local conditions of specific geographic regions. Wheat that is successful in one location may not be appropriate in another location. Plant breeders attempt to produce new wheat varieties each year that are optimized to be successful in local conditions, and these are marketed to farmers based on region. For example, William Farrer, an Australian agriculturist, first developed disease-resistant and drought-resistant wheat cultivars through

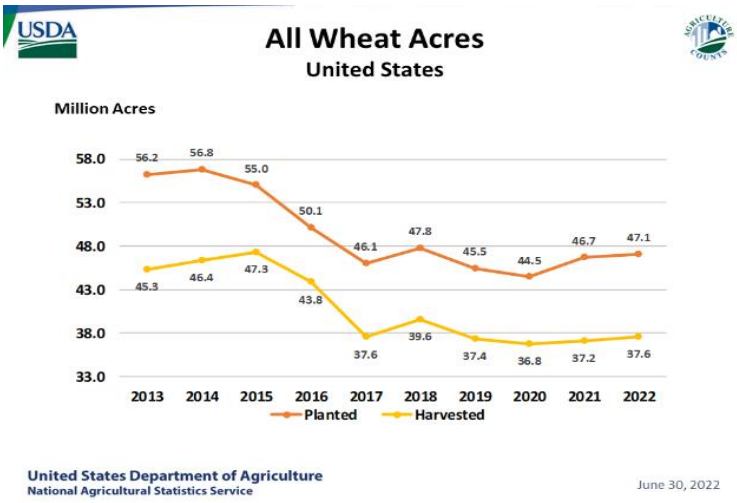

Figure 3: Every year, in the United States, about 10 million acres of wheat are planted but not harvested. Stalk lodging contributes to these losses. USDA, 2022 [5].

selective breeding in the early 1900's to make it possible to grow the crop successfully in Australia's challenging climate [8]. Selective breeding continues to be a useful solution today for minimizing the risk of crop failure and maximizing yields [9].

Stalk Lodging

Stalk lodging, which is defined as the structural collapse of stalks, typically due to wind [10], continues to challenge plant breeders and leads to significant harvest losses each year. Root lodging, a similar phenomenon which also results in the collapse of plants, occurs when the root system pulls out of the ground. Comparatively, root lodging is more common in areas that receive heavy rainfall, and stalk lodging is more prevalent in arid regions where root wads are held firmly in dry, stable earth [10]. Figure 4 and Figure 5 show a lodging event south of Moscow, Idaho that occurred in August of 2022. Both stalk lodging and root lodging were present in this case.

Figure 4: A lodging event in Idaho.

Figure 5: Lodging of wheat stems in Idaho.

Lodging complicates automated harvesting with a combine harvester [11] and reduces photosynthetic area (i.e. leaf surface area), thereby reducing plant health, which in turn makes fields more susceptible to pest and disease.

Factors Affecting Lodging Resistance

Many factors impact stalk lodging propensity and resistance. Scientific investigations been made into the relationship between lodging and management factors such as seeding density [12] [13], soil composition [14], and the application of nitrogen and plant growth regulators [15]. Anatomical factors have been widely studied, including lignin content in cell walls [16], arrangement of vascular bundles [16] [17], microfibril angle in cell walls [18], and cellular turgor pressure [19]. Externally-observable morphological factors, such as plant height and stem diameter, have also been shown to be predictors of stalk lodging susceptibility [10]. For maize (*zea mays* L.), stem diameter has been shown to be a more important lodging predictor than chemical composition factors [20]. The same has been shown for wheat [21]. Perhaps the most notable achievement which has reduced stalk

lodging was the development of 'dwarf' wheat varieties in the 1970's which exhibit reduced plant height [10].

Regardless of the many factors that impact a plant's health and structural robustness, a wheat stalk ultimately lodges when the forces applied to it exceeds the plant's structural bending strength. Consequently, there have been numerous attempts to accurately quantify the structural bending strength of wheat stems. One of the biggest challenges to characterizing structural bending strength of wheat stems is the time and effort required to conduct flexural testing. The equipment and labor cost involved in testing individual stems often prevents plant breeding programs from utilizing structural bending strength as a target of selected breeding efforts.

Existing Solutions and Devices to Quantify Lodging Resistance

The lodging propensity of genetic varieties of crops can be compared statistically, with data that has been sourced from lab devices, field devices, and historical lodging rates.

Laboratory-based Devices for Assessing Lodging Resistance

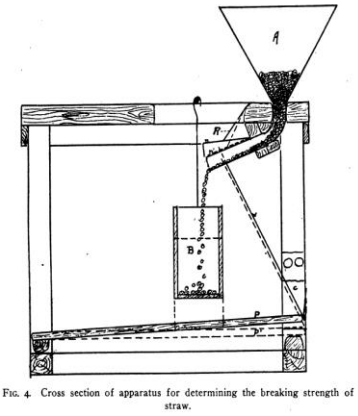

Figure 6: Three-point bending, Helmick, 1915 (Fig. 4, [22]).

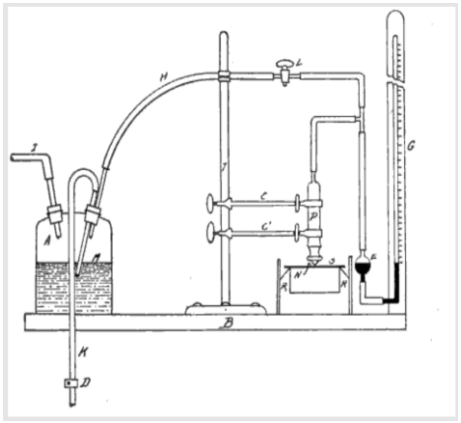

Figure 7: Three-point bending, Willis, 1925 (Fig. 1, [23]).

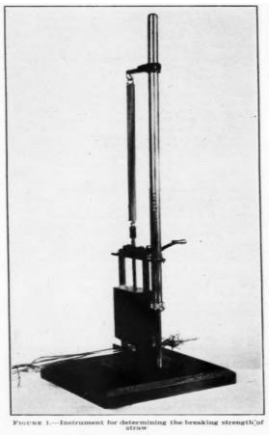

Figure 8: Three-point bending, Salmon, 1931 (Fig. 1, [11]).

The first devices to measure the breaking resistance of wheat stems were developed in the first half of the 1900's [11] [22] [23]. These early devices, shown in Figure 6, Figure 7, and Figure 8, while varied in their machinations, all performed three-point bending tests in a laboratory setting, with a stem oriented horizontally, supported at two ends, and pressed downward in the center. Beginning in 1926, S.C. Salmon at Kansas State University compared three-point bending results with observed lodging rates of various genetic varieties of wheat [11] and found weak correlation.

In general, laboratory-based testing of wheat stems has the benefit of consistency of treatment and the clear isolation of desired traits (e.g., bending strength). The drawback of laboratory testing is

that specimens must be collected and then transported to the lab, and, in the case of stems, throughput is very low and labor demand is high, because each specimen must be tested one at a time. Modern universal testing systems provide more precise control and measurements than the systems used in the early 1900's to characterize stem bending strength, however, these systems still require significant human labor to conduct tests on individual specimens.

Field Devices

Since the 1970's, devices have been developed which can be deployed in a field setting to test the mechanical properties of stalks. Most of these devices were primarily developed for testing of large grain crops such as maize (*Zea mays* L.) and sorghum (*Sorghum bicolor* L.). However, field-based devices have also been developed for cereal crops like wheat (*Triticum aestivum* L.), barley (*Hordeum vulgare* L.), and rice (*Oryza sativa* L.). A primary benefit of field testing is that it does not require harvesting and transporting of samples to a lab.

Stalk pushing devices are a common type of field device that has been developed to estimate stalk lodging resistance in the field. Stalk pushing devices treat each stalk as a cantilever beam that protrudes from the earth, as the earth ideally holds the stalk firmly upright.

Most existing stalk bending devices record applied force and displacement data while the plant is deflected from its upright position. Various algorithms have been developed to use these inputs to calculate values which correlate with stalk lodging resistance. Important values that have been used to draw conclusions include: The energy required to displace the stalk to a certain angle [24], the energy of the plant returning to an upright position [25], the slope of the linear region of the load-deflection curve [26], and the ultimate breaking force at which a stalk artificially lodges due to manual application of force [26][27].

This writing will cover four different devices, shown in Figure 9, that operate on the principle of stalk pushing.

The DARLING device has been developed by team members from New York University, University of Idaho, and Brigham Young University [26], and ongoing development continues into 2022. The load cell of the DARLING contacts a single maize or sorghum plant. The user pushes the device to deflect the stalk, and the load force and deflection angle are measured and stored as data. The DARLING requires that users be trained in good technique, and testing throughput is low because one plant is tested at a time [28].

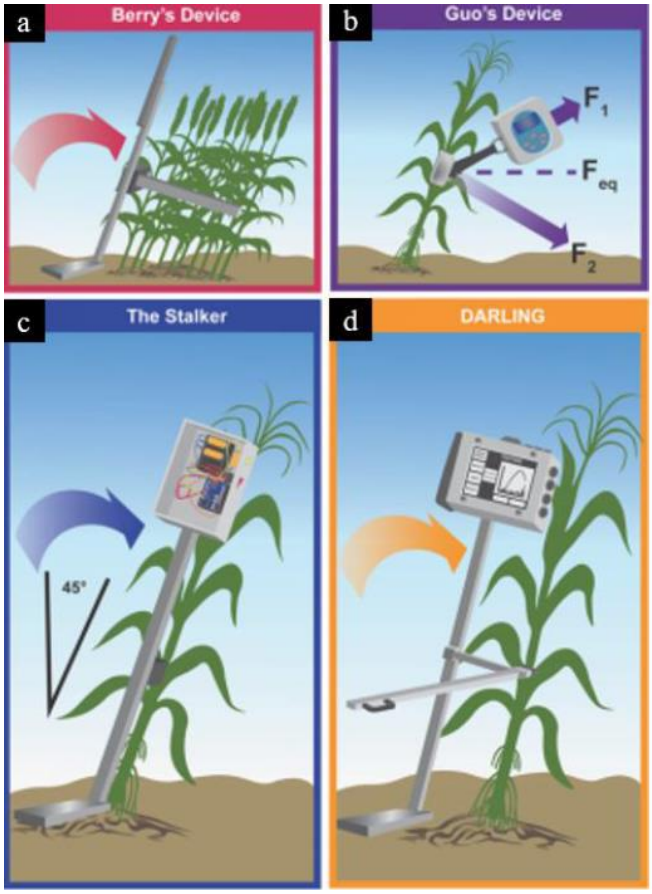

Figure 9: Various devices used for measuring the lodging resistance of crops. Erndwein, 2019 (Fig. 3, [26]).

Berry and Sterling first developed their "Lodging Instrument" in 2000 in the United Kingdom [24]. Berry's device, much like the DARLING, uses a load cell mounted on a shaft that pivots relative to one spot on the ground. Berry's device differs because it is meant to measure many cereal stems at once, rather than a single stalk of a large grain crop. While this is higher throughput than a single plant, the push-and-return motion does not allow for a continuous testing of an entire plot of cereal crops. Berry's device, like all push-and-return devices, requires good technique by the user.

The Stalker was developed at the University of Minnesota, starting in 2018 [25]. The Stalker continues to be developed, with results published in 2022 [29]. Operation of the Stalker requires exactly 45 degrees of deflection from the upright position, and then the difference between the energy of displacement and the energy return to upright is calculated. The Stalker has been used to deflect individual maize stalks, like the DARLING device, and has also been outfitted with a force bar attachment to contact several cereal stems at once (similar to Berry's device). The Stalker team has pursued an open-source model, and a parts list and instruction for manufacturing a Stalker device

are available for free online [30]. One challenge with the stalker is that plants often break prior to being deflected by 45 degrees.

Guo's device, presented in China in 2018, uses a load cell in tension, pulled at a specific angle by the user [27]. This device has the smallest form factor. Proper technique is very important for accurate results, and the throughput is low. Guo's device has a well-made interface with colorful buttons, though current development status is unknown.

The drawback of these existing devices is that they are not high throughput. Also, the underlying mechanical principles, particularly canopy interactions, are different when testing an individual maize stalk as opposed to wheat stems that are in close contact with one another.

Using Historical Stalk Lodging Rates to Assess Lodging Resistance

Agriculturists have long kept records of failed crops. Figure 58 (in Appendix A) shows an early example of recorded historical lodging rates [11]. Figure 10 shows the results of one of the first comparisons between breaking strength and lodging rates. Crop failure records ostensibly enables the identification of weak and strong genetic varieties. However, numerous environmental factors affect stalk lodging, and these factors cannot be controlled across environments, locations, or years. This prevents a straightforward analysis of historical lodging data. In fact, the lodging percentage of a given genetic variety grown in a specific location will often not be statistically correlated with the lodging percentage of that same variety grown in the same location the following year.

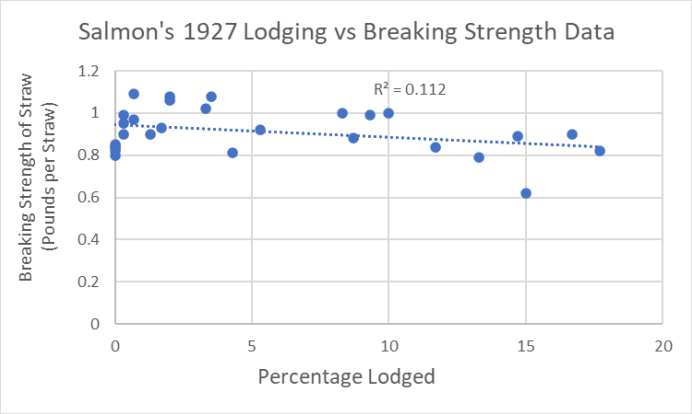

Figure 10: Data from the first recorded comparison between lodging percentage and three-point bending stem performance. The slope of a linear best fit is negative, as expected, though the correlation is numerically weak, $R^2 = 0.112$.

Mechanics of Materials of Cereal Stems

Cantilever Beams

A plant rooted in the ground and growing upward toward the sky can be modeled or approximated as a cantilever beam. Wind forces would realistically be distributed along the exposed profile of the plant, including its stem, leaves, and spike. However, an idealized model is often utilized when analyzing plant biomechanics which approximates the wind load as a single point force [31]. Field based phenotyping devices also apply a point force at a distinct location along the length of the stalk. When loaded in this manner the bending moment induced in the plant is proportional to the applied force multiplied by the distance between the stable base and the placement of the point-force [32]. Therefore, when conducting field-based testing the height of the force-measuring load cell from the ground needs to be considered when calculating a stalk's flexural rigidity.

Defining Stem Strength

If a plant is considered a cantilever beam, then the flexural rigidity (EI) of the plant can be solved for in terms of applied force and displacement by rearranging the cantilever beam deflection equation provided by Shigley [32]. Young's modulus, E (Eq. 1), is the slope of the elastic region of the stress-strain curve for a given material. The area moment of inertia, I, is dependent on the geometry and size of an object [33]. The area moment of inertia of a solid cylinder and a hollow cylinder are defined in Eq. 2 and Eq. 3, listed in Table 1 below, based on the dimensions and axes shown in Figure 11. Dimensions are defined further in Table 2. Flexural rigidity, EI (Eq. 7), also known as bending stiffness, has been previously used as a numeric mechanical property of plant samples [34] as well as bone [35], with the benefit of usefulness despite complex geometries. It is important to note the length between the base of a plant and the applied point-force has a third-power impact on the measured flexural rigidity of a cantilever beam (e.g., plant). Flexural rigidity is a useful quantity to determine during field testing because it can be determined experimentally without needing to determine independently the values of diameters, wall thickness, area moment of inertia, or Young's modulus.

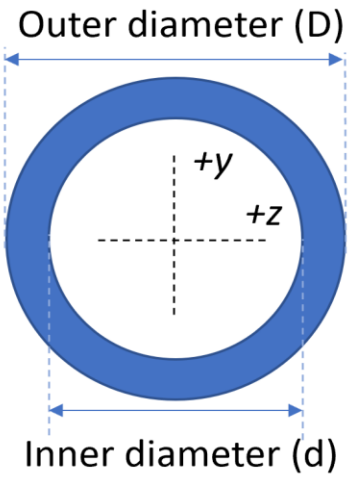

Figure 11: Cross section of a hollow cylinder. Bending is enacted about the z axis.

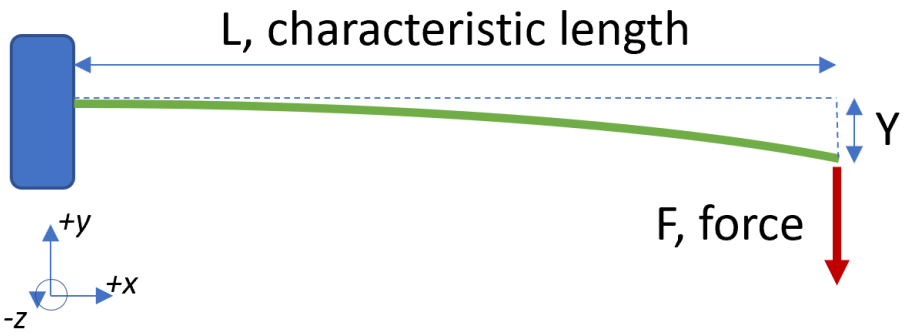

Figure 12: Cantilever beam with deflection Y given a point force F applied at length L.

Table 1: Equations for mechanics of materials of round cantilever beams in bending.

$E = \frac{\sigma}{\varepsilon}$	Eq. 1: Young's modulus [33].
$I_{z,solid\ cylinder} = \frac{\pi}{64}(D^4)$	Eq. 2: Area moment of inertia, for a solid round beam deflecting about the Z axis, according to Figure 11 [33].
$I_{z,hollow\ cylinder} = \frac{\pi}{64}(D^4 - d^4)$	Eq. 3: Area moment of inertia, for a hollow round beam deflecting about the Z axis, according to Figure 11 [33].
$Y_{max} = \frac{FL^3}{48EI}$	Eq. 4: Deflection, Y, that causes maximums stress in a cantilever beam [33]. See Figure 12.
$EI = \frac{F}{Y} \cdot \frac{L^3}{12}$	Eq. 5: Flexural stiffness, cantilever beam. Reorganized from Eq. 4. See Figure 12 and Figure 13.

Table 2: Definitions of variables

Σ	Stress
ε	Strain
F	Force applied
Y	Deflection along the primary axis
Y_{max}	Deflection that incurs maximum stress
F/Y	Slope of linear region of the load deflection curve
L	Characteristic length of bending
I	Area moment of inertia, based on geometry
E	Young's modulus, i.e., stiffness
EI	Flexural rigidity

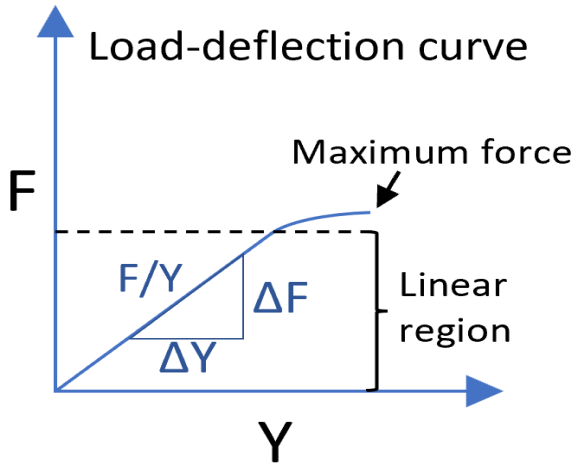

Figure 13: Load-deflection curve, applicable to both three-point bending and cantilever bending.

Stacking Beam Model

Stems in a canopy support one another. As each stem deflects, it makes contact with its neighbors, transferring load to other stems. Therefore, the instantaneous force required to deflect a wheat stalk is not necessarily a true representation of the deflection resistance of that plant because it is being supported by neighboring plants. An algorithm was developed by Bebee to calculate the stiffness of a beam given full-contact support from its neighbors [36]. This model will be referred to as the Stacking Beam model herein.

Figure 14 shows an example of stacking beams which support one another in one dimension of travel. In juxtaposition to the ideal beam geometry in Figure 14, Figure 15 shows an example of the complexity of wheat canopy. In a plant canopy, three dimensions of interaction are at play, with additional elements of complex geometries of leaves and friction between contacting surfaces. However, the reality of individual plant flexural rigidity within a canopy is bounded, with the reality being more than no support (i.e., no contact) and less than perfect support from neighbors as depicted in Figure 14.

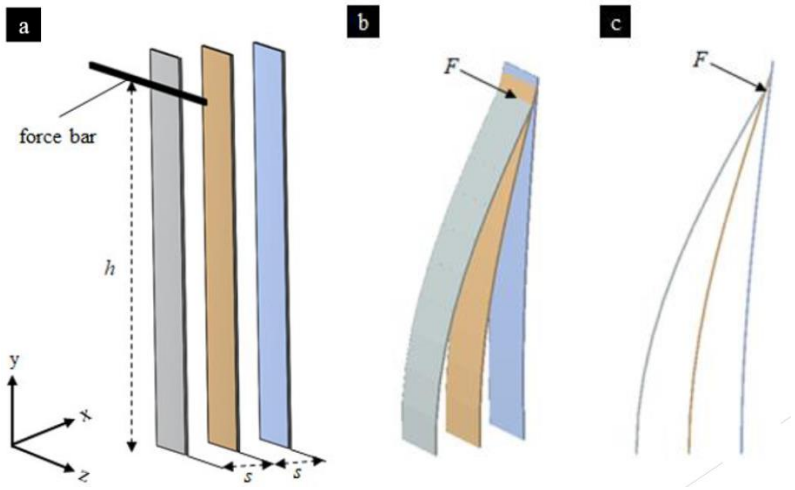

Figure 14: Stacking Beam Model. Bebee, 2020 (Fig. 2.1, [36]).

Figure 15: Small plots of wheat, for experimentation. University of Idaho, 2022 [37].

Chapter 2: Description and Validation of the SOCEM Device

Wheat breeding decisions can be informed by field testing small plots of wheat with the newly developed Strength of Crops Extrapolation Machine (SOCEM). In particular, flexural rigidity results from "SOCEM trials" can be used to determine the stalk lodging resistance of different genetic varieties. Devices have previously been developed for this purpose [26] [30], but they are low-throughput and do not consider canopy interactions between plants. In this writing, we present and validate the SOCEM device, which is designed to perform high-throughput testing while also considering interactions between plants.

What is the SOCEM?

The SOCEM, shown in Figure 16, is a field deploying measurement device which was originally developed by the AgMEQ laboratory at the University of Idaho under the supervision of Dr. Daniel Robertson. The SOCEM can be used to experimentally determine the flexural rigidity of cereal stems in a canopy in a high throughput manner. To achieve this, the SOCEM device is pushed through a plot of cereal crops. While in motion, a load cell and a rotary encoder are used to record data for force and distance traveled. A rigid bar is connected to the load cell such that many stems are

Figure 16: The SOCEM, as of 2020. Bebee, 2020 (Fig.3.1, [36]).

simultaneously deflected when the device moves through the canopy. All data is processed and stored to memory by a purpose-built program called StemBerry, written in Python 3 (Python Software Foundation, Wilmington, Delaware, USA). Analysis of the data using the PeakClick Python module [36] enables calculation of the average flexural rigidity of the stems deflected during the test.

Hardware

The SOCEM includes several pieces of hardware, described below and shown in Figure 17. A load cell is mounted on a chassis that can be adjusted vertically using a pair of quick release sliders. This enables the load cell height to be rapidly changed, based on the available height of wheat stems. A round carbon fiber tube, referred to hereafter as the "force bar," is attached to the front of the load cell. The bar contacts plant stems as the SOCEM is pushed through a small plot, and the resistance force of the stems is transmitted to the load cell. In order to collect distance traveled information, a cog is fixed to the rear wheel such that it rotates with the same angular velocity as the wheel itself. A belt connects the cog to a rotary encoder which enables calculation of distance traveled.

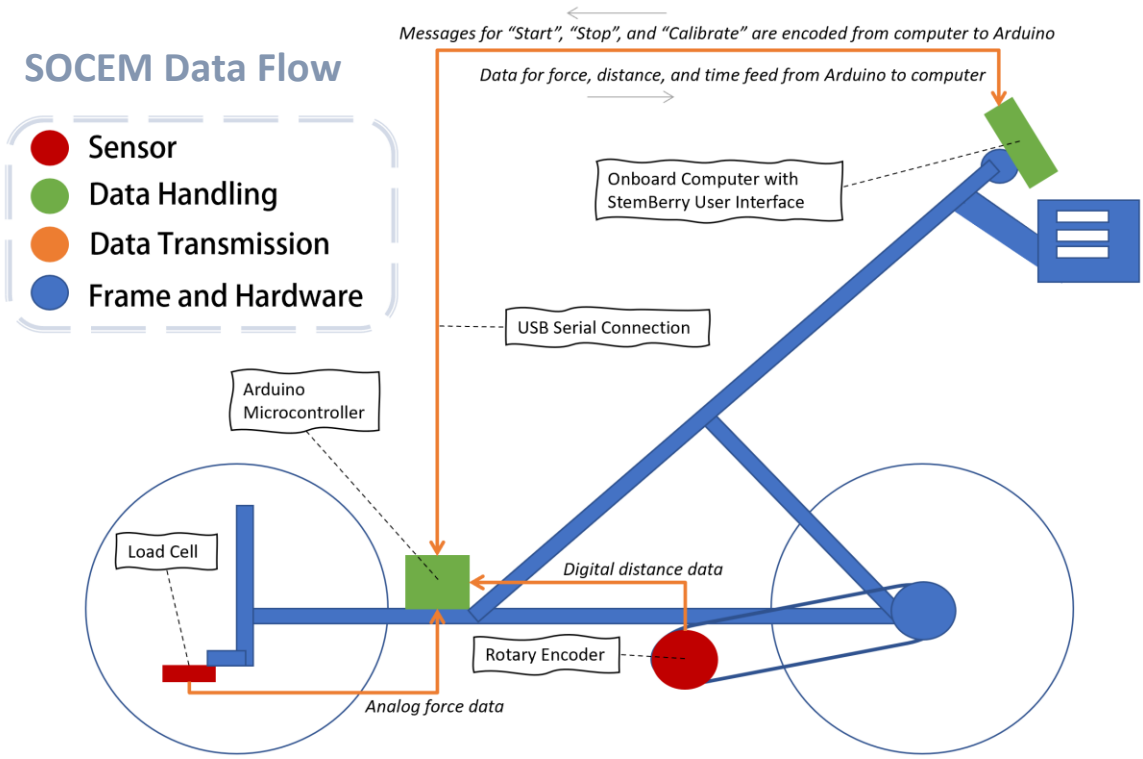

Figure 17: A diagram showing the flow of data from the sensors to the Arduino and then finally to the onboard computer which runs the StemBerry interface.

Software

The SOCEM device includes an onboard computer which runs the StemBerry program. The StemBerry program is a graphical user interface (GUI) coupled with a backend component for analysis and databasing. Through the GUI, a user can record raw force data and input important metadata (e.g., plot number, stem height, force bar height setting, stem count, etc.). Behind the scenes, the StemBerry backend processes the data collected and organizes it into a file system. A Raspberry Pi onboard computer (Raspberry Pi 3 Model B+, Cambridge, England) and an Arduino microcontroller (Arduino Uno Rev3, Arduino.cc, Turin, Italy) communicate via serial connection. Figure 17, above, provides an overview for data software communication and sensor input. More detail about data flow is provided below in Figure 18. Button clicks on the Pi-based user interface, shown in Figure 19, command the Arduino when data collection should start and stop.

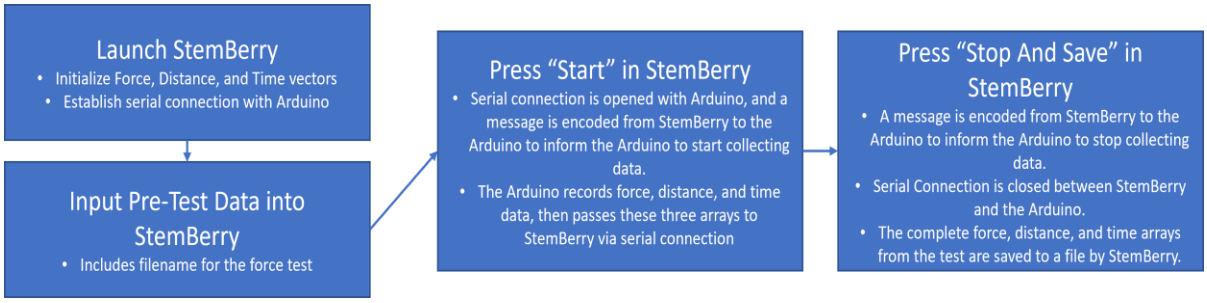

Figure 18: Serial connection and data flow between StemBerry program and Arduino Uno.

During data collection, three data vectors are recorded: Force from the load cell, distance traveled from the rotary encoder, and time since the start of the test. The force, distance, and time data from each test are stored in a XLSX file (Microsoft Excel Open XML Spreadsheet, Standard ECMA-376, ISO/IEC 29500), along with user defined metadata.

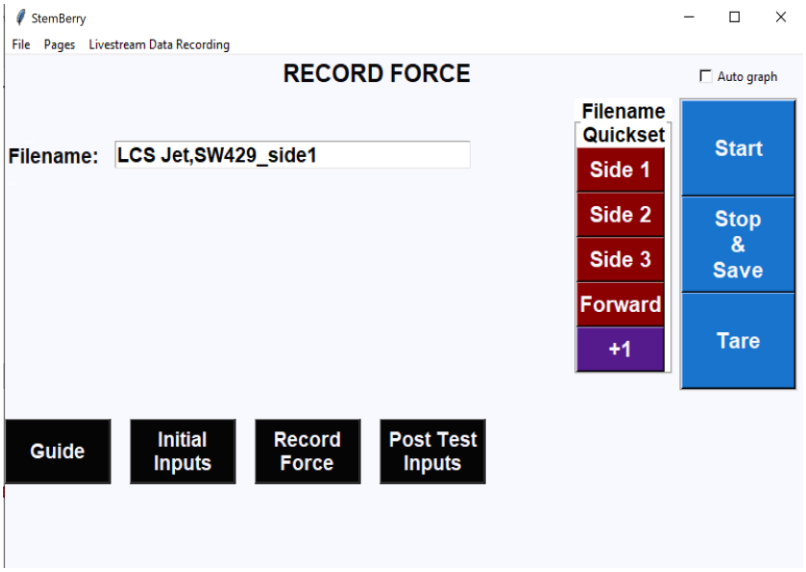

Figure 19: The data collection frame from the StemBerry interface, in a 2022 version. Start and Stop buttons control data collection.

For 2020 and 2021 data, these files were ultimately compiled and analyzed using Python and MATLAB (MathWorks, Inc., in Natick, Massachusetts, USA) scripts on a separate computer to enable calculations of stem flexural rigidity.

Calculating Flexural Stiffness Values from SOCEM Data

A purpose-built Python script, called SOCEM_DataAnalysis.py, was used to calculate values of flexural rigidity given force data, displacement data, and metadata collected with the SOCEM. The program interface, shown in Figure 20 and Figure 21, displays raw force and distance data, and the user is able to choose points of interest. From initial graph of raw data, mouse clicks are used to define the yellow lines shown in Figure 20, between which data is considered useful, and rejecting data impacted by edge effect, which is discussed later in Chapter 2. Once the useful data is identified, characteristic force peaks are selected, represented by the red dots in Figure 21. From the Stacking Beam model, discussed in Chapter 1, we know that force peaks represent points of high interaction between stems. These points of high force are selected, and stem flexural rigidity is automatically calculated for each point, based on force bar height, plant height, and plant to plant spacing.

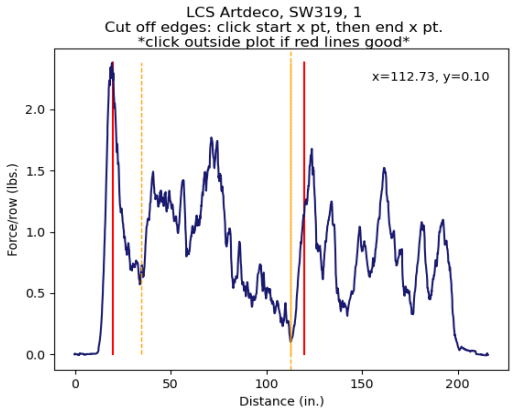

Figure 20: EI assessment tool, Screen 1. First, choose the range of useful data. The red lines show the suggested range, based on edge effect. The yellow lines show the range that the user selected for analysis.

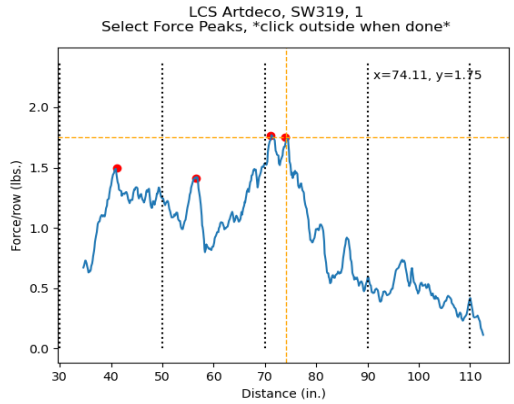

Figure 21: EI assessment tool, Screen 2. Select force peaks that appear substantial. This is subjective, and the user should be consistent. The calculated EI will be the average of the EI that is computed for these discreet points.

The stacking beam model is codified as two Python scripts, EI_Interaction_Fx.py and EI_No_Interaction_Fx.py. These modules take inputs of peak selection data and stem height data. Specifically, the input variables are: force bar height, plant to plant spacing, plant height at the selected point, and the raw force at the selected point. EI_Interaction_Fx returns a low flexural rigidity number by assuming that stems provided full support to one another, and thus are

characteristically weaker than raw force divided by stem count. EI_No_Interaction_Fx returns a high flexural rigidity value by assuming no canopy support, with every stem standing alone. These two outputs are averaged together to supply a flexural rigidity result that assumes a median level of canopy interaction. The average flexural rigidity from selected points is then considered the characteristic lodging resistance for the small plot at hand.

Experimental Data Collection with the SOCEM

Experimental plots of wheat, commonly referred to as “small plots”, can vary in size from a single row of plants to an entire field. One of the more common sizes of experimental plots used for genetic variety testing of wheat is shown below in Figure 22. This photograph was taken during SOCEM trials in 2020, and the photograph represents the size and condition of small plots tested in this study. These plots are approximately 200 inches long and about 40 inches wide. The space between each row, referred to in this writing as the “interrow”, is approximately five inches wide. The SOCEM was developed and optimized to characterize experimental plots of approximately this size as opposed to large multi-acre agricultural fields.

Figure 22: A standard size experimental wheat plot for small grain yield trials, commonly referred to as a small plot.

The SOCEM can test several small plots in a short period of time. In its current form, the SOCEM is designed to be used immediately following harvest so that data is not influenced by plant decay. With modification, the SOCEM could test plants during the growing season, though this has yet to be

accomplished. After a combine harvester cuts a plot, stem stubble remains in the field. The SOCEM is used to characterize the stubble left in the field after harvest. In this way, SOCEM data collection does not interfere with other measurements a plant breeder may wish to collect (e.g., yield).

Edge effect [37] [38] [39] is an important consideration when testing small plots. Plants that are close to the edge of a plot experience less competition for nutrients and sunlight [37]. The mechanical properties of plants along the edge of a plot may therefore differ from the rest of the plot. Importantly plants on the edge of the plot may not be representative of a variety's performance in a large agriculture-scale growing environment [39].

Due to edge effect, SOCEM data should be filtered to only include the central region of the small plot. To accomplish this, the force bar is typically adjusted such that the rows that are on the extreme sides of the plot do not contact the force bar. In addition, data collected from the first and last twenty inches of the plot is typically excluded from analysis, as previously discussed and shown in Figure 21a. Additionally, the back end of a plot is generally full of cut chaff that the harvester will dump there before moving on to harvest the adjacent plot. Even if the plot is manually cleaned with careful preparation, chaff is often woven between standing stems. This chaff can alter the force deflection characteristics of the stems. Therefore, during data analysis, it is common to remove data the last forty inches of the plot.

Validating SOCEM Measurements: Experimental Design Overview

As part of this Thesis, field trials ("SOCEM trials") were performed for two years, at two sites in the Palouse region of Idaho. In 2020, SOCEM data was collected for 15 hybrid varieties, with replicants across 57 small plots. In 2021, SOCEM trials were run for 13 hybrid varieties, with replicants across 48 small plots.

To validate the SOCEM, the characteristic lodging resistances of each plot (i.e., the average flexural rigidity result from the SOCEM) were calculated. Multiple stems from each of plot were also collected prior to SOCEM testing, packaged, and later subjected to three-point bending testing using an Instron (trademark registered to Illinois Tool Works Inc.) universal testing machine. These tests are hereafter referred to as "Instron trials". Linear correlation analysis was then conducted to compare SOCEM flexural stiffness results to flexural stiffness results obtained on the Instron. In addition, SOCEM results were compared to historical lodging rate data that had been collected for each of the hybrids. Specific details of these experiments are given below.

Experimental Methodology

Overview

Thirty undamaged stems were collected from each small plot immediately prior to SOCEM testing. Ten of these stems would later be tested for flexural rigidity using a universal testing machine, specifically an Instron 6800 Series single column system (manufactured by Illinois Tool Works Inc., Norwood, Massachusetts, USA).

SOCEM flexural rigidity calculations require information on plant spacing and stem height. Plant spacing data was obtained prior to SOCEM testing by counting the number of stems in a forty-inch span in two separate rows within the test region. These counts were then averaged together to provide a single count value, which would be assumed to represent homogenous count density. Stem heights were also measured at several locations throughout the plots prior to testing. The height of the force bar on the SOCEM was set based on the stem heights, with intent to use a force bar height appropriate for stem height variation within the small plot. Initial stem count and height values were plugged into the StemBerry Interface on the SOCEM onboard computer immediately preceding each test, and then force, distance, and time data was collected. This process was repeated for each small plot.

SOCEM Experimental Method

1. Collect stems. Gather at least thirty stems from each plot. Bundle and label with the plot identifier (e.g., SW429).
2. Count stems to estimate stem density. In 2020 and 2021, the standard was to count stems for two different rows within a 40-inch length, and then to average the stem counts together.
3. Measure stem heights. Once stem heights are measured, the force bar can be set properly to a level that is between 70% and 90% of the height of the wheat stubble. In 2020 and 2021, stem heights were recorded every 20 inches within the intended test region, generally from 20 inches to 120 inches measured from the start of the small plot horizontally. These stem height values were entered into a StemBerry input screen (Figure 23) that fed data into the optiH.py module, which calculated the ideal force bar height.

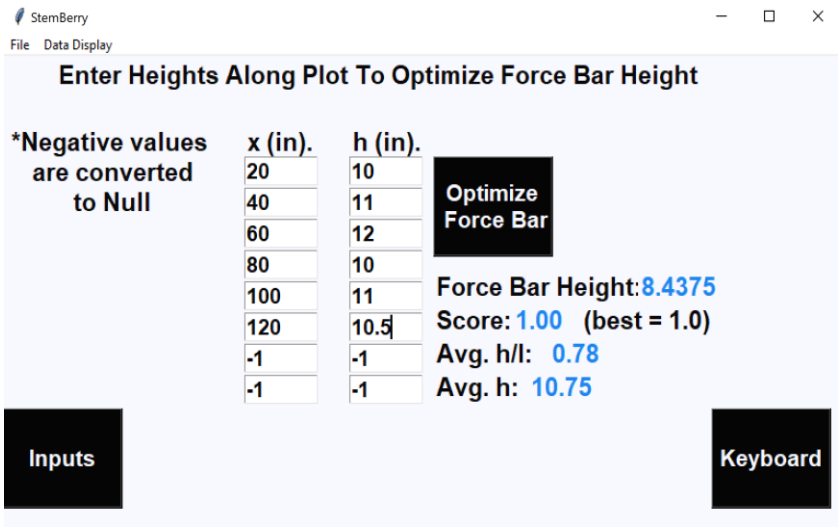

Figure 23: StemBerry height calculator screen, used in 2020 and 2021. Numbers are fed to the optiH.py method.

4. Set the height of the force bar. Set the device at the start of the plot and input necessary data into StemBerry.
5. Start the test. Push the handlebars to roll the device through the plot. Keep an eye on the force bar and the wheels to ensure that the device is travelling straight, so that contact is made consistently with the desired rows. Push through to the end of the desired test region, and then stop the test. Name the output file with a useful name, and then save the output file.

The output files for 2020 and 2021 data collection included raw data for time, force, and distance travelled, taken at a frequency of about 90 Hz.

In 2020 and 2021, raw data was processed after-the-fact using a prototype version of PeakClick, formerly known as SOCEM_DataAnalysis.py (see Figure 20 earlier in this chapter), along with several other Python tools for importing and processing data. An external spreadsheet of additional data, known as plotHeights.xlsx, was continually referenced. Python tools included fixRawDataExcess.py, getFileCreationDate.py, and fileSort_timeBasedVsDistanceBased.py. As of 2022, these tools are all obsolete, due to software improvements covered in Chapter 3.

Several methods, hardware, and software tools were updated for the 2022 field season. The StemBerry software is currently at Version 98 in its development and is referred to as StemBerry_v98.py. Appendix B1 contains the text for StemBerry_v98.py. See Chapter 3 for updates and Chapter 4 for suggestions regarding future development.

Three-Point Bending Experimental Method

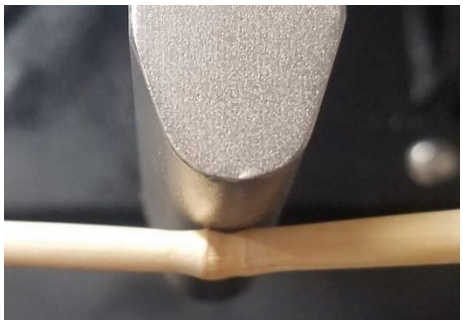

Figure 24: Instron anvil, as used in 2021 trials, in contact with a node of a wheat stem, in the direction of major diameter.

Ten stems from each plot were individually subjected to three-point bending tests, to determine the stiffness and max load of each stem.

Three-point bending tests were performed with an Instron universal testing machine (see Figure 24). The specific methodology for these tests is shown in Table 2. Flexural rigidity values were calculated from three-point bending test results, as shown in Figure 25 and in the equations in Table 3. Flexural rigidity results from all stems from a plot were then averaged.

Table 2: Three-point bending method, numbered steps

1. Select a stem from the bundle of samples. Ensure that is unbroken, does not have any abnormal shape features, and that its size is neither oddly large or oddly small.
2. Place the stem on the supports and then lower the anvil manually, slowly, applying about 0.02N of force to the stem to fix it in place.
3. Start the test. The automatic Instron Bluehill method is predefined by the user.
4. Observe the test, noticing how the stem breaks.
5. Record the break type and record any notes.

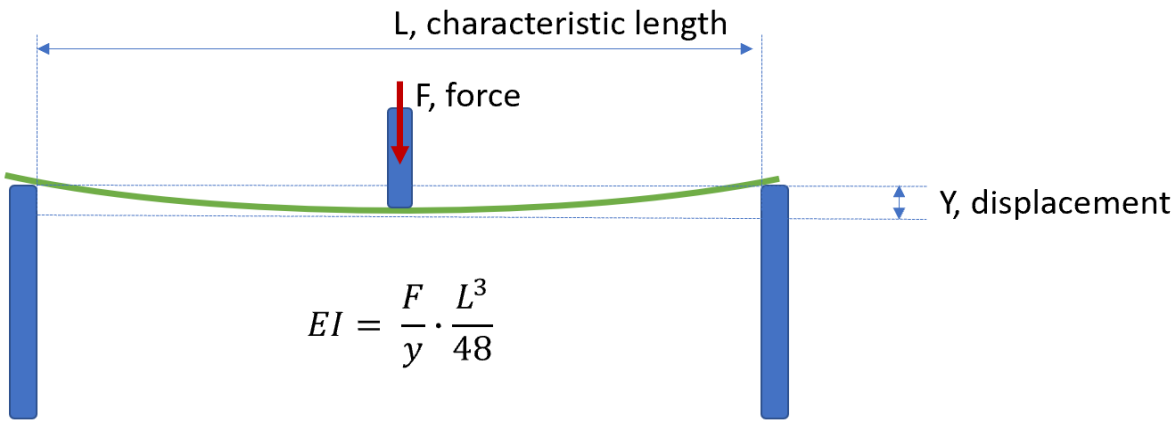

Figure 25: Flexural rigidity, EI, for three-point bending. Span length, L, was 8 cm for 2020 and 2021 trials.

Table 3: Equations for mechanics of materials of round beams in three-point bending.

$Y_{max} = \frac{FL^3}{48EI}$	Eq. 6: Deflection, Y, that causes maximum stress for a beam in three-point bending [33].
$EI = \frac{F}{Y} \cdot \frac{L^3}{48}$	Eq. 7: Flexural rigidity of a beam in three-point bending. See Figure 25.

During three-point bending testing, a load-displacement curve is generated for each stem, showing the force that the stem is subjected to at the loaded node versus the vertical displacement of the anvil as it moves to apply load. Examples of curves generated for wheat stems are shown in Figure 26.

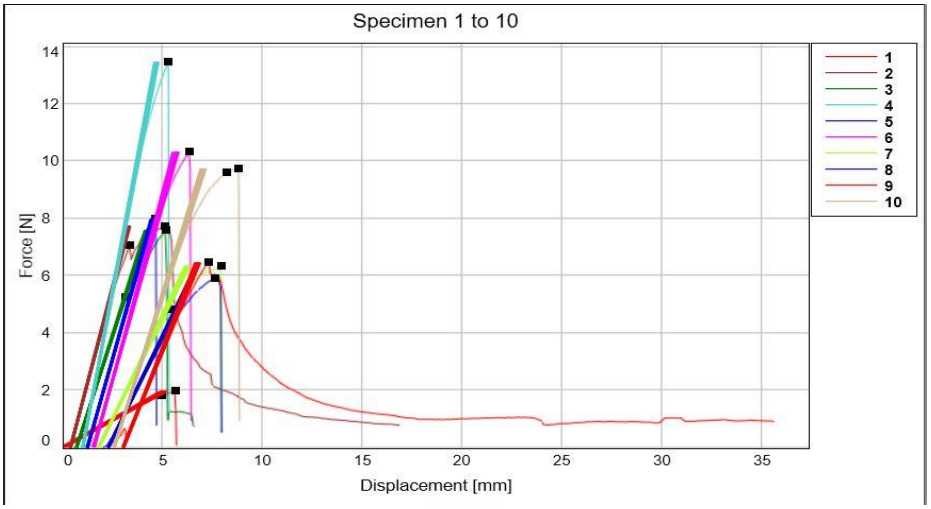

Figure 26: Results for three-point bending tests for ten stems from the same wheat plot, shown together in the Bluehill software interface.

The slope of the linear region from a given load-deflection curve (see Figure 13 in Chapter 1) is used to calculate stiffness, E, also referred to as Young's modulus. Max load is another key element of each load-deflection curve. Max load, also known as breaking strength and ultimate strength, is generally proportional to flexural stiffness. If stiffness and breaking strength are not proportional, it could be because premature breakage is caused by a morphology issue (i.e., there is an irregular bend angle at the tested node, and premature contact is made with the test anvil). Lastly, the type of break (e.g., snap, splinter, or crush) is assessed [40].

Break type identification (Figure 27) is used to characterize the morphological weakness of a stem. Crushed stems have collapsed internode walls. Snapped stems have instantaneous breaks just above

or below the node. Splintering occurs in the internode, though it was observed in less than 1% of stems tested in this study.

Figure 27: Break types of wheat stems. Cornwall et al., 2021 (Fig. 1, [41]).

Results

For 2020, SOCEM trials correlate well with observed lodging percentages across the Pacific Northwest region. Comparing SOCEM and three-point bending test results, an R^2 linear correlation value of 0.54 was achieved in 2020 and an R^2 of 0.31 was achieved in 2021.

SOCEM vs Three-Point Bending Test Results

The SOCEM device provides valid results. This is exhibited by an R^2 value of 0.54 compared to three-point bending results in the 2020 trials and an R^2 value of 0.31 compared to three-point bending results for the 2021 wheat trials. Results are shown in Figure 28.

Results from Hard Winter wheat varieties for 2021 were skewed due to non-typical morphology, and therefore these results were not included in this report's analysis. Non-straight stem morphology (see Figure 29) caused skewed results for three-point bending tests, because the true length of the specimen between the vertical supports is not in keeping with the span length assumption of the experiment.

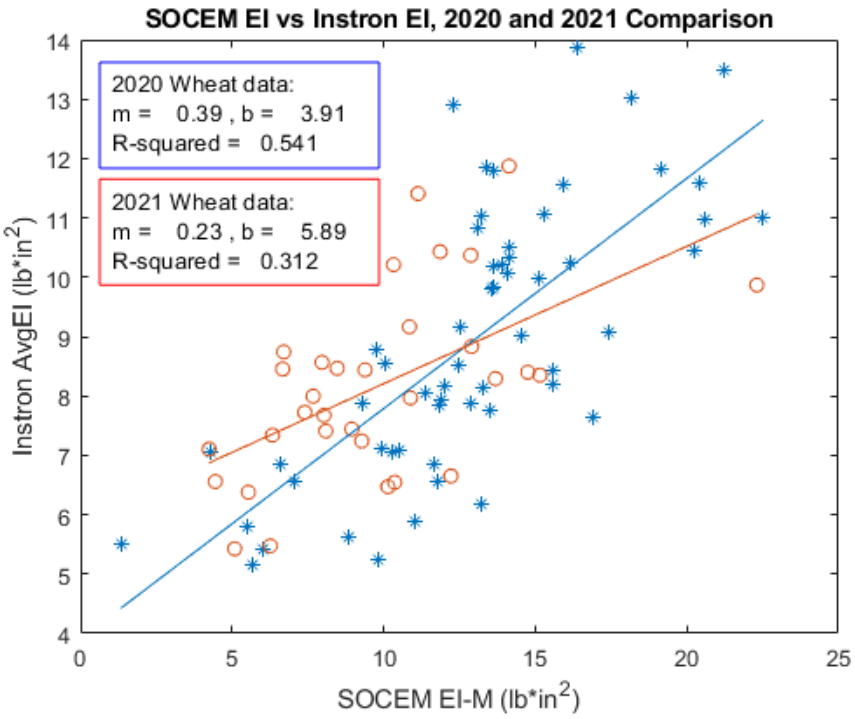

Figure 28: SOCEM vs Instron compiled flexural stiffness results for the 2020 and 2021 wheat trials. 2020 data includes 57 small plots representing 15 genetic varieties of wheat from Clearfield, Soft White Winter, and Hard Winter classes. 2021 data includes 32 small plots representing 8 genetic varieties of wheat from Clearfield and Soft White Winter classes.

Figure 29: Non-typical morphology of a Hard Winter wheat stem collected in 2021. This specimen is from plot HW122, representing the MT1745 variety. This pattern has been observed in wheat that lodged earlier in the season and then self-corrected through gravitropism.

Averaging the results for all replicants from each variety shows a clear linear correlation, with an R^2 value of 0.93, shown in Figure 30. Averaging is necessary to overcome error and to arrive at pervasive genetic-based performance trends. Sources of error include environmental variation throughout a field and non-homogeneity of planting density within each plot.

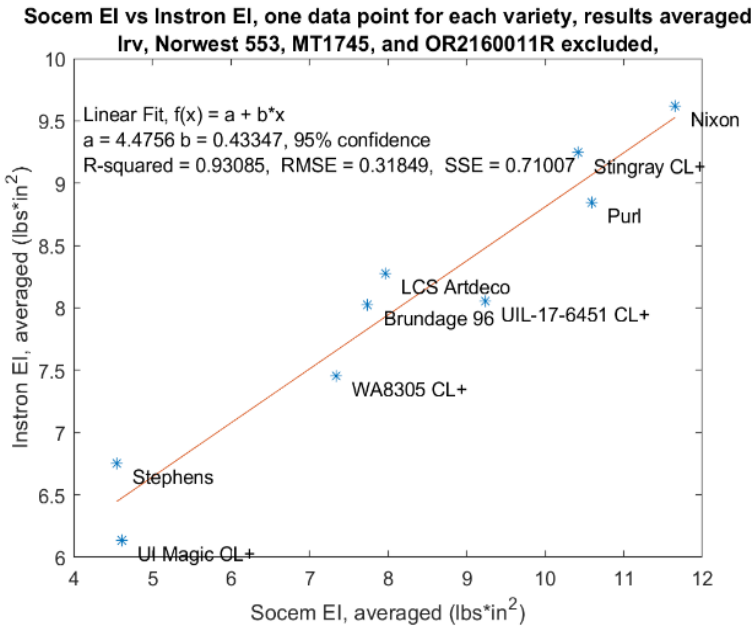

Figure 30: Average performance for each variety tested in 2021 Instron and SOCEM trials, each represented by three or four replicants.

SOCEM vs Historical Stalk Lodging Results

Historical lodging data is recorded as a percentage of crop area lodged for each genetic variety. High SOCEM flexural rigidity values are expected to correlate with low percentages of observed lodging, and vice versa. In other words, negative sloped trendlines are expected in scatter plots that comparing flexural stiffness data versus historical lodging rates. An example of recorded lodging rates is available in Figure 58 in Appendix A, and a scatter plot of this data is shown in Chapter 1 in Figure 10.

Small grain performance reports are generated by researchers, typically yearly, to inform the cereals industry about the experimental quality of common genetic varieties. For this study, reports including lodging data were compiled from the University of Idaho Northern Idaho Small Grain and Grain Legume Research and Extension Program [42] [43] [44] [45].

Validation of the SOCEM device is supported by the comparison between SOCEM 2020 trials (performed in Moscow, Idaho) and the small grain report lodging data represented in Figure 31. The historical lodging data included in this figure represents average performance from fields in these municipalities in Idaho: Bonners Ferry, Craigmont, Genesee, and Moscow. Due to variation over time and location, an ideal correlation would include varieties from the same field in the same year.

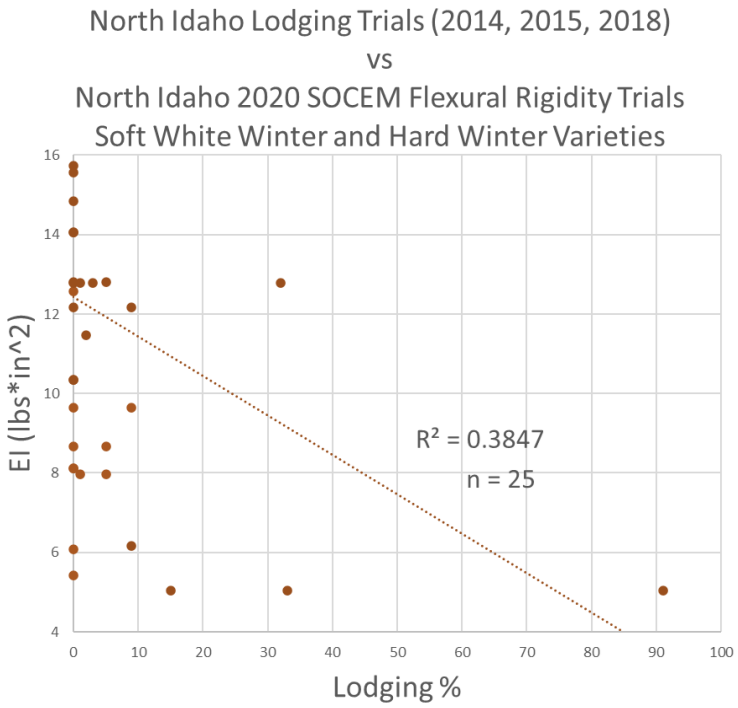

Figure 31: Historical lodging rates from North Idaho, compared to SOCEM 2020 flexural stiffness results.

However, lodging does not occur consistently enough to make lodging rate a reliable metric of comparison. Many years, little to no lodging is observed for any variety, so no useful variation in performance is provided by the small grain report. This is a key reason why SOCEM flexural rigidity data could be more useful than observed lodging percentage data.

Discussion

To prove that results from the SOCEM are valid, a linear correlation should be achieved between the results of the SOCEM and another trusted method. Linear correlation has long been considered a useful metric for determining the validity of data, ever since 1885 when Galton first introduced the concept of linear regression [46]. Typically, linear correlations above 0.3 are considered meaningful, with R^2 values about 0.7 being quite desirable [47] .

R^2 values have to do with dependance, and good R^2 values indicate that results from multiple sources depend on the same underlying factors [47] [48]. SOCEM and Instron correlation in this study may be imperfect because of error; however, results also differ because dependance differs.

If the results of both the Instron and the SOCEM are perfectly identical, the slope of the best fit line should be m=1.0, with an R^2 value of 1.0 [48]. This would mean that every result from the SOCEM is identical to the result from the Instron. However, a perfect R^2 value of 1.0 is impossible, because flexural rigidity factors are different for each device. The SOCEM is a device that measures wholistic lodging resistance, impacted by many possible factors, namely root lodging, whereas the Instron device isolates the load-deflection slope of a stem, measuring only flexural stiffness of the stem.

Even without perfect correlation between the SOCEM and the Instron, the range and domain of results is similar for both datasets taken in 2020 and 2021. Additionally, correlation is very strong, R^2 = 0.93, when results from all small plot replicants for each genetic variety are averaged, as seen in Figure 30.

In SOCEM testing, stem density has been assumed to be homogenous. In fact, stem density can vary wildly within even a small experimental plot. The most substantial outliers from the SOCEM 2021 wheat strength results were very high flexural rigidity values which came from points of high force performance within plots that had artificially low stem counts. Accurate stem counts can be achieved by changing the push direction of the SOCEM to perpendicular relative the direction of planted rows, rather than parallel. Perpendicular testing methods (i.e., “side hits”) are discussed further in Chapter 3 and Chapter 4.

Incorrect stem height is a major source of error in SOCEM results. Stem heights vary within small plots due to uneven cutting during harvest as well as uneven ground. Error due to ground variation can be reduced by using an additional sensor, a height sensor to measure the distance between the ground and the load cell. Managing stem height variation is addressed in depth in Chapter 4.

Improving the correlation between Instron results and SOCEM results should not be a goal in the immediate future. The validation experiment has served its purpose, to prove that the SOCEM offers useful results. The SOCEM results can be improved, but a new heuristic should be used to determine continued goodness. In other words, the human labor cost associated with three-point bending testing in a laboratory is no longer necessary, and it is recommended that the SOCEM process become entirely field based in the future.

Comparison to historical lodging is inexact but is still useful to show a general inverse correlation between observed lodging percentage and flexural rigidity results from the SOCEM. Therefore, future developers of the SOCEM should take care to test crop varieties of low, medium, and high rates of typical lodging performance, so that comparison can be readily made.

The SOCEM device provides an objective numeric measure of stalk lodging resistance that can be determined without relying on crop failure comparisons. It is not hard to imagine that flexural stiffness results from SOCEM devices might soon supplant lodging rates published in small grain reports as a useful and easily communicated measure of stalk lodging resistance.

Chapter 3: Improvements and Investigations

The SOCEM device is being continuously developed for improved accuracy of results, ease of use, and broadening of application. Recent hardware component additions include a wood laser cut keyboard tray, a larger screen, and a detachable storage box. Software has been developed to improve the user experience, reduce the data processing time, and to increase accuracy of results. In terms of analysis and visualization of data, tools have been developed for SOCEM force and flexural rigidity results to be visualized in 3D models in compact FBX files. To improve methods of experimentation with the SOCEM, investigations have been made into ideas such as multiple passes and passes from the side direction rather than from the front of a small plot. In addition, Instron results have been validated by investigating the relationship in performance between the lowest two nodes on sampled stems. Each of these improvements and areas of investigation are highlighted in the sections below.

Hardware Improvements

Several small annoyances can add up to hard days in the field and erroneous data collection. Hardware has been developed to minimize common problems. Solutions are listed below in Table 4 and are shown in Figure 32. A keyboard tray with cubbies has been installed, and more support in the frame has been added to compensate for the additional weight. There is a larger 7” screen, with

Figure 32: The SOCEM in 2022, prepared for a side hit through a small plot. Improvements include the wooden laser cut keyboard tray, the 3D printed nylon 7" screen housing with attached sunshade, and the removable storage box.

attached sunshade and damage protection. An easily accessible Arduino microcontroller storage compartment has been developed, and a protoboard data shield is connected to the microcontroller to improve connectivity and minimize risk of disconnection. A real-time clock module, type DS3231, has been installed on the Raspberry Pi GPIO pins to allow for accurate file creation times. A load cell protection compartment has been added, alongside a detachable storage box which can hold stem samples and supplies necessary for fieldwork. The rotary encoder bracket hardware has been modified to allow for proper belt tensioning.

Table 4: Hardware upgrades

Common Problem	**Solution**	**Figure**
Screen is hard to see	Sunshade, screen size increase from 5” to 7”	Figure 32
Supplies are difficult to manage	Storage box, hinged, easily removable	Figure 32
Data entry is slow with stylus	Keyboard tray, wireless keyboard, and mouse	Figure 33
Rotary encoder belt slips	Longer mounting slot on rotary encoder housing, for belt tensioning	Figure 34
Load cell is at risk of damage	Load cell protection compartment	Figure 35
Faulty Arduino wiring	Soldered protoboard to replace breadboard	Figure 36
File time stamps are inaccurate	Real Time Clock (DS3231) on Raspberry Pi	

Figure 33: CAD model of keyboard tray.

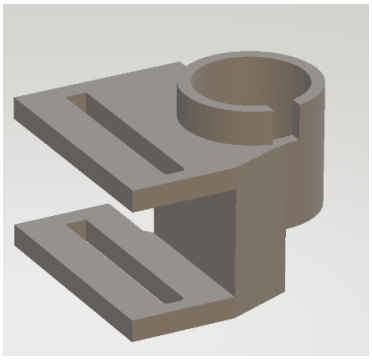

Figure 34: Improved design of the rotary encoder mounting hardware, with overly long mounting slots to allow for belt tensioning.

Rather than dedicate the SOCEM to running only StemBerry software, launched at startup, the onboard computer with Raspian OS launches into the standard desktop environment, offering a familiar graphical user interface. The purpose of this is to allow for creative development in the field if users are struck with inspiration to write new data-handling Python scripts for new experimental

methods. As a research device, wheat breeders should be allowed the freedom to test fresh ideas, and so the desktop environment with a keyboard may continue to be the standard beyond laboratory development. The improved screen housing has openings for ease of access to all USB ports and the microSD card slot on the Raspberry Pi.

Software Improvements

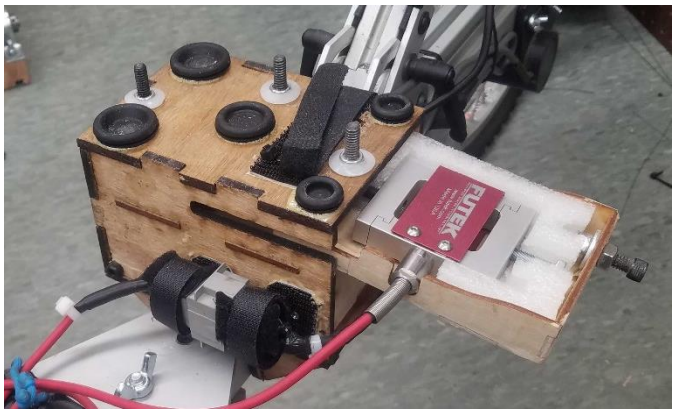

Figure 35: Load cell protection drawer with custom foam.

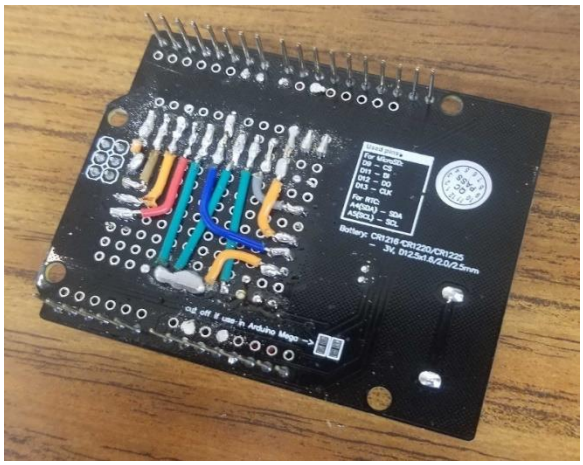

Figure 36: The Arduino protoboard datashield, with soldered connections to supplant the need for jumper wires.

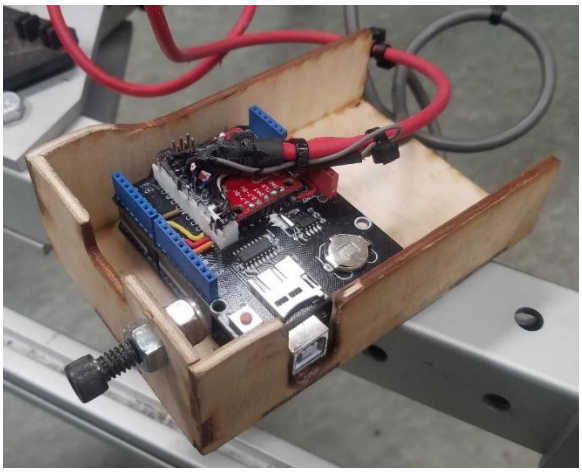

Figure 37: Arduino with protoboard datashield and snap-in connections for the load cell and rotary encoder.

StemBerry software, written in Python 3 (Python Software Foundation, Wilmington, Delaware, USA), has been developed to be easier to use, especially in terms of file naming. Immediately following each SOCEM test, data oversight can be performed through graphical peak selection (see in Figure 63 Appendix A for a PeakClick output image), so post-processing is no longer necessary. Previously, when using SOCEM_DataAnalysis.py, users had to type in plot heights, plot names, and variety

names, even though this data was already known and had been entered elsewhere. The time and labor to type in this information caused a significant bottleneck in data processing.

Software elements have been improved to enhance the serial connection between the microcontroller and the onboard computer, resulting in increased sampling frequency and reduction of data loss. Time vector values are now set by the microcontroller at the time as data creation, rather than by the StemBerry program at the time of data transfer.

For StemBerry Version 70 and beyond, multiple CSV (Standard RFC 4180) files are output from each plot. These include a file for pretest values, a file for posttest values, files for raw data from each push, and user oversight graphical choice data from each push. Upon completion of each plot, a button click of the "Compile" button in the StemBerry interface triggers the compilation of data from these multiple CSV files into a single XLSX (Microsoft Excel Open XML Spreadsheet, Standard ECMA-376, ISO/IEC 29500) file. MATLAB (MathWorks, Inc., Natick, Massachusetts, USA) scripts have been developed to import all columnar data from numerous multipage XLSX files in a folder into one compiled MATLAB table, such that all data from a field of plots is available in a single table, which can be saved as a single MAT-file (Version 7.3, MathWorks, Inc.).

Instron (trademark registered to Illinois Tool Works Inc.) testing was improved for the 2021 data by altering the three-point bending test method (.im_ccyclic filetype, developed by Illinois Tool Works Inc.) used by Bluehill Universal software (Version 3, trademark registered to Illinois Tool Works Inc.). Changes in the test method dictated that each stem was automatically cycled, between 0.3 N and 0.8 N, within the linear region of deflection prior to the push-to-failure phase. This was meant to

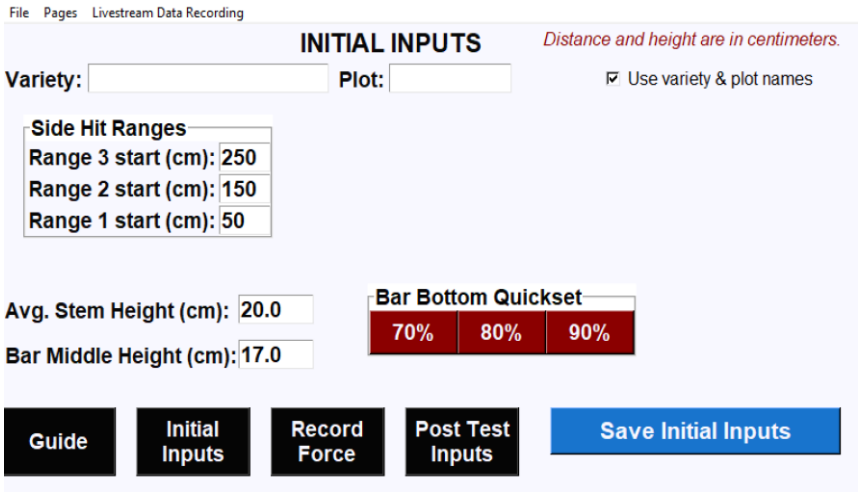

Figure 38: StemBerry Initial Inputs Screen, from StemBerry_v89.py, 2022. The GUI was developed using Tkinter.

test for hysteresis, which is discussed later in this chapter. An added benefit of pre-cycling was that good contact was ensured between the stem and the anvil throughout testing such that the automatic load-deflection slope assessment performed by the Bluehill software was more accurate. The test method was also altered to continue deflecting the sample until it failed entirely, with deflection stopping after a 99% drop relative to an automatically identified maximum force value. Originally, in 2020, tests were set to stop after a 15% drop in force. The longer tests had the benefit of providing longer curves (see Figure 26 in Chapter 2), which could be used to identify characteristic differences between break types (e.g., snap, crease, and splinter). Curve characterization eased break assessment for the experimenter.

Visualization Software Development

MATLAB scripts convert raw data as well as interpolated flexural rigidity results into curves that are represented by STL (developed by 3D Systems, Rock Hill, South Carolina, USA) objects. Python scripts generate color-gradient maps of numerous labeled STL objects in Blender (Blender Foundation, Version 3.1.2, GNU General Public License, Amsterdam, Netherlands). These maps (see Figure 39) are output as FBX files (trademark registered to Autodesk, Inc.), which retain color and are of emailable size (e.g., 4 mb to 20 mb).

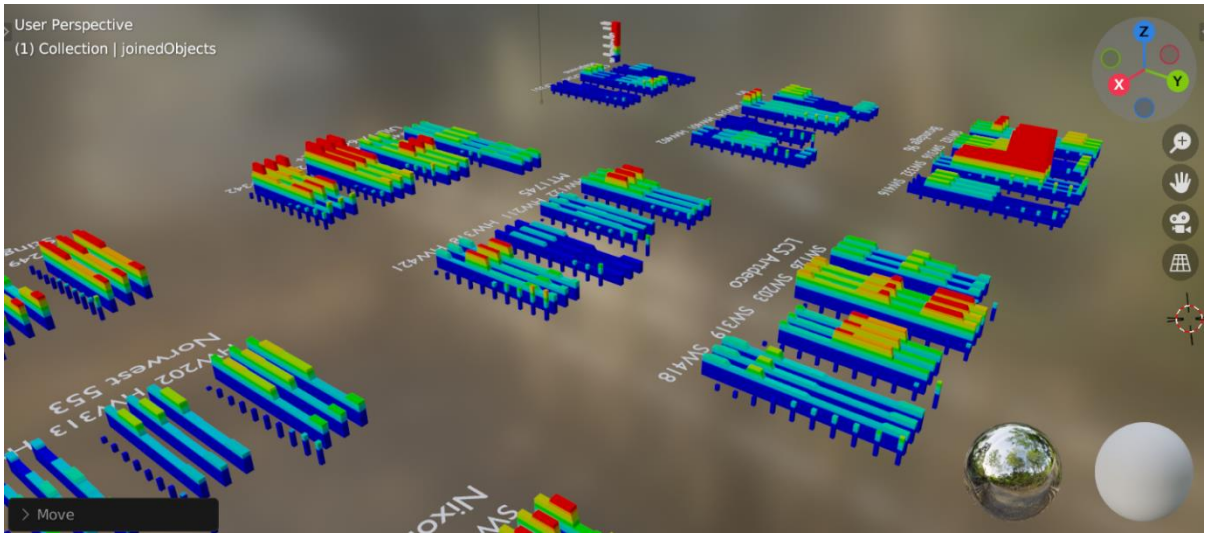

Figure 39: A rendering of a map of 2021 flexural rigidity results in Blender. Here, results are grouped by genetic variety, and Instron stem results are shown next to SOCEM small plot results.

Experimental Design Improvements, 2021

For 2021 testing, a lower surface area Instron anvil (see Figure 24 in Chapter 2) was used, such that premature breakage would not be induced by erroneous contact with a stem internode. As discussed previously in this chapter, in Software Improvements, the Instron method was altered to include an initial cycling of each stem to ensure good contact and to experiment for hysteresis. "Hysteresis" is the disparity in mechanical behavior between loading and unloading, with the internal area between these curves representing energy lost [49]. Little to no hysteresis was not observed in the dry wheat stem samples.

Figure 40: Major internode diameter measurement with calipers.

Figure 41: Major node diameter measurement with calipers.

Prior to each three-point bending test in 2021, each stem was measured for diameter, at four different points: At the node, in the major and minor plane, and at the necked internode, in the major and minor plane. See Figure 40 and Figure 41 for examples of diameter measurement. The major plane is dictated by the bend direction of the node, with the direction of natural lay in the three-point bending fixture due to gravity referred to as the major plane. The minor plane of each stem is perpendicular to its major plane, sharing the axis defined by the contact points on the two vertical supports. Cumulative results for stem diameter measurements are shown in Figure 42.

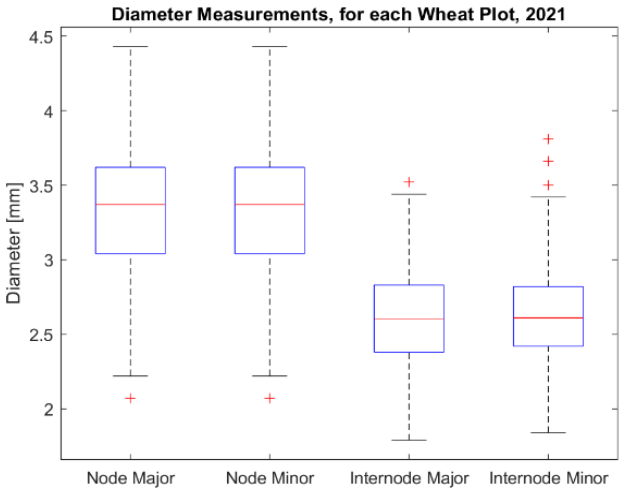

Figure 42: Box plot showing range of 2021 wheat diameter measurements

Knowledge and Investigations

During field testing with the SOCEM, additional questions were formed and investigated outside of the prescribed experimental method. Curiosity has formed over questions shown in Table 5.

Table 5: Questions investigated during 2020 and 2021 testing, beyond the central experiment.

Can diameter sampling in the field replace three-point bending tests in the laboratory?
How does three-point bending and diameter performance differ between the lowest two nodes on a stem?
Can useful information be gleaned from the difference between force values from repeated SOCEM passes?
Does testing from a side direction, perpendicular to the direction of rows, offer any benefits over testing in the standard forward direction, parallel to rows?
Can mass sampling favorably replace stem counting, with increased speed?

Three-Point Bending Stiffness vs Diameter Correlation

Three-point bending tests are labor intensive. To overcome the need for future three-point bending tests, four diameter measurements were recorded for every stem tested in three-point bending in 2021, with the idea that useful relationships may become apparent. Both stiffness and max force (i.e., strength) values were compared to diameter values. The best correlation between stem stiffness and stem diameter (see Figure 43) came from node diameter rather than internode diameter. The correlations shown in Figure 43 are linear, but the relationship between stiffness and diameter is expected to be a fourth-power relationship, as shown in Eq. 2.

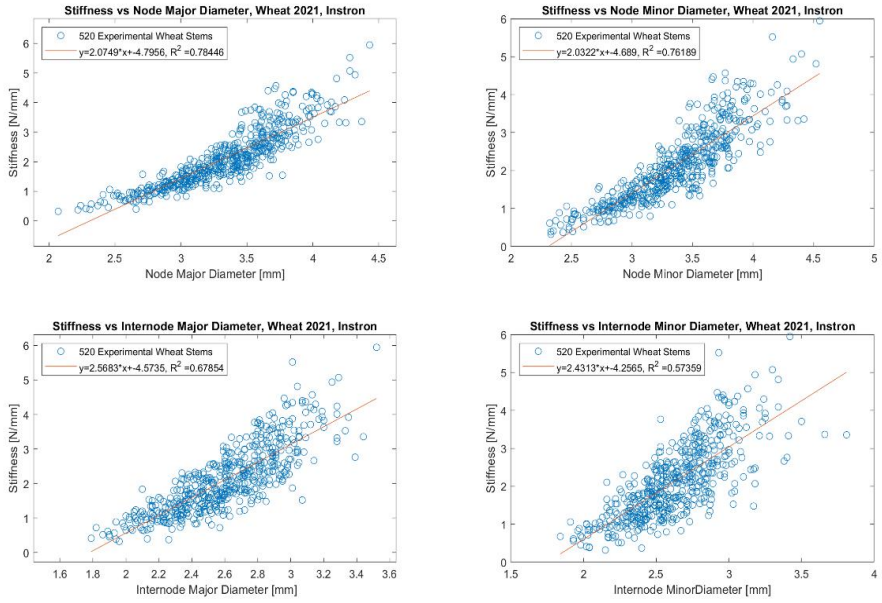

Figure 43: Three-point bending stem stiffness compared to four different stem diameter measurements for 520 wheat stems measured with the SOCEM in 2021.

A single best-fit line was found to summarize the performance of all stems tested in the 2021 standard flexural rigidity correlation experiment. These results can be seen in Figure 44. To study how stiffness and diameter performance is distrubuted for each variety, individual graphs per variety were generated to show all stems from the four small plots from each variety, set against the overall best fit line, as exemplified in Figure 45 for the LCD Artdeco variety.

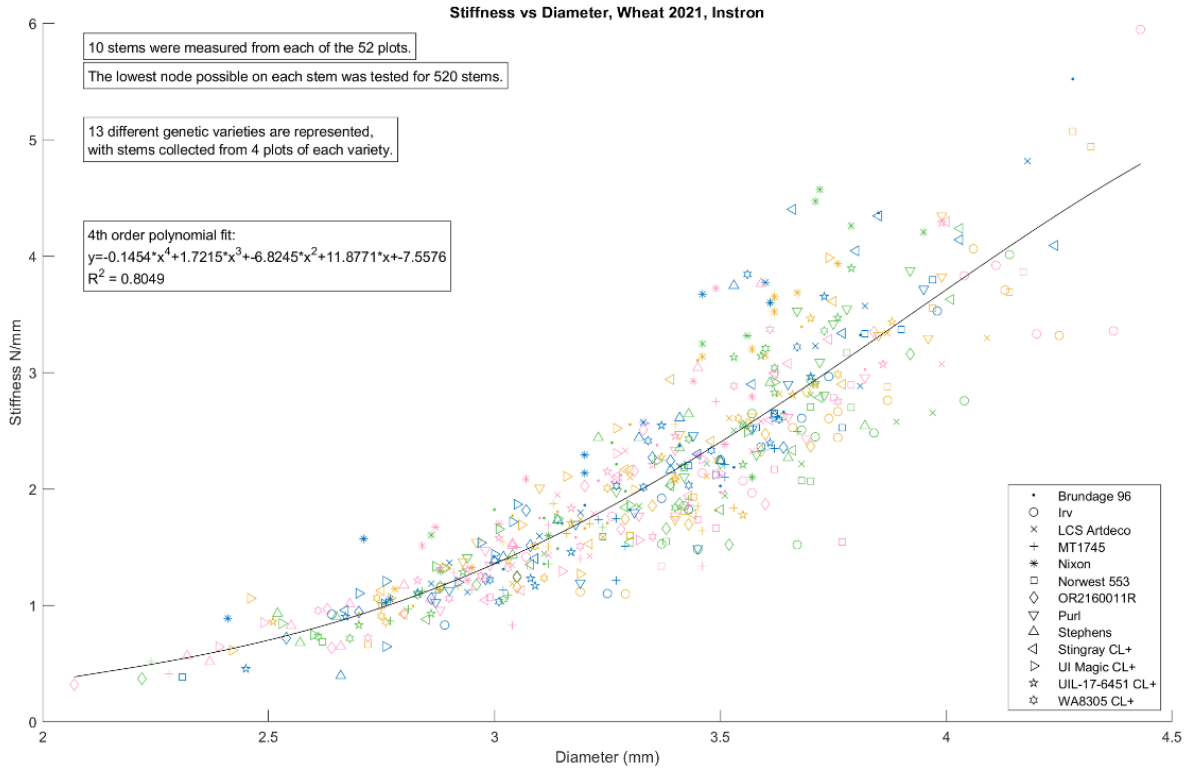

Figure 44: A fourth-order best fit line was found for the major node diameter vs flexural stiffness in three-point bending for all stems tested in the basic experiment for 2021.

Because the strong relationship between major node diameter and three-point bending stiffness has been determined, future three-point bending tets in the lab are not necessary, and diameter measurements can now occur in the field. See Chapter 4 for further information.

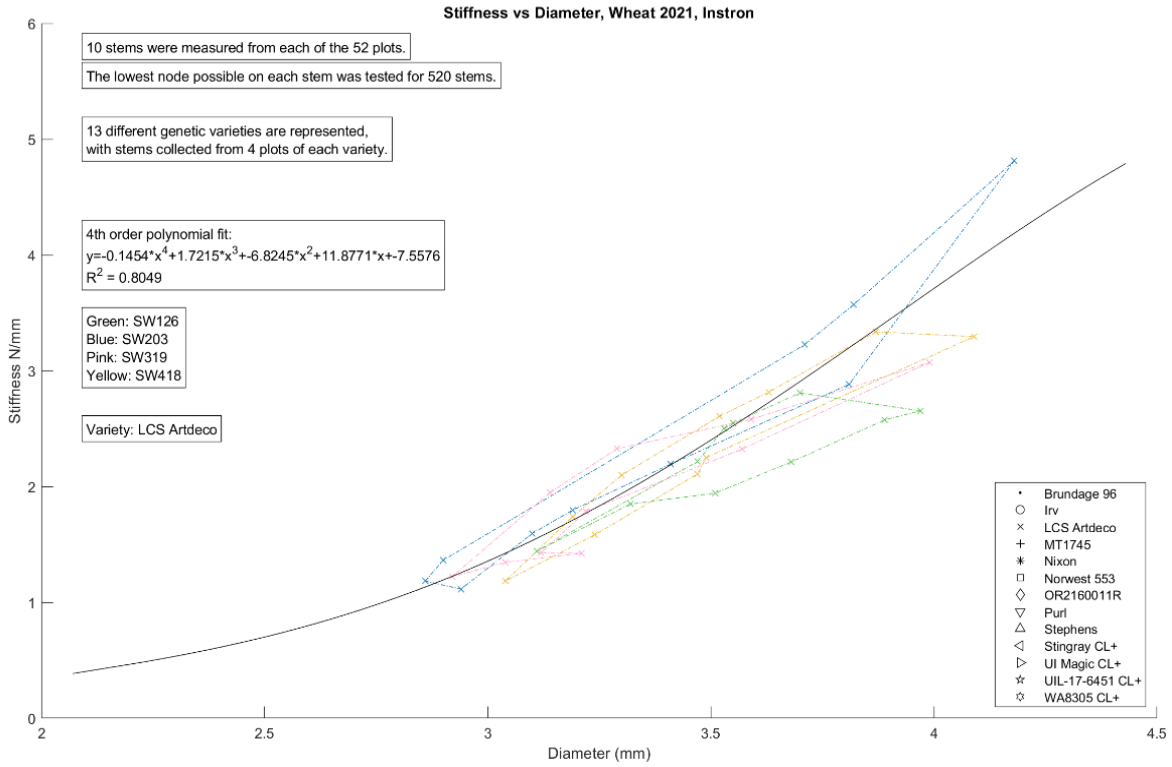

Figure 45: Example of the Stiffness and Diameter results for all 40 stems tested from the four plots from one genetic variety of wheat, LCS Artdeco. 2021. The different colors represent the different plots from which stems are sourced.

Node Choice

During Instron testing, the lowest possible node was tested. Due to the eight centimeter span length of the three-point bending test, or sometimes because of damage, the lowest node could not be used, so the next-to-bottom node up was often used. Using five additonal stems from each small plot tested in 2021, with each stem bearing two testable nodes (see Figure 46), a study was done to analyze whether using the next node up would skew the data. The results shown that there is a normal distribution of performance between the two lowest testable nodes. This allows the conclusion that occasionally using a higher node does not have a substantial impact on the dataset. **Error! Reference source not found.** shows the f lexural rigidity and node major diameter performance for all tested two-node stems. **Error! Reference source not found.** shows a normal d istribution of stiffness, max breaking force (strength), and major node

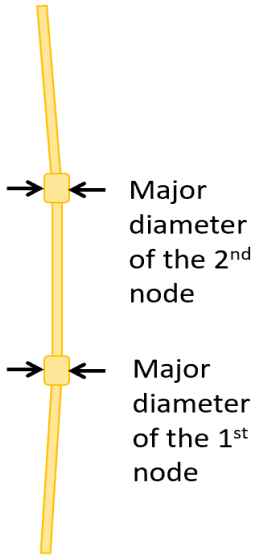

Figure 46: Node comparison, node numbering.

diameter for nodes measured from the same stalk. More thorough data from the node comparison data is available in Appendix A, in Figure 69, Figure 70, Figure 71, Figure 72, Figure 74, and Figure 73.

Force Drop Over Subsequent SOCEM Pushes

The force response of wheat stems is significantly impacted by the height percentage setting of the force bar, relative to the heights of the stems within the plot. This height percentage, of force bar height divided by stem height, can be referred to as the "force bar height ratio." Repeated testing of the same small plot renders a drop in force for each subsequent hit. The magnitude of the force drop between subsequent tests is, in part, a function of the force bar height ratio. This phenomenon has been observed through exploratory SOCEM tests in 2020, 2021, and 2022.

When bending loads are applied to plants, they deflect. Acting as springs, the stems will stand back up once no longer loaded, but full recovery is not immediate or even inevitable. With more applied force, stems will need more time to recover to its original stiffness. Once impacted, lower force performance has been observed for subsequent hits at the same height setting. This could be due to the viscoelastic nature of plants [50], though it may also be due in part to damage incurred by SOCEM testing, causing there to be effectively less contributing stems for each repeat push.

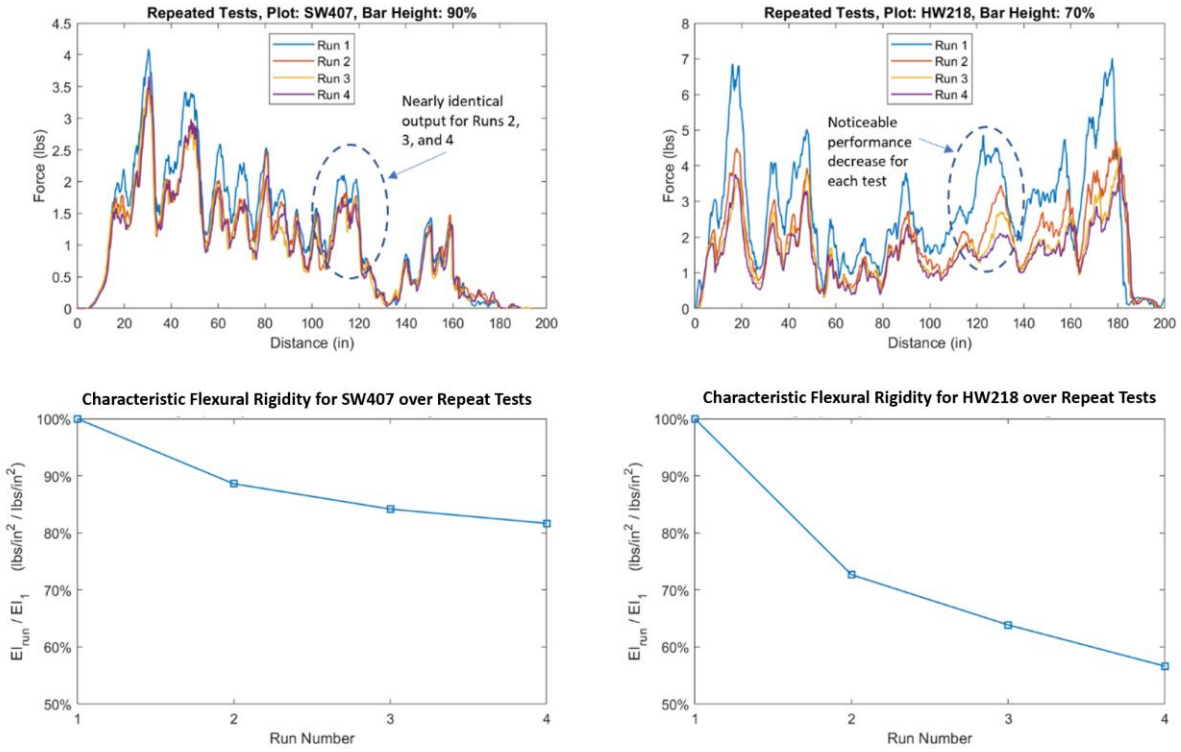

Figure 48: Force-drop has observed for subsequent SOCEM pushes. Relatively lower force bar setting causes greater force drop. This phenomenon can be used to observe proper height setting of the force bar.

Increasing the force bar contact height to something like 90% of the average plot height results in less force-drop effect and less damage. Cycling plants hypothetically might be useful, though cycling should not be done aggressively at heights lower than 70% of the plot height, because this can substantially decrease the performance of the plot. Figure 48 displays force-drop behavior in two different wheat plots tested in August 2021. The wheat plot for which a lower percentage height setting (70%) was used displays a noticeable strength performance decrease with each pass. In this case, the strength performance of the plant is almost cut in half after four cycles. The wheat plot for which a greater percentage height setting (90%) was used demonstrates the capacity of wheat to achieve a near-steady state in terms of strength performance, with over 80% strength still displayed after four hits.

The force-drop phenomenon can be used to provide oversight for proper force bar height setting, due to the relationship between force drop percentage and force bar height ratio. This is discussed further in Chapter 4.

Nine-Cell Scheme

Testing from a side direction with the SOCEM, perpendicular to the direction of rows, is referred to as a “side hit.” Side hits are distinguishable from ”forward hits”, i.e., pushing in the standard forward direction, parallel to rows. The outcome of a side hit is that one force peak is generated for each row contacted.

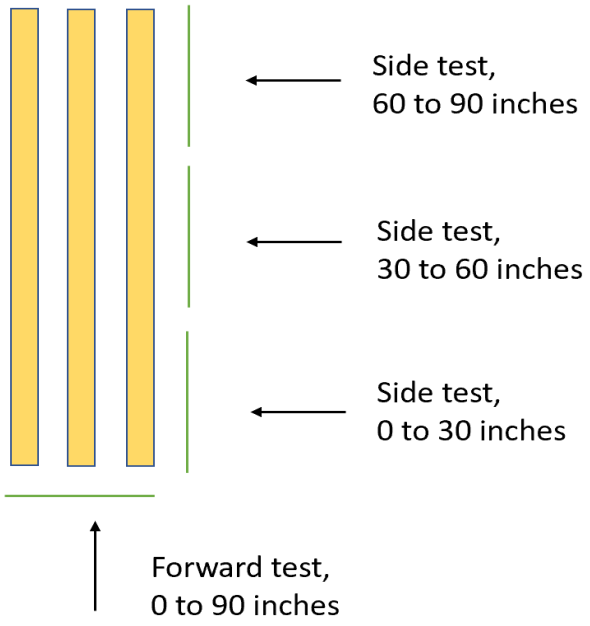

Figure 49: SOCEM testing with a nine-cell scheme, as explored during 2022 experimentation.

Side hits provide the advantage of clear force response peaks in the raw data (see Figure 63 in Appendix A), which can be easily converted into flexural rigidity data, especially because every stem contacted can be acurately accounted for. If three side hits are performed in one small plot, with three rows contacted for each push, the result is nine cells of data. A fourth hit, in the forward direction, provides the raw data necesaary for rich three-dimensional modeling, as seen in Figure 65 in Appendix A. Figure 49 shows a pictoral guide to nine-cell testing.

Each of the nine cells can be assessed for stem count, peak force, and flexural rigidty. Figure 50 shows an example of a three-dimensional rendering of nine-cell force data, comparable to a bar chart with nine bars.

Mass Sampling

Mass sampling could hypothetically help improve the throughput of the Stem counting process. Useful relationships might be found between stem count, stem mass, and structural rigidity. If a strong correlation between count and mass was discovered, the counting process in the SOCEM experimental procedure could be replaced with a mass sampling process.

Mass sampling was attempted during the 2022 data collection season. The scale used had a maximum weight of fifty pounds, and the measurements were shown at a granularity of 1 gram. Wheat stems each weigh much less than 1 gram, and masses of 4 to 8 grams were recorded for

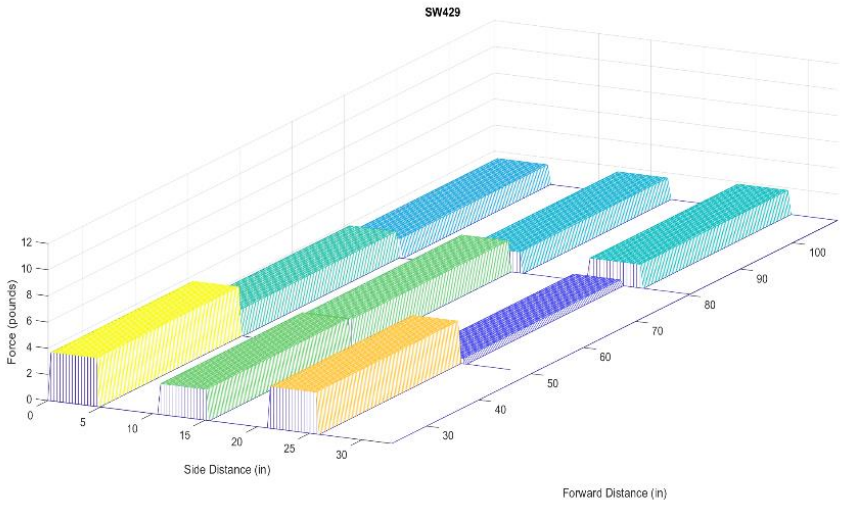

Figure 50: Example of a nine-cell scheme. Here, the average force for each cell is shown.

collections of ten to thirty stems, though these numbers were not trusted due to wind and the lack of granularity in the mass scale.

Mass sampling was labor intensive in 2022, and the current version of the WheatSqueezer hardware (see Figure 62 in Appendix A) was slow to use. Hardware for quick release and quick compression would both increase sampling speed.

Chapter 4: Suggestions for Future Work

The SOCEM device can be further modified to improve user ergonomics and accuracy of results. Additionally, in the future, the device should be scaled to other applications, including automation, multi-acre agricultural testing, and testing with large grains. To accomplish this scalability, alterations and additions can be made to the hardware, the software, and the experimental design. Useful hardware additions can enhance the user experience, address ground height variation, and build toward automated experimentation. Strides in software development can be made to improve experimentation at the small plot level, specifically regarding error reduction, while also building tools applicable to large scale experimentation with a load cell and force bar as a combine harvester accessory. As tools are developed, modularity should be a primary concern in software development such that tools can be scaled to multiple use cases. The experimental design for investigations in the near future should continue to explore useful relationships between measurable factors while providing yearly data to continuously product test and solidify hardware and software elements.

Hardware Suggestions

To achieve optimization of load cell height, the SOCEM can be equipped with additional sensors Potentially useful sensors include an automatic distance sensor used to continuously measure the height between the load cell and the center of a wheel that rolls on the ground underneath the load cell. See Figure 51 for an example of a ready-made height sensor typically used for automotive suspension [51]. Two vertical lead screws could be installed to control the height of the force bar

Figure 51: Example of a ready-made height sensor that can be used to automatically measure height between the ground and the SOCEM load cell. Roverparts.com, 2022 [51].

digitally using the StemBerry interface. This will increase the speed and accuracy of setting the force bar and can later result in automatic height adjustment during a test.

A second load cell would be useful, positioned behind the first load cell by approximately 20 centimeters, to monitor the force variation between the first and second load cell. As discussed in Chapter 3, variation between subsequent testing of force can be used to monitor that the first load cell is set at a proper height. For further monitoring of load cell height accuracy, a third load cell of basic quality and with little or no force bar could be positioned above the primary load cell by a few centimeters, with an ideal reading of zero load when the primary load cell is not positioned too low. This third load cell is applicable only to combine-scale testing, where automatic height adjustment would be useful, though product testing should occur at the small plot scale.

Figure 52, which can be compared to Figure 17 in Chapter 2, shows an overview of the theorized additions.

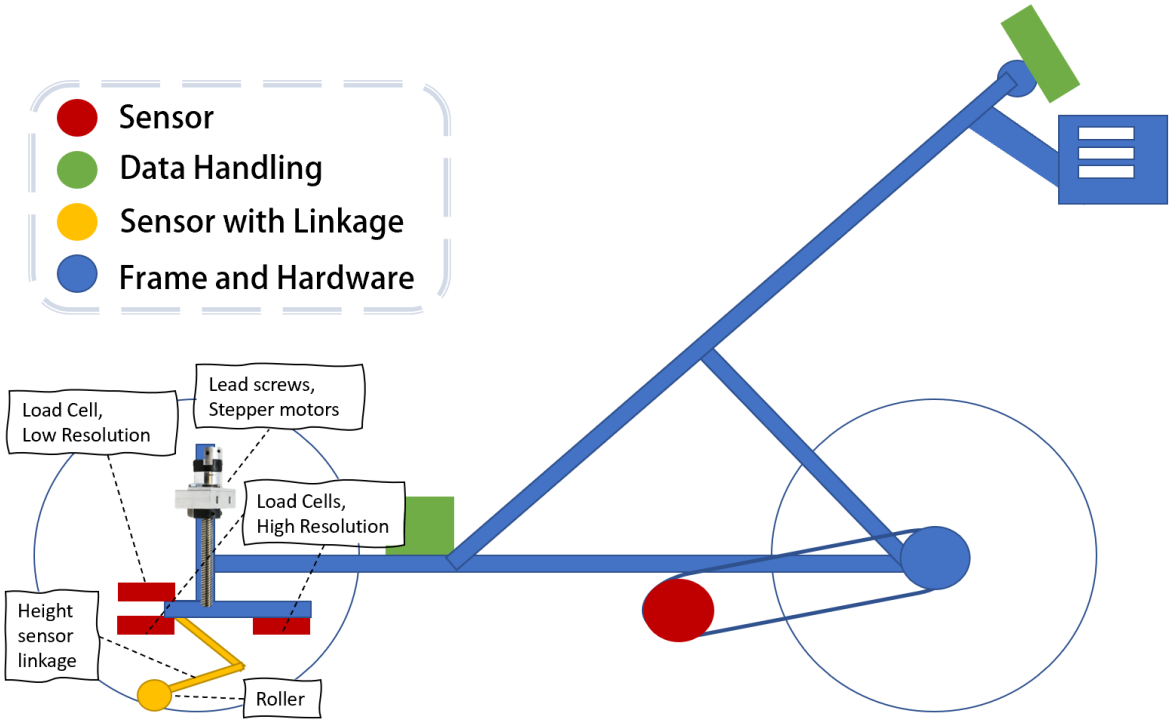

Figure 52: Suggested future version of the SOCEM, with additional sensors. New elements include two additional load cells, two lead screws for digital height adjustment of the floating sensor chassis, and a height sensor that monitors the height of the load cell from the ground. The function the second high resolution load cell is to monitor proper height setting based on the force difference compared to the first high-resolution load cell. The function of the low-resolution load cell above the primary load cell is to monitor for binary load, which should be no load, if the height of the sensor chassis is appropriate for useful data collection.

A camera could be added to the SOCEM to provide video recording of contact with the force bar, from above. A second camera may also be positioned for a side view. These cameras would enable video capture of stem density and could possibly one day lead to the replacement of manual stem counting with counting via a computer vision algorithm.

Beyond sensors, frame components could be improved to make the SOCEM easier to use and transport. The handlebar stem can become more easily removable. The keyboard tray can be made larger, with intent to support laptops. The benefit of using a laptop for SOCEM testing is that future consumers can use StemBerry software on their own computers and thus reduce the cost of the device by mitigating the need for an onboard computer.

Finally, additional hardware tools could be developed for ease of experimentation and to address current problems. A drone would be ideal for overhead images of the field, such that stem density of each plot can be easily examined alongside data during post-processing. The most useful additional piece of purpose-built gear for field experimentation would be a ground height variation

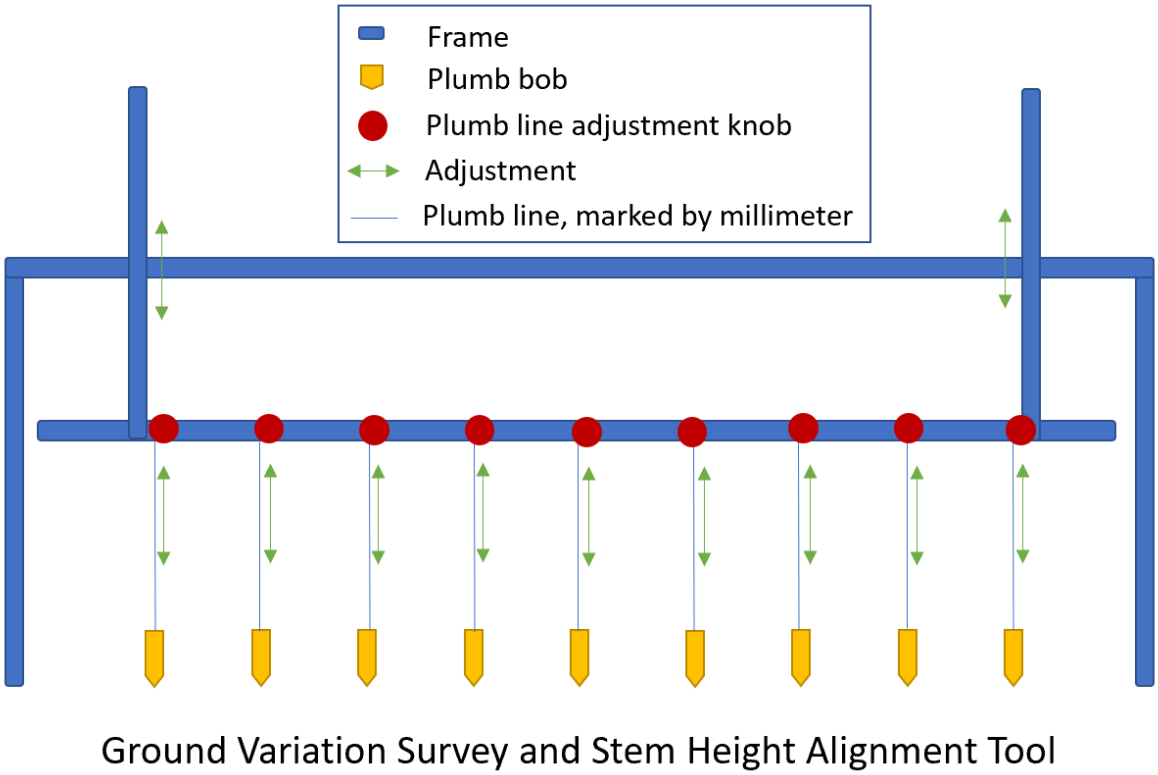

Figure 53: This is the proposed tool for surveying ground height variation in small plots and for aligning stem heights prior to SOCEM tests. Without the plumb bobs, this tool would still be useful for identifying stem contact and protrusion, to assist with stem height leveling, prior to SOCEM testing. The vertical support width should be slightly less than the wheels of the SOCEM, so that the stem height alignment tool can be placed directly next to the force bar of the SOCEM for height comparison and replication.

survey tool, with survey plumb weights and adjustable plumblines that mark distance. A simple drawing of a hypothetical ground height variation survey tool is shown in Figure 53, and an in-depth analysis is provided later in this chapter. The dimensions of the survey tool should mimic that of the SOCEM force bar and wheel width. The survey tool should also have a bar of manually adjustable height, similar to the current force bar, which can be deftly placed tangential to a row of stems prior to SOCEM testing. This will make it easier to level plots, to choose the proper force bar height setting, and to understand the amount of ground variation thus addressing the largest source of error in small plot testing. The best stem leveling tool identified is manual hedge shears.

Software Development Suggestions

Load cell height values from the suggested automatic height sensor should be continually recorded and stored at the same rate as force data. During data collection, force peaks can be automatically assessed and identified by a high percentage drop, such as 80% less than each maximum force value. The Bluehill Universal Instron software, discussed in Chapter 2, uses a percentage drop assessment of this kind, which can be mimicked, resulting in the partitioning of each separate force response and force peak from each row tested in a SOCEM side hit.

When preparing for each test, the suggested lead screws should be used to adjust the height of the load cell by using on-screen buttons in the StemBerry GUI and also with physical buttons mounted on the Arduino compartment near the front of the SOCEM. Furthermore, automatic height adjustment of the force bar during testing can hypothetically be made possible in the future using comparison between force values from the first and second load cell. This comparison relies on distance data supplied by the rotary encoder, and the distance between the load cells must be known. Complexity of StemBerry data collection software would need to increase with these additional features. It is suggested that a simple version of the device be maintained while an additional complex device is developed in parallel. For modularity, StemBerry should be converted from a single script to a folder of modules. The scope of modularity should aim to succeed in both small plot usage and in usage as a combine harvester accessory. In terms of computer hardware, StemBerry software can be run on any device that supports Python, including Windows, Linux, iOS, and Android operating systems. Android devices must be equipped with a serial connection application and a Python IDE application, and automatic window size scaling based on screen size should be developed.

Ideally, data can be collected, assessed, and compiled all in the field. This pipeline has previously required several months between data collection and useful visualization. A clear goal is to achieve outputs for an entire field of plots, in terms of visualizations and compiled numerical results, as soon as data collection is completed. Standard output files from each plot should include a raw data peak selection graph image, a 3D object STL file, raw CSV files, a CSV file for calculated EI results, and a XLSX file of complete numerical data sourced from the multiple CSV files for each plot. An entire data set for a year, including all tested plots, should be represented as a color-gradient FBX file showing all labeled STL objects and as a MATLAB table with all data compiled. Continued validation of results should be shown using scatter plots of flexural rigidity (EI) vs sampled diameter and flexural rigidity vs local historical lodging rates.

The current roadblock in the data compilation pipeline is flexural rigidity calculation. The Python methods that calculate flexural rigidity (EI_Interaction_Fx.py and EI_No_Interaction_Fx.py) often fail due to error. During 2022 data collection and processing, it was common that the inputs fed to these Python methods would cause error, because calculations were outside of the bounds of the codified model. Trigonometric functions were the primary point of failure. More work needs to be done to ensure that edge cases are addressed within the scripts, so that accurate results are calculated without causing software failure.

Experimental Design Suggestions

Side hits (i.e., pushing the SOCEM in a direction perpendicular to planted rows) should be the focus of testing in the immediate future. In 2020, only forwards hits were used. In 2021, forward hits and side hits were both explored. In 2022, three side hits through three rows from each plot were used to explore a nine-cell scheme. In the future, the goal should be to assess each small plot using a single side hit of three or four rows. Data should be collected regarding the stem count and diameter of stems from each row contacted (see Figure 41). The mass of each “cell” (see glossary, Appendix D) may continue to be explored, though the mass testing hardware and procedure will need to greatly improve for speed. See Figure 62 for an analysis of the current WheatSqueezer hardware, which is necessary for mass measurements.

When selecting plots to test, experimenters should choose genetic varieties that are represented in local small grain reports, such that flexural rigidity results can be readily compared to lodging percentage rates in the local area. Prior to harvest, communicate with the field manager to please leave a minimum of 14 cm of stem stubble remaining when small plots are cut, to improve data

quality. At least 20 cm of stem stubble height is preferable, to allow for minimal error reduction and to ease the need for precision. Additionally, experimentation with the SOCEM should be attempted prior to harvest, for full size stems with heads still attached.

Addressing Stem Height Variation

Incorrect stem height is the central confounding factor and source of noise that stands in the way of the SOCEM providing more accurate data. Uneven ground conditions (see Figure 54) cause stem heights to vary when measured from the point of contact with the force bar to the base of the stem. In 2022, these ground height variations were observed to be typically 2 centimeters and up to 4 centimeters for stems that were simultaneously in contact with the force bar. Stem heights also vary in the amount that stems protrude above the force bar, due to uneven cutting during harvest. Excessive stem length above the force bar causes inaccurate results, because the low-angle beam deflection assumption (see glossary) is broken and because the canopy interaction increases beyond the bounds of the current Stacking Beam model. Accurate data can be achieved when all stems are contacted between 70% and 90% of their length and when the length from the ground to the point of force bar contact is known with precision.

Figure 54: Ground variation under a small plot.

Stem protrusion equalization above the force bar (see Figure 55) has two known solutions, which can be used separately or in tandem. First, stem heights can be equalized to make even contact with the force bar such that all stems protrude above the bar at a homogenous length. Second, the force bar height and angle can be adjusted during testing to contact the stems at an ideal point of low protrusion. These two sources of height variation occur on both the axis of travel and the axis of force bar contact. There are multiple approaches to both stem height equalization and mid-test load

Figure 55: Three rows prepared to be side tested with the SOCEM in 2022. Notice variations in the height of the ground at the base of stems.

cell adjustment, though neither of these outcomes can overcome ground height variation. Overcoming ground unevenness can be approached in one of two ways. First, the conditions of a SOCEM trial can be controlled such that error due to ground unevenness is overcome (i.e., through testing stems that are at least 14 cm tall and through careful pruning of stem heights and careful selection of each test region). Second, the ground can be mapped, and then stem height variation can be compensated for.

Ground mapping may be worthwhile for achieving accurate results and for ultimately automating SOCEM data collection; however, the cost of the ground mapping is increased hardware and software complexity.

It may be acceptable that stems within a single plot are contacted at differing height ratios within some range. Because ground variation does not typically exceed four centimeters, and because stems should be contacted between 70% and 90% of their height, stems would need to be at least 14 centimeters tall (and 18 centimeters tall measured from the ground at low points within the plot) for ground variation to not exceed height variation limits. See Table 6 and Figure 56.

Shortest stem (cm), at high point on ground	Tallest stem (cm), at low point on ground*	90% of shortest (cm)	70% of tallest (cm)	Overlap (cm) must be positive
10	14	9	9.8	-0.8
11	15	9.9	10.5	-0.6
12	16	10.8	11.2	-0.4
13	17	11.7	11.9	-0.2
14	18	12.6	12.6	0
15	19	13.5	13.3	0.2
16	20	14.4	14	0.4
17	21	15.3	14.7	0.6
18	22	16.2	15.4	0.8
** Assuming 4 cm maximum ground variation*				

Table 6: Ground variation analysis demonstrating that the SOCEM can accurately measure plots that are at least 14 cm tall when the ground under a plot varies by 4 cm or less.

Height of the load cell greatly impacts the results of calculated flexural stiffness. Due to variation in ground level, the load cell height data from the automatic distance sensor should be considered to represent multiple possible heights (plus 2 cm, plus 0 cm, and minus 2 cm), in the same way that multiple levels of interaction (full interaction, intermediate interaction, and no interaction) are currently considered in the EI calculation software. Long stems and no canopy interaction would result in the highest possible EI. Short stems and full interaction would result in the weakest possible EI. The average flexural rigidity value would result from average canopy interaction and the height sourced directly from the automatic height sensor.

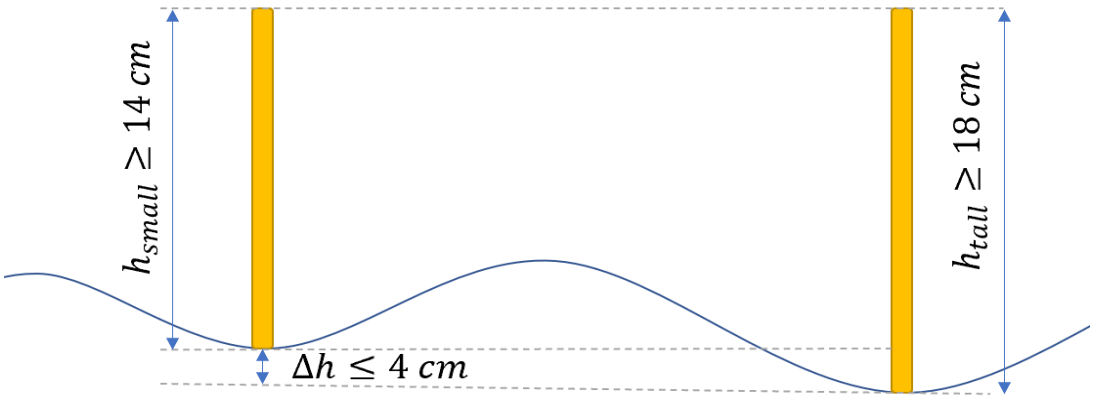

Figure 56: Ground height variation parameters. If the ground variation is up to 4 centimeters between plant bases, and stems can be contacted between 70% and 90% of their height, then the shortest plant must be at least 14 cm from its base to the contact point with the SOCEM force bar to overcome error.

Python as a Standard

Python and MATLAB, as of 2022, are both used in the data collection and processing for the SOCEM. Ideally, in the future, Python could be used for all computation.

MATLAB has been used to compile data, analyze trends, and visualize trends. MATLAB has also been used to generate STL files for the 3D visualization of SOCEM tests. MATLAB tables, which can be imported and exported as .mat files, are ideal for handling large data sets. A great feature of MATLAB tables is that whole arrays can be stored in a single cell. If tables are well organized, each row can effectively be treated as an object with columnar attributes.

For 2022 experimentation, flow of data has been simplified by allowing raw data to be processed in the field, immediately following each test. Also, the organization of output files has been improved, with multiple CSV files generated during different portions of a SOCEM test to collectively represent each small plot as a single data object. Once a plot has been completed, all useful data from the multiple CSV files are saved to a multipage XLSX file, which can then easily be imported into MATLAB as a complete unit using spreadsheetsToTable_v3.m (Appendix B2). Hypothetically, all data compilation, analysis, and visualization can be done in Python. Quite a few MATLAB scripts have been developed, but the migration to pure Python should be a medium-tier priority in ongoing development of the SOCEM.

The primary advantage to Python is that the license is free and that it is easily accessible the world over. The primary advantage of MATLAB is the ability to create, reference, and export tables. Object oriented programming in Python could supplant the need for MATLAB tables, hypothetically using the Python pickle module, the Python json module, through Python dictionaries, or even SQL files. The Python pickle module is a functional and simple solution for saving Python classes [52].

The SOCEM as a Combine Harvester Accessory

SOCEM data can be useful to farmers to inform crop management decisions. A load cell and a force bar attached to a combine during harvest, behind the cutting apparatus, is enough to provide data about stem force response and density throughout an entire agricultural field. Raw force data can be used to generate 3D maps of entire fields, which farmers can use to make field management decisions. Digital maps, in the form of FBX files that retain color gradient and labels, can be easily stored and compared from year to year. High resolution GPS data would allow for accurate spatial distribution of force data from a single continuously recorded file.

It is possible that side hits can be performed with a harvester combine accessory version of the SOCEM. Side hits with the current SOCEM device require picking up the device, placing it to the side of the plot, and manually pushing it through the plot over multiple rows. Travel direction for combine harvesters typically follows row planting direction, and side hits could be performed by a linkage, such as a load cell mounted on a wiper or on a rotating member. Automatic diameter sampling, stem counting, and mass measurements are also possible through complex hardware and software that could be implemented on a combine.

Appendix A: Additional Figures

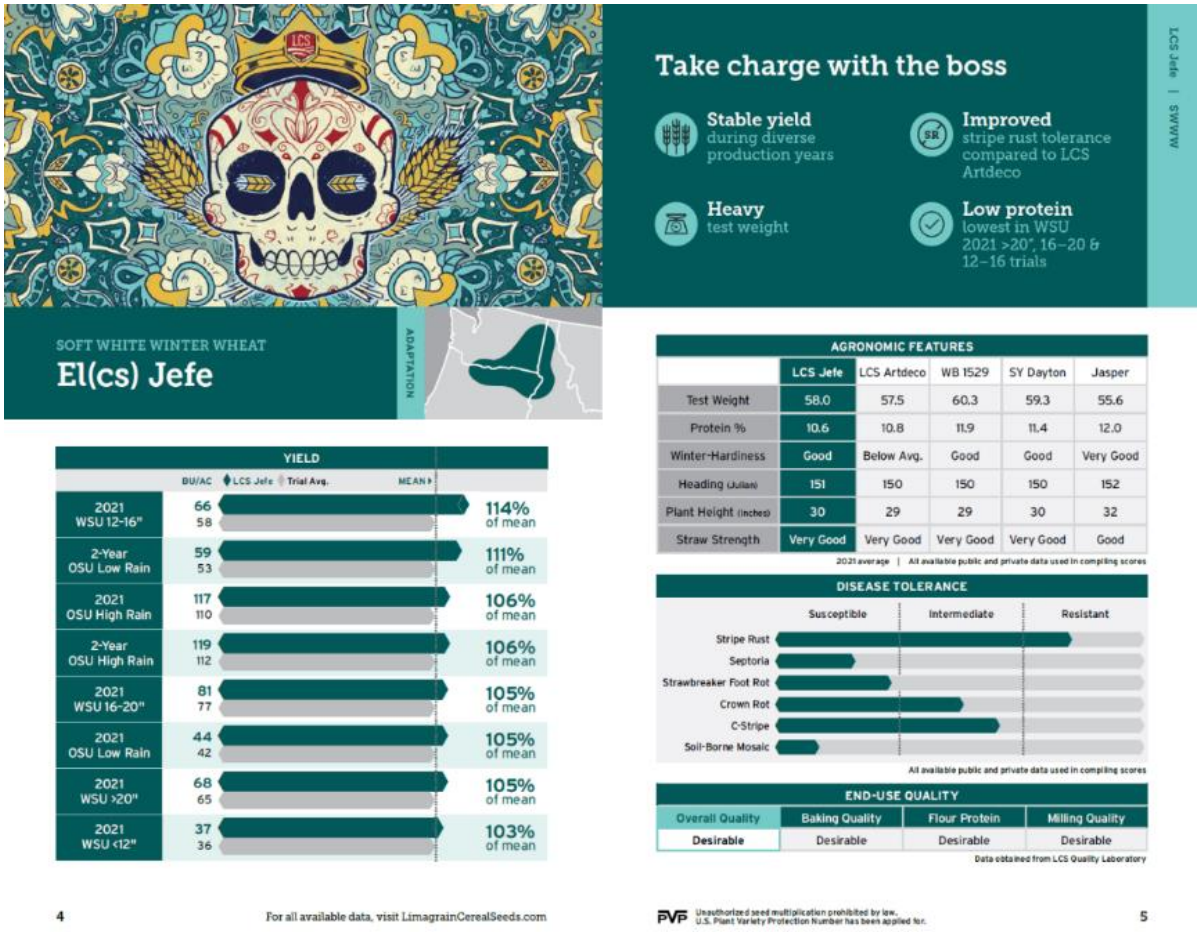

Figure 57: Sales brochure for a genetic variety of wheat seed. Here, flexural rigidity (i.e., stalk lodging resistance) is referred to as "stem strength". Limagrain Cereal Seeds, 2020 [7].

TABLE 2.—*Breaking strength of straw, lodging, and culms per acre of winter wheat varieties at harvest in 1927*

Variety	Accession Record No. *	Culms per acre at harvest (000 omitted)	Breaking strength of straw in pounds per straw	Lodging
				Per cent
Hussar	4843	2,489	1.08±0.009	2.0
Harvest Queen	6199	2,841	1.09± .008	.7
Shepherd	6163	2,404	1.08± .007	3.5
Iobred	6934	2,848	1.06± .009	2.0
Sherman	4430	2,629	1.02± .007	3.3
Kawvale	5274	2,853	1.00± .008	10.0
Fulcaster	6471	2,972	1.00± .007	8.3
Tenmarq	6936	2,698	.99± .008	9.3
Nebraska No. 6	6249	2,692	.99± .008	.3
Turkey	1558	2,488	.97± .006	.7
Currell	3326	2,865	.95± .008	.3
Turkey	5592	2,769	.93± .006	1.7
P 1066×Marquis	Kans. 442	2,794	.92± .006	5.3
Altara	5797	2,803	.90± .006	.3
Kharkof	6206	2,388	.90± .006	1.3
Superhard	8054	2,758	.90± .007	16.7
Blackhull	6251	2,665	.89± .006	14.7
Marquis×Kanred	Kans. 443	3,241	.88± .006	8.7
Hard Winter Defiance	6205	2,871	.85± .006	.0
Kanmarq	6937	3,163	.84± .006	11.7
Newturk	6935	3,063	.84± .006	0
Zimmerman	6211	2,852	.84± .006	0
Regal	7364	2,615	.82± .007	0
P 1068 selection	Kans. 2658	2,906	.82± .008	17.7
Nebraska No. 28	5147	2,607	.81± .007	4.3
Fulhard	8257	2,822	.80± .011	0
Kanred	5146	3,085	.79± .004	13.3
Early Kanred	Kans. 2521	2,549	.62± .011	15.0

* See note *a*, Table 1.

Figure 58: The first published instance in American academia of the comparison between lodging rates of wheat alongside breaking strength results. Salmon, 1931 [11].

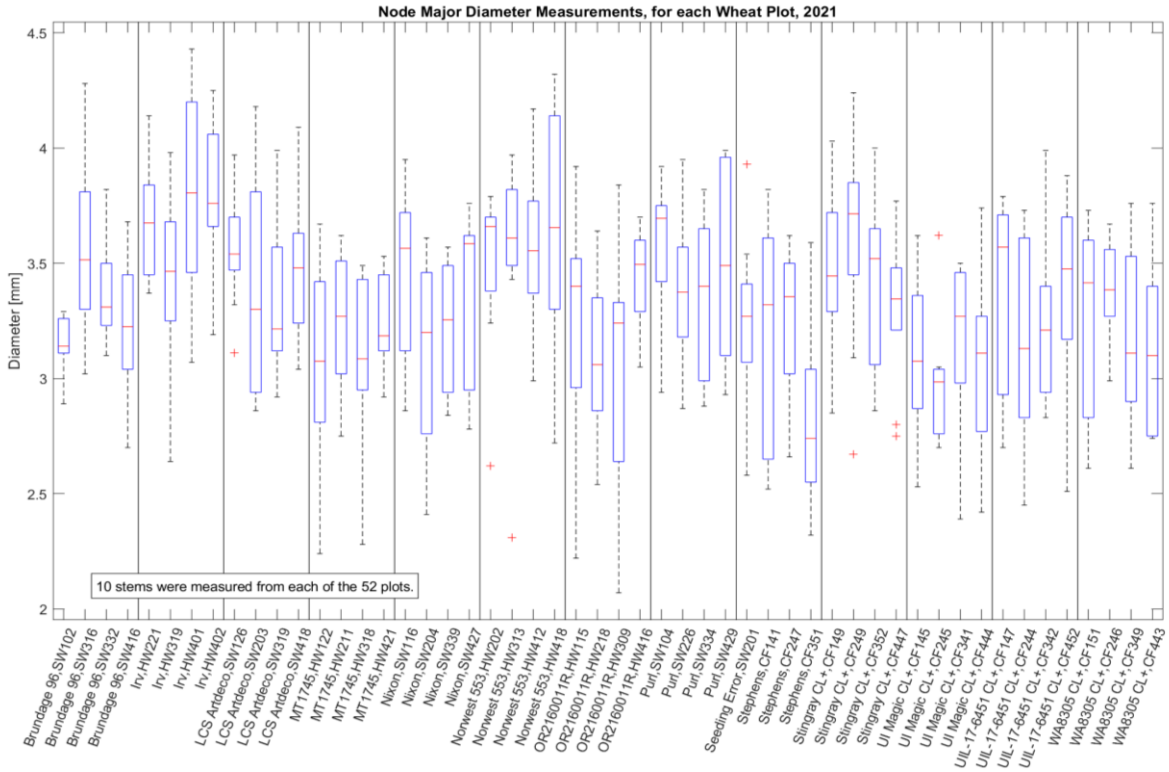

Figure 59: Box plots for the range of stem diameters from each plot of wheat. 10 stems were measured from each plot. Bennett 2022.

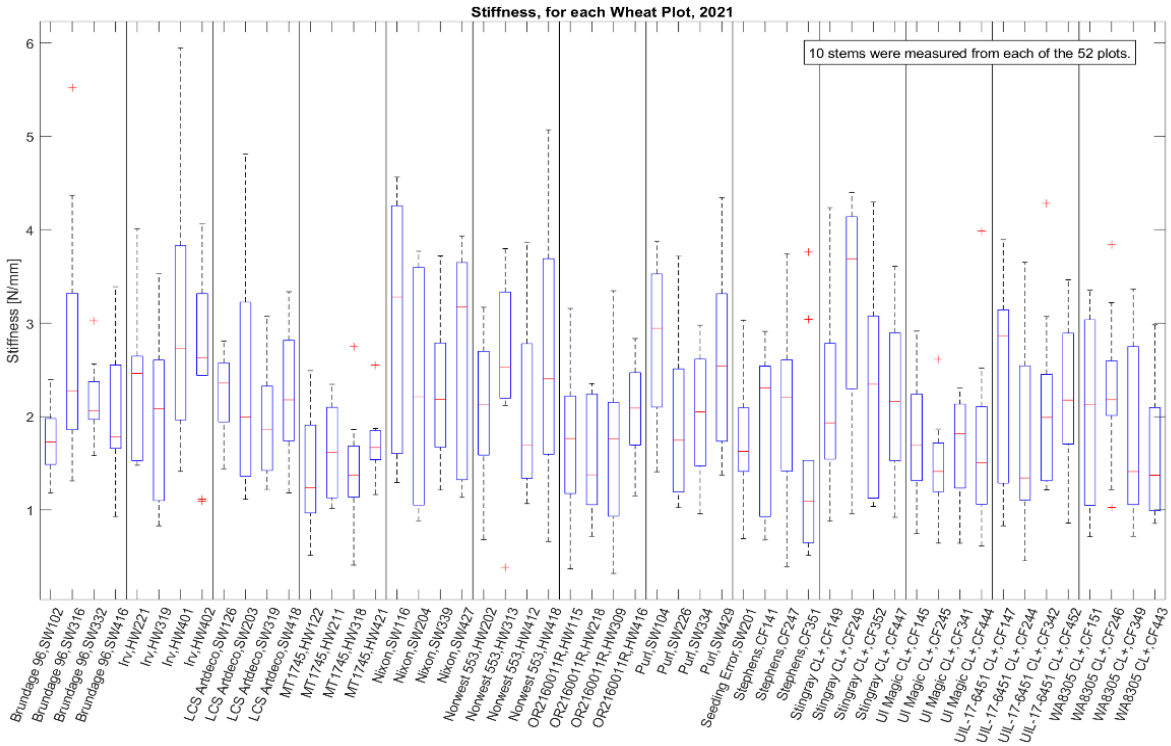

Figure 60: Box plots of three-point bending stiffness performance from each wheat plot tested in 2021. Data collected with a Instron universal testing machine and then was compiled and visualized using MATLAB.

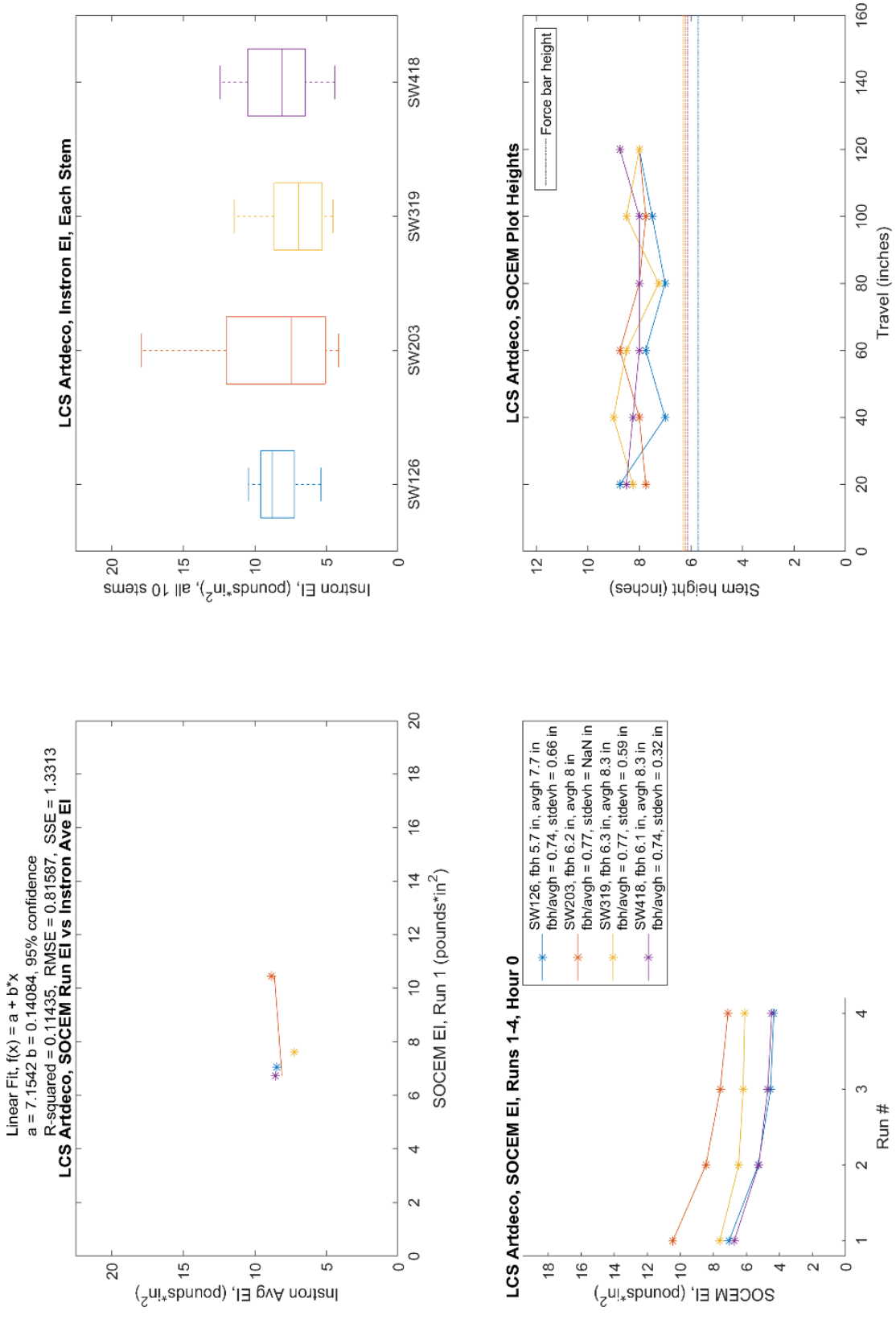

Figure 61: Complete data overview for four plots from the LCS Artdeco variety. Wheat 2021.

Figure 62: A cell of wheat gripped by the WheatSqueezer version 2, during 2022 testing. Because clamping the cell firmly requires twisting of two nuts on two bolts, the process is slow. Improvements can be made to make mass measurement time-effective.

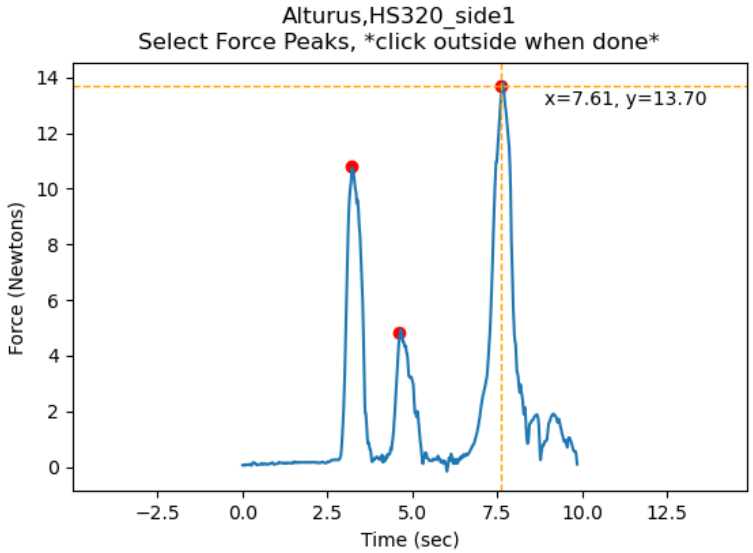

Figure 63: Image generated during peak selection with the PeakClick Python module, immediately following a SOCEM push in 2022.

Figure 64: SOCEM lined up for a side hit. Plants have been removed on each side.

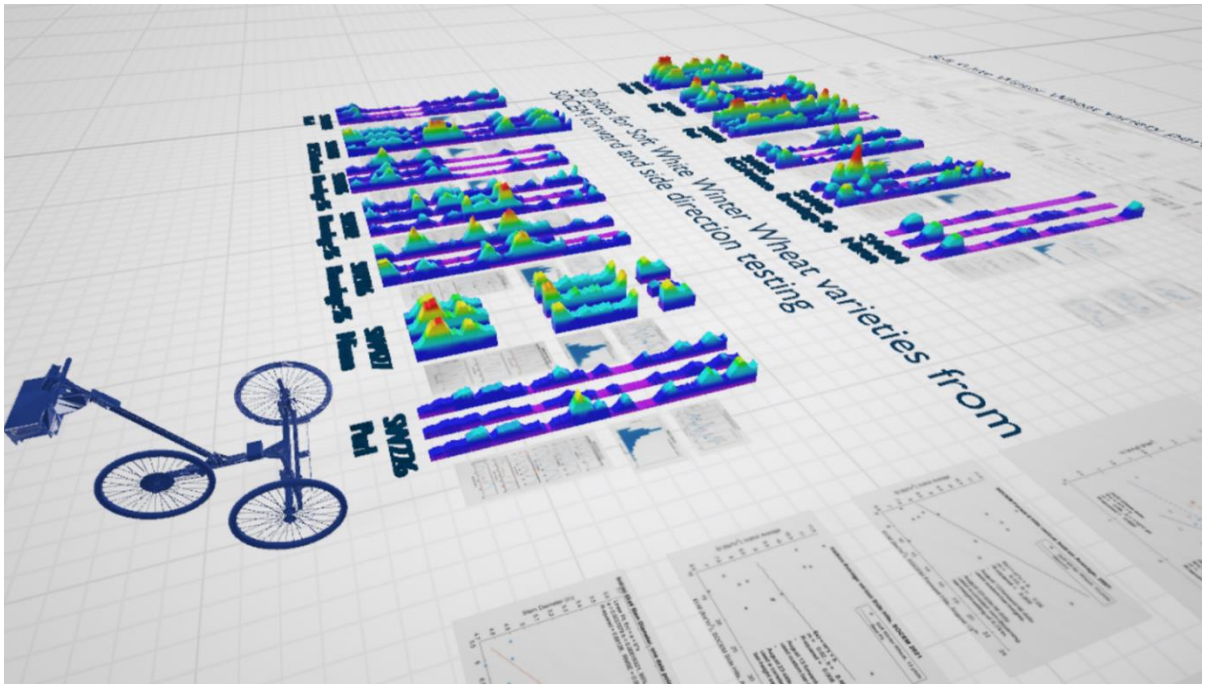

Figure 65: Raw force results from 2021 wheat data for Soft Winter varieties, output as an FBX file, shown in Microsoft 3D Viewer software. Soft Winter varieties in 2021 were subjected to both forward tests and multiple side tests, and the objects shown represent the multiplication and stitching of these force results.

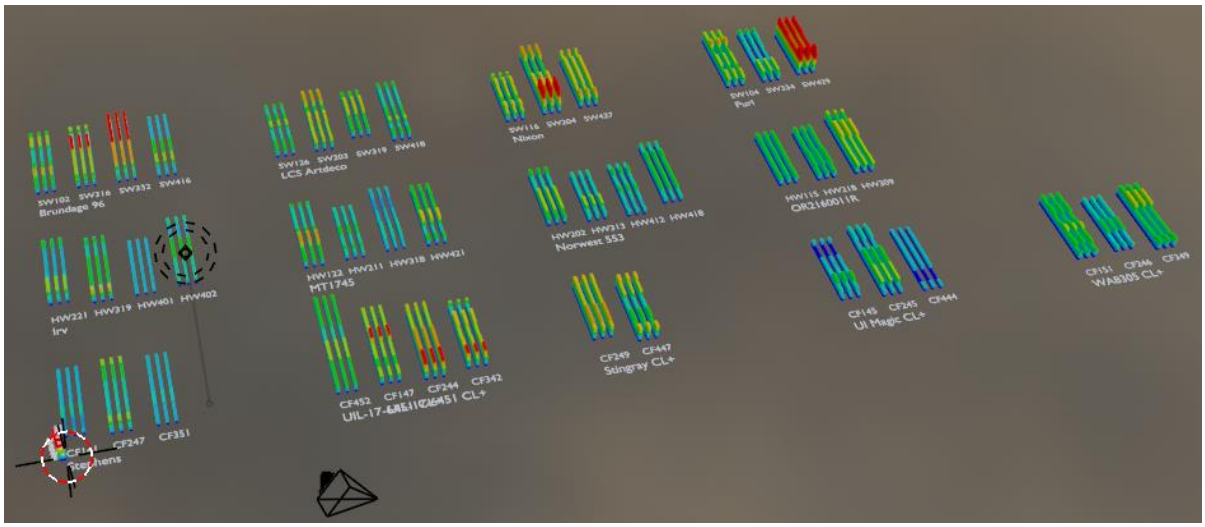

Figure 66: Flexural rigidity results from 2021 wheat data, shown in three-dimensions in the Blender software interface.

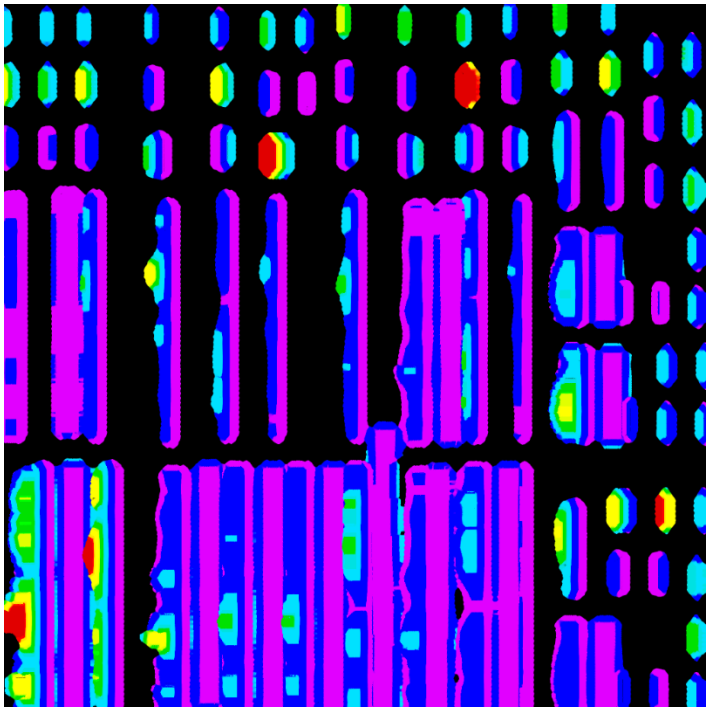

Figure 67: An example of a baked UV unwrap image. In Blender software, for 3D models to be exported with procedural color, it is required that the procedural material texture be unwrapped in the Cycles render engine and then recast in the Eevee render engine.

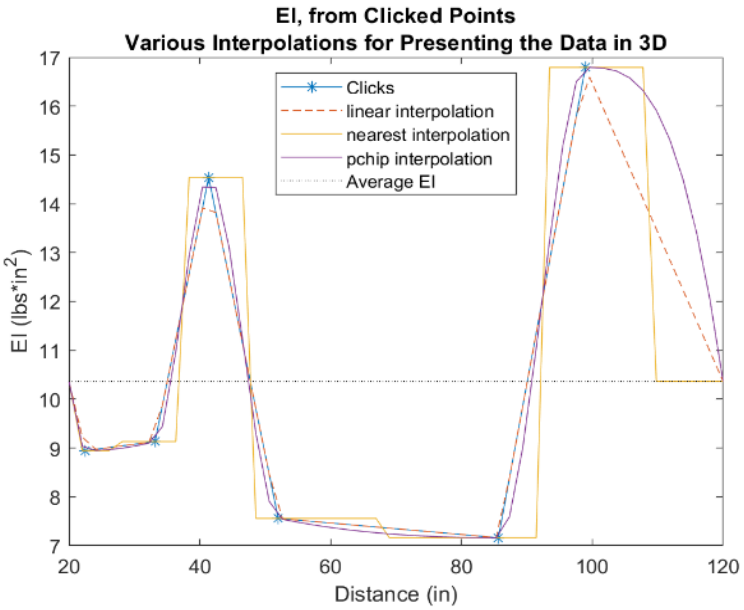

Figure 68: To achieve a 2D profile for the EI strength results from each SOCEM test, interpolation was used in MATLAB. Interpolation is necessary because EI is only assessed for discrete points from each test.

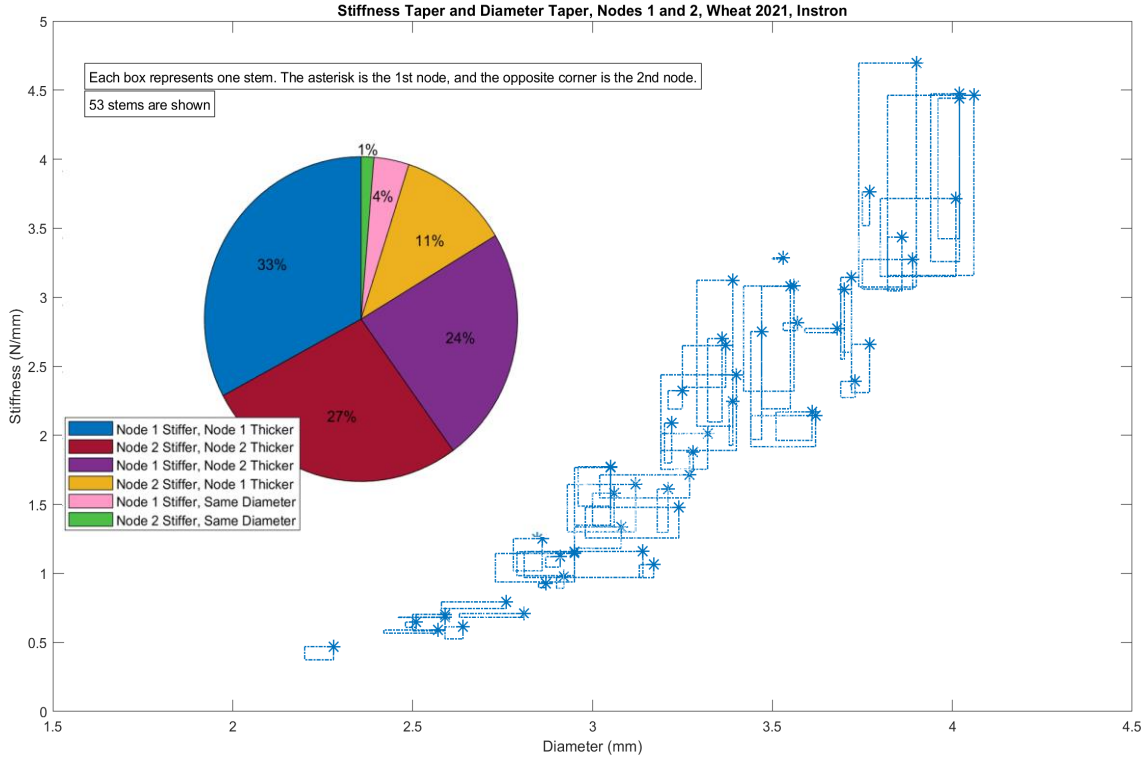

Figure 69: Node comparison results, showing stems with a stiffer Node 1 and a thicker Node 1.

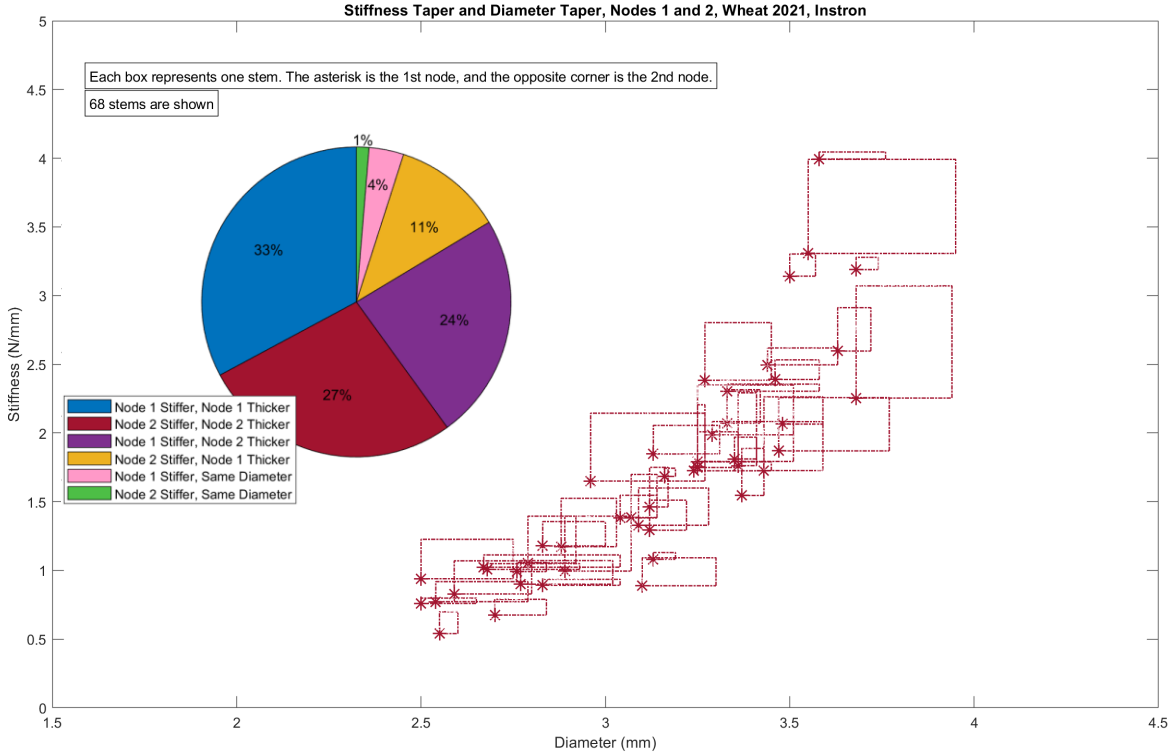

Figure 70: Node comparison results, showing stems with a stiffer Node 2 and a thicker Node 2.

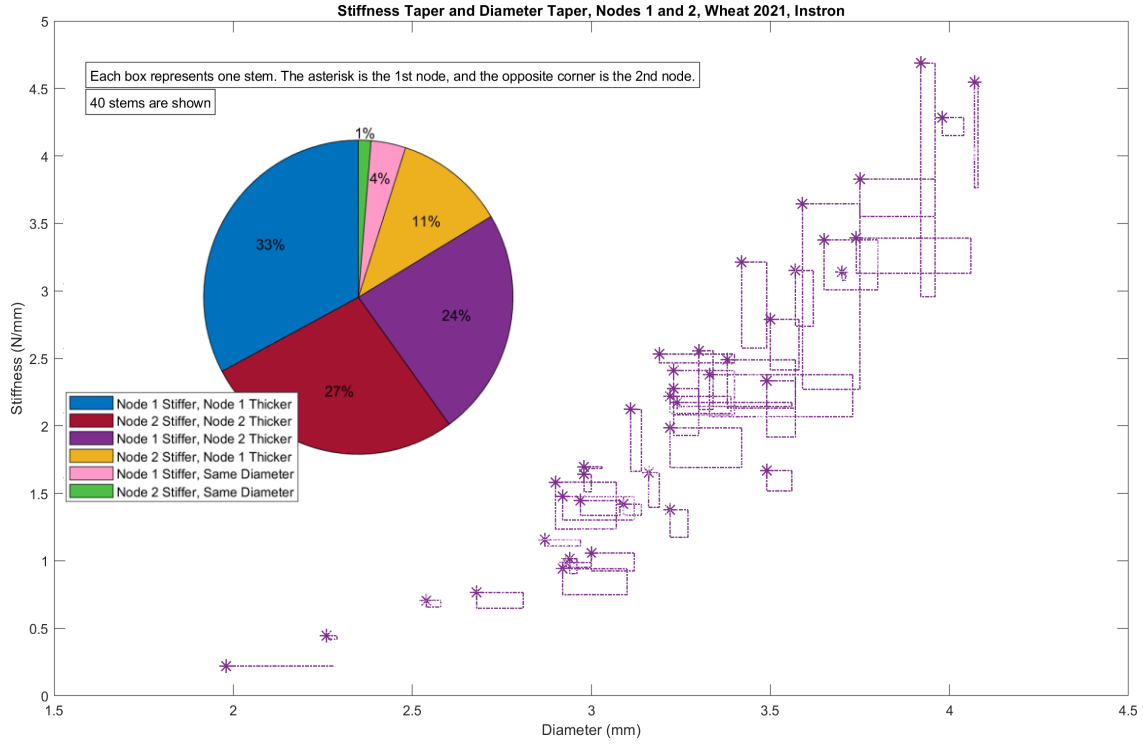

Figure 71: Node comparison results, showing stems with a stiffer Node 1 and a thicker Node 2.

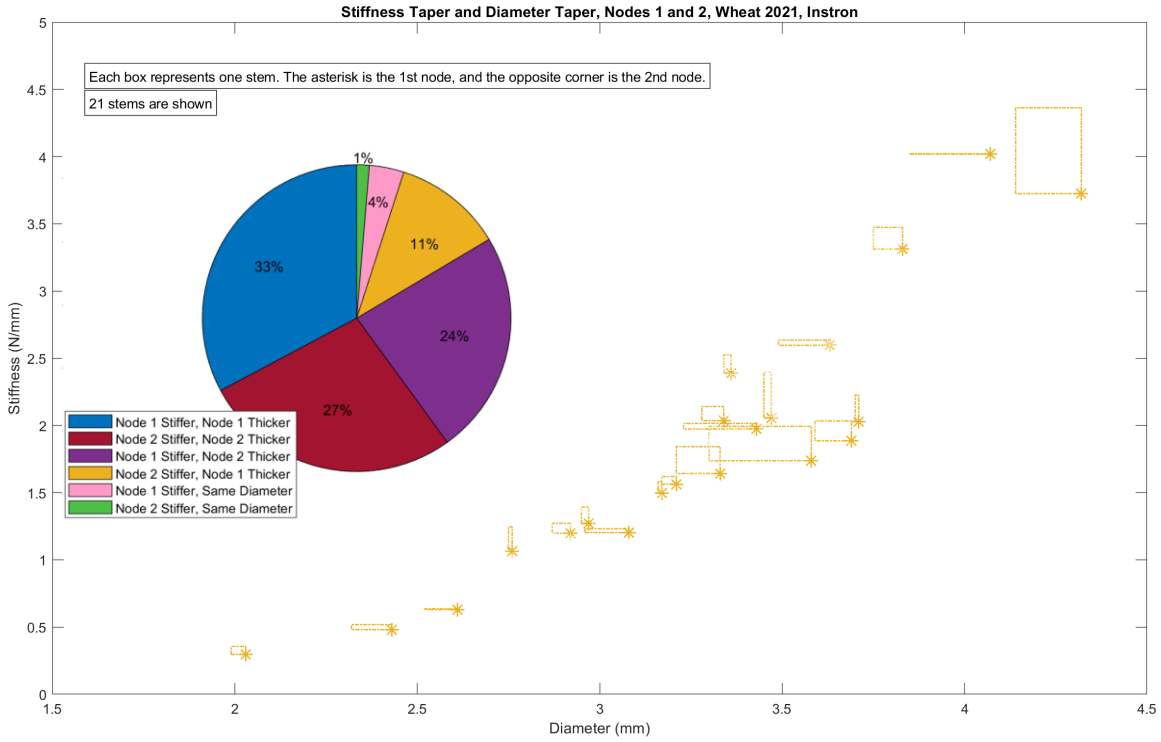

Figure 72: Node comparison results, showing stems with a stiffer Node 2 and a thicker Node 1.

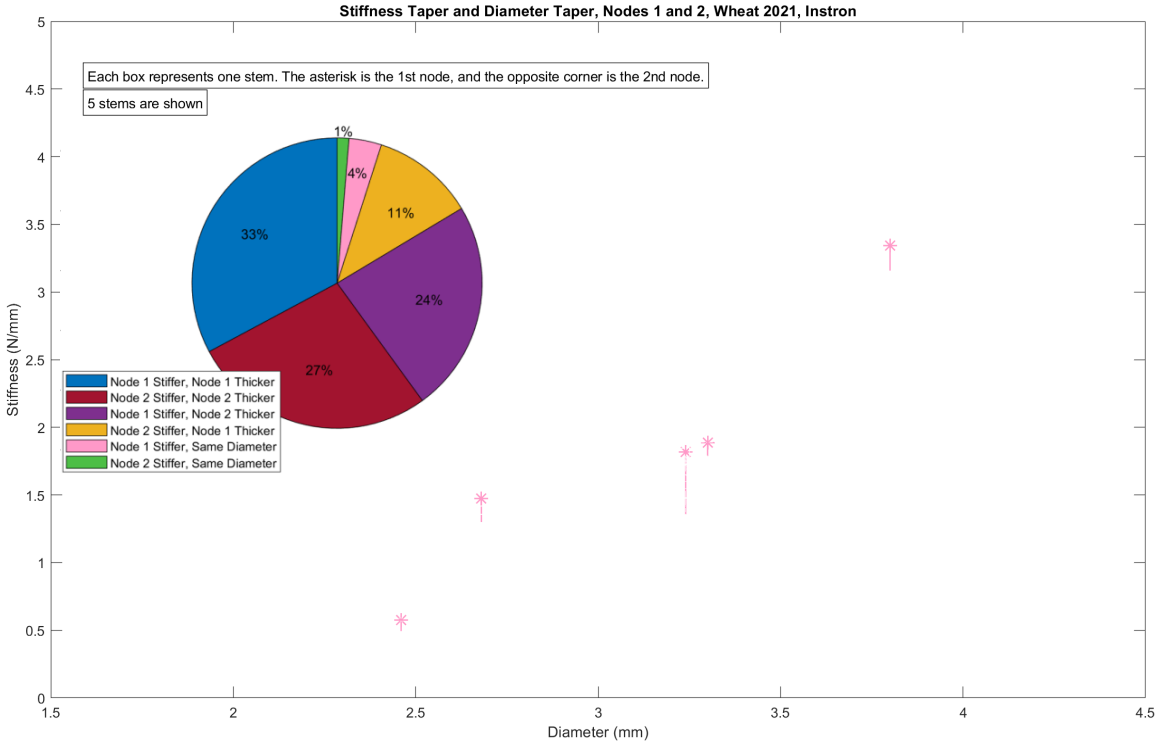

Figure 73: Node comparison results, showing stems with a stiffer Node 1 and equal diameter.

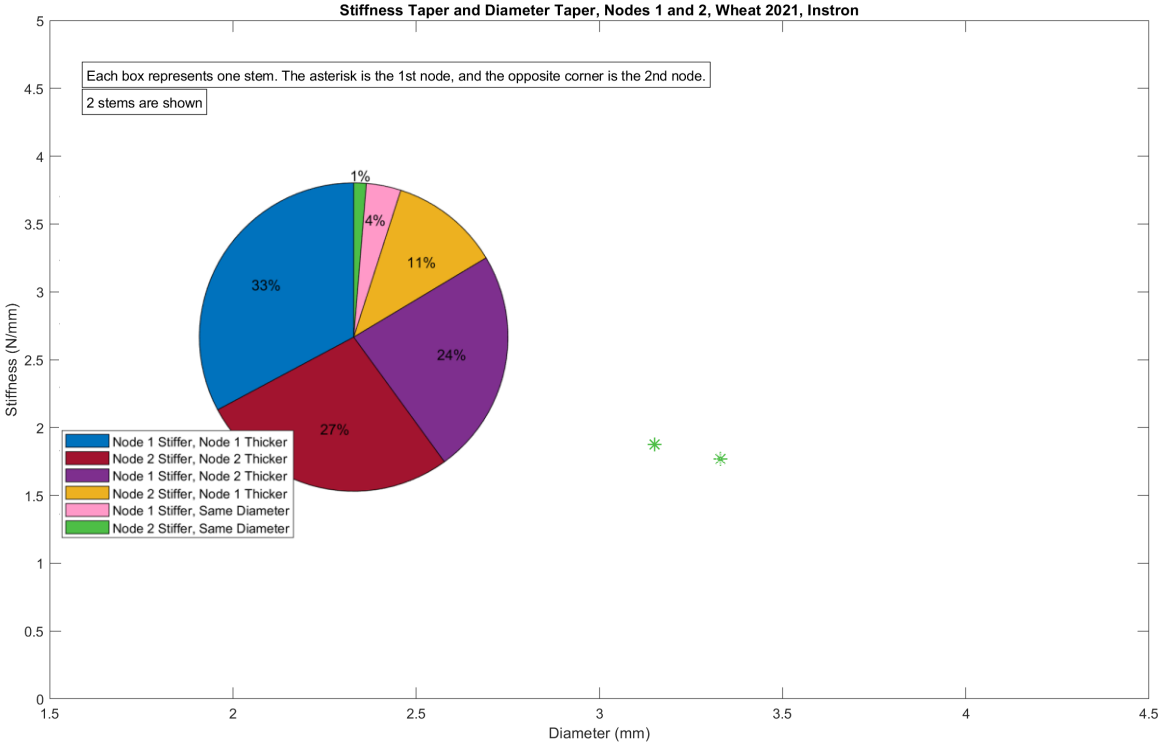

Figure 74: Node comparison results, showing stems with a stiffer Node 2 and equal diameter.

Appendix B: Software

Shared here are two scripts that have been central SOCEM improvement and the use of SOCEM data.

First is StemBerry_v97.py, which generates the Tkinter GUI interface for the SOCEM, collects data, and handles data.

Second is spreasheetsToTable_v3.m, which can be used to import several multipage spreadsheets into a single combined MATBLAB table. The column names in the generated table will reflect the column names from the spreadsheets. If the header organization is not identical for all spreadsheets in the target folder, additional columns will be generated in the table. There are applications for this script beyond SOCEM output files.

Appendix B1: StemBerry_v97.py

```
#!/usr/bin/Python3
#do not erase (needed to be executable for autostart)
'''
StemBerry V.97
Title: StemBerry_v97.py
Last updated: 9/08/2022
Dev: Clayton Bennett
OG dev: Austin Bebee
Description: SOCEM GUI. Connect RPi to Arduino, collect raw data. Save text inputs.

Contents (in order):
- Library imports
- Global Variables
- Global Functions
- GUI Class
   - Home / Initial input screen
   - Data collection (Record Force) screen
      - Runs data collection function
      - Stores data & saves data
      - Plots F v D graph
   - Load cell calibration screen
   - Error report screen
- Excute GUI command

V15
   - Change to 9 cell and 3 range count inputs
V19
   - Rip out defunct calculations
   - Clean up code, specifically by organizing statements of place for tkinter items
V37
   - Dial in functionality with pretty new GUI.
   - barbottom (not barmiddle) set to 70%-90% of stem height
V42
   - Develop top level methods
V50
   - Functional save state, save files, naming convention edge cases, and crisp appearance

V54
   - Generate CSV's, suppress XLSX's
V56
   - Retain 9-cell variables, for EI assessment upon saving counts, without reopening CSV files
V67
   - So many things.
v77
   - Serial collection functial, drinking from a waterhose, high hz
   - Tare button message.
```

- PeakClick popup window.

V84
- The way peak clicks are handled and saved was moved to the inside of the choose peaks code, becuase plt.show() won't give up.
- Shut down plt.show after CSV file is saved.

V88
- GUI.filename_force updated on page change to either record force frame or final inputs page
- nameBlackBox updated to remove excess hyphen when direction =='
- XLSX compilation file functional, currently set to seek force and EI files
- EI calcualtion works - only needs 1 file for all four nine-cell-scheme tests.
- This thing is getting heavy, 2844 lines.

V90
- Identify OS and choose filepath accordingly.

V92
- Trigger peak selection for all tests, with the assessAllTests boolean.
- Noticed that encoderWorked_override is poorly implemented. No reason to fix now, but, should be alterable as opposed to needing manual suppression through commenting
- GUI.currentdirection.get() set to "" on_frame_show RecordForce.

V94
- Changed mass measurement from kg to gramsa
- Fixed all time units to be (sec), not (s) or (seconds), and certainly not (ms)

V96
- EI is now calculated in lbs*in^2, then converted to metric N*cm^2. Input is metric, conversion happens inside, processing is SAE, then conversion to metric before output to metric.

V97
- EI calculation betaV edge cases dealt with: if nan, set betaV to 0.

Fix:
- Change compilation to access CSV data rather than state data. This is to protect against data loss if the computer dies.
- And, load state. Load state would be sick.
- Add more variables to state save backup text file.
- Remove auto graph button, or at least uncheck it: use it to refer to auto clicker
- Finish autoclicker by setting plt.show() into an inset tkinter gui popup, and then mainoop.
  Use: FigureCanvasTkAgg,NavigationToolbar2Tk,plt,Cursor.
- dev port is currently defined manually, given dev_manualOverride
- move header variable inputs
- make directory inputtable using dropdown menu item and textbox
- upgrade tkinter items to CustomTkinter
- PRIORITY: CREATE BASE NAME FROM VARIABLE AND PLOT: GUI.filename_force.get() is getting dangerous.

Notes:
- exec() is your friend. Use is to run multiple lines of code which you can copy and paste into a shell, using triple ' commenting
- save as separate CSV files, then as one combined XLSX file with multiple pages
'''

```
''' Libraries '''
import serial
from serial import Serial
# from serial import *
import serial.tools.list_ports # need this
import tkinter as tk
from tkinter import * # tk.Label == Label, tk.Button == Button, tk.Entry == Entry
import threading
from multiprocessing import Process
import csv

import matplotlib
from matplotlib import style
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
from matplotlib.figure import Figure
#import matplotlib.animation as animation
import matplotlib.pyplot as plt
from matplotlib.widgets import Cursor
import itertools
from itertools import zip_longest
import subprocess
import sys
import os
import platform
from os import path
import numpy as np
import pandas as pd
#import peakutils
#from PeakUtils.Plot import plot as pplot
import math
import struct
from PIL import ImageTk,Image
import datetime
from datetime import date
import time

''' Global Variables '''
operator = 'Clayton Bennett'
location = 'Kambitsch Farm'
coordinates = '46.592516,-116.946268'
script = os.path.basename(__file__)
directory = os.path.dirname(__file__)
operatingsystem = platform.system() #determine OS
# use or sys.plaform instead of platform.system, to avoid importing platform
print("operatingsystem =",operatingsystem)
print("os.getlogin() =",os.getlogin())
```

```
print("operator =",operator)
print("location =",location)
today = date.today()
datestring = today.strftime("%b-%d-%Y")
ignoreserial = False # True
#ignoreserial = True # delete this # if RecordForce.ser.isOpen() == False:
barlength = 76 # cm. this shouldn't ever change, unless the bar is replaced. i.e. the width of a side hit cell.
#dev_manual = 'COM7' # manual override
dev_manual = '/dev/ttyACM0' # manual override
dev_manualOverride = True
useInitialPlot_PeackClick = False
disReferenced_PeakClick = False
barradius = .8 # 1 cm = 0.32 inches
#barradius = 1 # 1 cm = 0.32 inches
default_stemheight = 10.0 # cm
initial_barbottomOverStemheight_coeff = 0.8
convert_KgToLbs = 2.20462262 #kg to lbs
convert_KgToN = 1/9.81 #kg to N # CHECK FOR ACCURACY CB 8/9/2022
convert_NToLbs = 4.44822
#calibrationFactor = 199750 # 23.4 N > 5 lbs; 5 lbs = 22.2411
calibrationFactor = 204200 # 22.24 N = 5 lbs

inchonvert = (((math.pi*(0.764))*31.4136)/359) # converts displacement to inches, wheel diameter = 31.4136
visualizeDatastream = False #True #set to live graph for data display
sleepSend = 0.5
encoderWorked_override = False # False means encoder will be trated as not working. this is poor code and should be improved.
assessAllTests = True
refreshAllAuto = False
#visualizeDatastream = True
# visualizeDatastream ( search: "def datafeed(" ) is broken right now. Refer to earlier versions (pre v65)for reference of how Bebee left it.

vis = 's' # legacy
vis = 'nope' #

if operatingsystem == 'Windows':
    if os.getlogin() == 'clayt':
        address = r'C:\Users\clayton\OneDrive - University of Idaho\AqMEQ\SOCEM\Data - Instron and SOCEM - 2020, 2021\SOCEM_DATA_2021'
        dev_guess = 'COM3' # manual override, windows 10 OS
    else:
        dev_manualOverride = False
        address = directory + '/SOCEM_data'
        if not os.path.exists(address):
            os.makedirs(address)
```

```
elif operatingsystem == 'Linux':
   dev_guess = '/dev/ttyACM0' # manual override raspian OS
   address = '/home/pi/Desktop/SOCEM_data_2022'
else:
   address = directory + '/SOCEM_data'
   dev_guess = dev_manual
   dev_manualOverride = False
   if not os.path.exists(address):
      os.makedirs(address)

''' matplotlib Graph Settings '''
'''
style.use("ggplot")
f = Figure(figsize=(4.85,3.9), dpi=75)
a = f.add_subplot(111)
a.set_ylim(0, 25)
'''

''' Methods'''
# Determine Arduino serial port address
def SerConnect():
   #try:
   ports = serial.tools.list_ports.comports()
   try:
      dev = ports[0].device
   except:
      #dev = '/dev/ttyACM0' # only works on pi
      dev = dev_guess # based on operating system
      if dev_manualOverride == True:
         dev = dev_manual # manual override
   try:
      ser = serial.Serial(dev, 115200, timeout=4,writeTimeout = 2,) # 1 second timeout
      #print(type(ser))
      print("dev = "+dev)
      ser.reset_input_buffer()
      #ser.isOpen()
      #GUI.ignoreserial = False
      return ser # this is the only spot it should be called ser, not RecordForce.ser

   except:
      GUI.ignoreserial = True
      error = 'serial connection never established'
      eCode = 'e1' # eCode = e1
      GUI.errors.append(error) # append error label
      GUI.errorCodes.append(eCode) # append error code
      #popup('serial connection')
      print("eCode = "+eCode)
```

```
# if serial disconnect (unplugged) reconnect - NOTE: doesn't properly work currently.
def SerReconnect():
   print("SerReconnect()")
   try:
   #if GUI.ignoreserial == False:
      GUI.ignoreserial = False
      try:
         RecordForce.ser.close()
         GUI.ignoreserial = False
      except:
         GUI.ignoreserial = True
      RecordForce.ser = SerConnect()

   except:
   #else:
      GUI.ignoreserial = True
      print("\nYou hit the 'SerReconnect' dropdown menu item while GUI.ignoreserial ==
True.\nSerial cannot be reconnected because\neither an arduino is not connected to your
computer\nor the arduino is not sought by StemBerry.")
      RecordForce.message_connectArduino()

def overwriteGuard(filename):# prevents overwriting by checking if filename already exists in saving
folder
      return path.exists(filename) # True = already exits, False = doesn't exist

def overwriteGuardPage(filename):# prevents overwriting by checking if filename already exists in
saving folder
      #return path.exists(filename) # True = already exits, False = doesn't exist
      return False # don't mess up!

def data_display(visual): #changes display method    #DELETE?
   global visualizeDatastream
   visualizeDatastream = visual
   return visualizeDatastream

#if any error occurs, display popup error msg
def popup(error):
   popup = tk.Tk()
   popup.wm_title("Error")
   E_label = Label(popup, text="A {} error occurred.".format(error), font=("arial", 12, "bold"))
   E_label.pack(side="top", fill="x", pady=10)
   popup.mainloop()

def popup_chooseFolder():
   popup_chooseFolder = tk.Tk()
   popup_chooseFolder.wm_title("Choose Folder")
   E_label = Label(popup_chooseFolder, text="Paste file output directory here.", font=("arial", 12,
"bold"))
```

```
    #E_label.pack(side="top", fill="x", pady=10)
    E_label.grid(row=0, column=1)
    #GUI.addressInput.set("")
    folder_entry = Entry(popup_chooseFolder, textvariable=GUI.addressInput, font = ("arial", 11,
"bold"), width= 70, bg="white", fg="gray1")
    folder_entry.grid(row=1, column=1)
    save_button = Button(popup_chooseFolder,text = "Save", font = ("arial", 14, "bold"), height = 1,
width = 6, fg = "ghost white", bg = "dodgerblue3",command=lambda:updateAdress())
    save_button.grid(row=2, column=1)
    popup_chooseFolder.mainloop()

    ''' Frame: Folder Input Field''
    barset_frame = tk.LabelFrame(self, text='Bar Bottom Quickset',font = ("arial", 14, "bold"), width=
10, bg="white", fg="gray1")
    barset_frame.place(x = 340, y = 230)
    ''' ''

def updateAdress():
    print("updateAddress is broken. Please develop.")
    print("GUI.addressInput.get() = ",GUI.addressInput.get())
    print("GUI.address = ",GUI.address)
    #GUI.address = GUI.addressInput.get() # broken right now
    #print("GUI.address = ",GUI.address)

def showErrors():
    GUI.show_frame(ErrorReport) # show Error Report page
    ErrorReport.showErrors2(GUI.frames[ErrorReport]) # display errors in lists

def update_filename_preTest():
    filename_preTest = nameBlackBox("preTest",GUI.filename_preTest.get())
    GUI.filename_preTest.set(filename_preTest)
    filename_all = filename_preTest.replace("preTest","all")
    GUI.filename_all.set(filename_all)

def testForNineCellFilename(): # used to identify when nine-cell force, distance, and time data
exists, and passes it to state data.
    # the purpose of this is to avoid reopening CSV files in order to assess nine-cell data
    # because, we have to wait for counts after to assess EI
    # it would be easier to test right away to get peaks
    # have a check box for nine cell test
    # EI cannot be assessed for non-nine cell, because counts don't exist
    # if box not checked, post test frame goes to single input for stem count, one number, with
another number for range distance of count
    # # Assessment is trigged at save state button push
    #ninecellfilename = GUI.varietyname.get()+","+GUI.plotname.get()+"_"
    ninecellfilename = GUI.varietyname.get()+","+GUI.plotname.get()
    ninecellfilename_side1 = ninecellfilename+"_side1"
    ninecellfilename_side2 = ninecellfilename+"_side2"
```

```
    ninecellfilename_side3 = ninecellfilename+"_side3"
    ninecellfilename_forward = ninecellfilename+"_foward"
    currentFilename_force = GUI.filename_force.get()
    # create GUI variable, for handling without reopening CSV's
    #if (currentFilename_force == ninecellfilename_side1):
    if (GUI.currentdirection.get() == "side1"):
        GUI.forcePushed_side1 = GUI.forcePushed
        GUI.distanceTraveled_side1 = GUI.distanceTraveled
        GUI.timeElapsed_side1 = GUI.timeElapsed
        #if (currentFilename_force == ninecellfilename_side2):
    if (GUI.currentdirection.get() == "side2"):
        GUI.forcePushed_side2 = GUI.forcePushed
        GUI.distanceTraveled_side2 = GUI.distanceTraveled
        GUI.timeElapsed_side2 = GUI.timeElapsed
        #if (currentFilename_force == ninecellfilename_side3):
    if (GUI.currentdirection.get() == "side3"):
        GUI.forcePushed_side3 = GUI.forcePushed
        GUI.distanceTraveled_side3 = GUI.distanceTraveled
        GUI.timeElapsed_side3 = GUI.timeElapsed
        #if (currentFilename_force == ninecellfilename_forward):
    if (GUI.currentdirection.get() == "forward"):
        GUI.forcePushed_forward = GUI.forcePushed
        GUI.distanceTraveled_forward = GUI.distanceTraveled
        GUI.timeElapsed_forward = GUI.timeElapsed

def createBackupFile():
    ''' Create a temp text file, with a list of all variables and variable names, that would be awesome '''
    '''update_filename_preTest()
    update_filename_postTest()
    sniff_filename_force()
    update_filename_postTest()
    saveState_update_filenames()'''
    now = datetime.datetime.now()
    unix_now = time.mktime(now.timetuple())
    unix_now_int = int(unix_now) # still gets seconds # the purpose of this is to append to filenames
    str(unix_now_int)
    filename_savestate = "backup_stemberry_"+str(unix_now_int)+".txt"
    filename_savestate_full = GUI.address+"/"+filename_savestate
    print("State saved at "+str(datetime.datetime.fromtimestamp(unix_now_int))+":
"+filename_savestate)
    # list all GUI vars, add them to a txt file

GUI.masslist=[GUI.cell1Mass.get(),GUI.cell2Mass.get(),GUI.cell3Mass.get(),GUI.cell4Mass.get(),GUI.c
ell5Mass.get(),GUI.cell6Mass.get(),GUI.cell7Mass.get(),GUI.cell8Mass.get(),GUI.cell9Mass.get()]

GUI.stemcounts=[GUI.cell1Count.get(),GUI.cell2Count.get(),GUI.cell3Count.get(),GUI.cell4Count.get(
),GUI.cell5Count.get(),GUI.cell6Count.get(),GUI.cell7Count.get(),GUI.cell8Count.get(),GUI.cell9Count
.get()]
```

```
   GUI.diameters_cell1 =
[GUI.cell1Diameter1.get(),GUI.cell1Diameter2.get(),GUI.cell1Diameter3.get(),GUI.cell1Diameter4.ge
t()]
   GUI.diameters_cell2 =
[GUI.cell2Diameter1.get(),GUI.cell2Diameter2.get(),GUI.cell2Diameter3.get(),GUI.cell2Diameter4.ge
t()]
   GUI.diameters_cell3 =
[GUI.cell3Diameter1.get(),GUI.cell3Diameter2.get(),GUI.cell3Diameter3.get(),GUI.cell3Diameter4.ge
t()]
   GUI.diameters_cell4 =
[GUI.cell4Diameter1.get(),GUI.cell4Diameter2.get(),GUI.cell4Diameter3.get(),GUI.cell4Diameter4.ge
t()]
   GUI.diameters_cell5 =
[GUI.cell5Diameter1.get(),GUI.cell5Diameter2.get(),GUI.cell5Diameter3.get(),GUI.cell5Diameter4.ge
t()]
   GUI.diameters_cell6 =
[GUI.cell6Diameter1.get(),GUI.cell6Diameter2.get(),GUI.cell6Diameter3.get(),GUI.cell6Diameter4.ge
t()]
   GUI.diameters_cell7 =
[GUI.cell7Diameter1.get(),GUI.cell7Diameter2.get(),GUI.cell7Diameter3.get(),GUI.cell7Diameter4.ge
t()]
   GUI.diameters_cell8 =
[GUI.cell8Diameter1.get(),GUI.cell8Diameter2.get(),GUI.cell8Diameter3.get(),GUI.cell8Diameter4.ge
t()]
   GUI.diameters_cell9 =
[GUI.cell9Diameter1.get(),GUI.cell9Diameter2.get(),GUI.cell9Diameter3.get(),GUI.cell9Diameter4.ge
t()]

   lines = [
      'Units: diameter (mm), height (cm), range (cm), length (cm), mass (g), time (sec), force (N) \n',
      'script = '+script,
      'directory = '+directory+'/',
      'operatingsystem = '+operatingsystem,
      'os.getlogin() = '+os.getlogin(),
      'operator = '+operator,
      'location = '+location,
      'coordinates = '+coordinates,
      'GUI.ignoreserial = '+str(GUI.ignoreserial),
      'default_stemheight = '+str(default_stemheight),
      'calibrationFactor = '+str(calibrationFactor),
      'encoderWorked_override = '+str(encoderWorked_override),
      'assessAllTests = '+str(assessAllTests),
      'barlength = '+str(barlength),
      'datestring = '+datestring,
      'today = '+str(today),
      'now = '+str(now),
      'unix_now '+str(unix_now),
      'unix_now_int = '+str(unix_now_int),
```

```
        'backup filename unix_now_int decoded: '+
str(datetime.datetime.fromtimestamp(unix_now_int))+'\n',
        'GUI.timestring.get() = '+GUI.timestring.get(),
        'GUI.errors = '+makeDataString(GUI.errors),
        'GUI.errorCodes = '+makeDataString(GUI.errorCodes),
        'GUI.varietyname.get() = '+GUI.varietyname.get(),
        'GUI.plotname.get() = '+GUI.plotname.get(),
        'GUI.currentdirection.get() = '+GUI.currentdirection.get(),
        'GUI.filename_force.get() = '+GUI.filename_force.get(),
        'GUI.filename_preTest.get() = '+GUI.filename_preTest.get(),
        'GUI.filename_postTest.get() = '+GUI.filename_postTest.get(),
        'GUI.stemheight.get() = '+str(GUI.stemheight.get()),
        'GUI.barmiddle.get() = '+str(GUI.barmiddle.get()),
        'GUI.barbottom.get() = '+str(GUI.barbottom.get()),
        'GUI.passfillednames_checkbox.get() = '+str(GUI.passfillednames_checkbox.get()),
        'GUI.startRange1.get() = '+str(GUI.startRange1.get()),
        'GUI.startRange2.get() = '+str(GUI.startRange2.get()),
        'GUI.startRange3.get() = '+str(GUI.startRange3.get()),
        'GUI.travelvelocity = '+str(GUI.travelvelocity),
        'GUI.samplingrate = '+str(GUI.samplingrate),
        'GUI.masslist = '+makeDataString(GUI.masslist),
        'GUI.stemcounts = '+makeDataString(GUI.stemcounts),
        'GUI.diameters_cell1 = '+makeDataString(GUI.diameters_cell1),
        'GUI.diameters_cell2 = '+makeDataString(GUI.diameters_cell2),
        'GUI.diameters_cell3 = '+makeDataString(GUI.diameters_cell3),
        'GUI.diameters_cell4 = '+makeDataString(GUI.diameters_cell4),
        'GUI.diameters_cell5 = '+makeDataString(GUI.diameters_cell5),
        'GUI.diameters_cell6 = '+makeDataString(GUI.diameters_cell6),
        'GUI.diameters_cell7 = '+makeDataString(GUI.diameters_cell7),
        'GUI.diameters_cell8 = '+makeDataString(GUI.diameters_cell8),
        'GUI.diameters_cell9 = '+makeDataString(GUI.diameters_cell9),
        'GUI.EI_fullcontact = '+makeDataString(GUI.EI_fullcontact),
        'GUI.EI_intermediatecontact = '+makeDataString(GUI.EI_intermediatecontact),
        'GUI.EI_nocontact = '+makeDataString(GUI.EI_nocontact),
        'GUI.AvgEI_intermediatecontact = '+makeDataString(GUI.AvgEI_intermediatecontact),
        str(datetime.datetime.now())+'\n']

    evenmorelines = [
        'GUI.filename_all.get() = '+GUI.filename_all.get(), # no longer exists, compilation XLSX
        'GUI.distanceTraveled = '+makeDataString(GUI.distanceTraveled),
        'GUI.forcePushed = '+makeDataString(GUI.forcePushed),
        'GUI.timeElapsed = '+makeDataString(GUI.timeElapsed)+'\n',
        'Collected data, nine cell scheme:',
        'GUI.forcePushed_side1 = '+makeDataString(GUI.forcePushed_side1),
        'GUI.distanceTraveled_side1 = '+makeDataString(GUI.distanceTraveled_side1),
        'GUI.timeElapsed_side1 = '+makeDataString(GUI.timeElapsed_side1),
        'GUI.forcePushed_side2 = '+makeDataString(GUI.forcePushed_side2),
        'GUI.distanceTraveled_side2 = '+makeDataString(GUI.distanceTraveled_side2),
```

```
      'GUI.timeElapsed_side2 = '+makeDataString(GUI.timeElapsed_side2),
      'GUI.forcePushed_side3 = '+makeDataString(GUI.forcePushed_side3),
      'GUI.distanceTraveled_side3 = '+makeDataString(GUI.distanceTraveled_side3),
      'GUI.timeElapsed_side3 = '+makeDataString(GUI.timeElapsed_side3),
      'GUI.forcePushed_forward = '+makeDataString(GUI.forcePushed_forward),
      'GUI.distanceTraveled_forward = '+makeDataString(GUI.distanceTraveled_forward),
      'GUI.timeElapsed_forward = '+makeDataString(GUI.timeElapsed_forward),
      str(datetime.datetime.now())]

   try:
      morelines = [
         '\n',
         'RecordForce.ser = '+str(RecordForce.ser),
         str(datetime.datetime.now())]
   except:
      morelines = [
         '\n',
         'RecordForce.ser = '+'error',
         str(datetime.datetime.now())]

   with open(filename_savestate_full, 'w') as f:
      f.write('\n'.join(lines))
      f.write('\n'.join(morelines))
      try:
         f.write('\n'.join(evenmorelines))
      except:
         pass

def makeDataString(dataVector):
   #timeElapsed_string = ' '.join(str(e) for e in GUI.timeElapsed)
   dataString = ' '.join(str(e) for e in dataVector)
   return dataString

def restoreState():
   print("Please develop.")
   # choose txt file (example: backup_stemberry_1660192559.txt
   # trigger GUI directory and file selection would be sick.
   # only restore postTest fields? start there.

def rename(filename): #if filename already exists - prompt user to rename
   popup = tk.Tk()
   popup.wm_title('Prompt Rename')
   renameIt = Label(popup, text = 'Filename\n"{}"\nalready exists in the saving location.\nPlease
rename and press Save.'.format(filename), font = ('arial', 10, 'bold'))
   increment_button = Button(popup,text = "Auto Modify", font = ("arial", 14, "bold"), height = 2,
width = 6, fg = "ghost white", bg = "dodgerblue3",command=lambda:incrementRename(filename))
   overwrite_button = Button(popup, text = "Overwrite", font = ("arial", 14, "bold"), height = 2, width
= 6, fg = "ghost white", bg = "red4",command=lambda:overwrite(filename))
```

```
   renameIt.pack(side='top', fill='x', ipadx=10, ipady=10)
   increment_button.pack(side='top', fill='both', ipadx=10, ipady=1)
   overwrite_button.pack(side='top', fill='both', ipadx=10,ipady=1)

   popup.mainloop()
def renamePage(filename):
   print("Please develop, prevent pages from being overwritten in the filename_all spreadsheet")

def incrementRename(filename):
   print("please develop, auto modify filename")

def overwrite(filename):
   print("please develop, overwrite filename")

#closes GUI (from file menubar)
def close():
   createBackupFile()
   Python = sys.executable
   os.execl(Python, Python, * sys.argv)

def datafeed():
   #frame = tk.Frame.RecordForce
   frame = RecordForce.container
   RecordForce.datafeed_frame
   print("frame = ",frame)
   if visualizeDatastream == True:# data displayed in scrollbars (default)
      # Displays incoming data
      # scroll = Scrollbar(RecordForce.datafeed_frame)
      scroll = Scrollbar(frame)# what is this? TK!
      print("scroll = ",scroll)
      #scroll = Scrollbar(self)# what is this? TK!
      ''
      RecordForce.time_label = Label(RecordForce.datafeed_frame, text = "Time (sec)",font = ("arial",
14, "bold"), fg = "dodgerblue3", bg = "ghost white")
      RecordForce.Timelist = Listbox(RecordForce.datafeed_frame, yscrollcommand = scroll.set, bg =
"ghost white",highlightbackground = "gray2", width = 7, height = 1, font = ("arial", 14, "bold"), fg =
"dodgerblue3")
      RecordForce.dis_label = Label(RecordForce.datafeed_frame, text = "Distance (cm)",font =
("arial", 14, "bold"), fg = "dodgerblue3", bg = "ghost white")
      RecordForce.Dislist = Listbox(RecordForce.datafeed_frame, yscrollcommand = scroll.set, bg =
"ghost white",highlightbackground = "gray2", width = 7, height = 1, font = ("arial", 14, "bold"), fg =
"dodgerblue3")
      RecordForce.force_label = Label(RecordForce.datafeed_frame, text = "Force (N)",font = ("arial",
14, "bold"), fg = "dodgerblue3", bg = "ghost white")
```

```
      RecordForce.Forcelist = Listbox(RecordForce.datafeed_frame, yscrollcommand = scroll.set, bg =
"ghost white",highlightbackground = "gray2", width = 7, height = 5, font = ("arial", 14, "bold"), fg =
"dodgerblue3")
      '''
      RecordForce.time_label = Label(frame, text = "Time (sec)",font = ("arial", 14, "bold"), fg =
"dodgerblue3", bg = "ghost white")
      RecordForce.Timelist = Listbox(frame, yscrollcommand = scroll.set, bg = "ghost
white",highlightbackground = "gray2", width = 7, height = 1, font = ("arial", 14, "bold"), fg =
"dodgerblue3")
      RecordForce.dis_label = Label(frame, text = "Distance (cm)",font = ("arial", 14, "bold"), fg =
"dodgerblue3", bg = "ghost white")
      RecordForce.Dislist = Listbox(frame, yscrollcommand = scroll.set, bg = "ghost
white",highlightbackground = "gray2", width = 7, height = 1, font = ("arial", 14, "bold"), fg =
"dodgerblue3")
      RecordForce.force_label = Label(frame, text = "Force (N)",font = ("arial", 14, "bold"), fg =
"dodgerblue3", bg = "ghost white")
      RecordForce.Forcelist = Listbox(frame, yscrollcommand = scroll.set, bg = "ghost
white",highlightbackground = "gray2", width = 7, height = 5, font = ("arial", 14, "bold"), fg =
"dodgerblue3")
      '''
      RecordForce.time_label.place(x = 180, y = 110)

      RecordForce.Timelist.place(x = 180, y = 140)
      RecordForce.dis_label.place(x = 280, y = 110)
      RecordForce.Dislist.place(x = 280, y = 140)
      RecordForce.force_label.place(x = 420, y = 110)
      RecordForce.Forcelist.place(x = 420, y = 140)

   else:# user decided for no data display
      try:#clear scrollbars if they were there
         RecordForce.Dislist.place_forget()
         RecordForce.Forcelist.place_forget()
         RecordForce.Timelist.place_forget()
         RecordForce.dis_label.place_forget()
         RecordForce.force_label.place_forget()
         RecordForce.time_label.place_forget()
      except:# no scrollbars
         pass

def passData():

   '''Scrollbars Options'''
   # if scrollbars option = on:
   if visualizeDatastream == True:
      try: # puts data on GUI display by default (user can turn off)

         RecordForce.Dislist.insert(END, str(GUI.distanceTraveled[i]))# inserts at end of listbox to
actually display
```

```
            RecordForce.Dislist.see(END)# makes sure listbox is at end so it displays live data
            RecordForce.Forcelist.insert(END, str('%.2f' % GUI.forcePushed[i]))
            RecordForce.Forcelist.see(END)
            RecordForce.Timelist.insert(END, str('%.2f' % GUI.timeElapsed[i]))
            RecordForce.Timelist.see(END)

            #scrollbars options = off
            '''
        except:
            pass
        '''
        except:
            GUI.errors.append('data append') # label
            eCode = 'e4'
            GUI.errorCodes.append(eCode)
            print("eCode = "+eCode) # eCode = e4


# * # DATA COLLECTION FUNCTION - Acquires live data from Arduino # * #
def collectData():
    hang=0
    j=0
    nothingToRead=0 # controls timeout
    blankline = "b'\n"
    lasttimetick = -1
    while RecordForce.hasStarted==True and RecordForce.hasSentStop==False:
        time.sleep(0.02)
        bytecount = RecordForce.ser.in_waiting
        #print("RecordForce.ser.in_waiting = ",bytecount)
        if bytecount > 5 and RecordForce.hasSentStop==False: # this does happen
            #print("datachunk...") # stopping after this

            try:
                time.sleep(0.2) # no luck
                ser_bytes = RecordForce.ser.read(bytecount)
                if blankline in str(ser_bytes):
                    print("blankline")
                    continue
            except:
                print("Failed: ser_bytes = RecordForce.ser.readline()")
                continue
            hang = 0
            nothingToRead=0
            #print("ser_bytes = ",ser_bytes)
            line = ser_bytes.decode('utf-8').rstrip()
            datapacket = line.splitlines()
            # parse datapacket
            for i in datapacket:
```

```
            split = i.split("|")
            if RecordForce.hasSentStop == False:
               try:
                  #print("split = ",split, float(split[0]),float(split[1]),float(split[2]))
                  if round(j/10,0) == float(j/10):
                     print("j, split = ",j, ",",split)
                  distance = round(float(split[0]),3)
                  force = round(float(split[1]),3)
                  timetick = round(float(split[2]),3)/1000 # convert milliseconds to seconds
                  if timetick > lasttimetick:
                     GUI.timeElapsed.append(timetick)# list of GUI.distanceTraveled time
                  else:
                     timetick = lasttimetick # good enough.
                     GUI.timeElapsed.append(timetick)# list of GUI.distanceTraveled time
                  GUI.distanceTraveled.append(distance)# list of inches traveled @ does this happen
with the whole list, or one element at a time?
                  GUI.forcePushed.append(force)# list of force traveled
                  lasttimetick = timetick
               except:
                  print("missed a line, list index out of range.")
                  pass

            j+=1
         if line =="Stopped!":
            RecordForce.sendStop()

      # the purpose of this elif is to allow the while loop to iterate if there's nothing to read.
      # But also, it has primarily been entered if the serial connection has already timed out
      elif bytecount < 6 and bytecount > 0 :
         ser_bytes = RecordForce.ser.read(bytecount)
         #print("ser_bytes = ",ser_bytes)
         nothingToRead +=1
         if nothingToRead>5: # if the while loop goes through five iterations, without seeing anything
worth recording, give up.
            RecordForce.sendStop()
            print("Hung up.")
            SerReconnect()
            GUI.show_frame(InitialInputs)
      else:
         hang +=1
         print("go back to top of while loop")
         if hang>10: # if the while loop goes through ten iterations of radio silence, give up. The serial
connection probably timed out. search 'timeout = '
            RecordForce.sendStop()
            print("Hung up, timeout.")
            SerReconnect()
            GUI.show_frame(InitialInputs)
```

```
def runDataCollect():
   try:
      RecordForce.sendStart()
   except:
      print("run fail")
      GUI.errors.append('serial com. (start data)') # label
      eCode = 'e2' # eCode = e2
      GUI.errorCodes.append(eCode)
      print("eCode = "+eCode)
      popup('start data collect')

   RecordForce.thread2_collectData = threading.Thread(target = collectData)
   RecordForce.thread2_collectData.start()

def incrementName(filename):
      hyphen = "_"
      # determine last few characters from a filename
      def incrementvars(filename):
         lastchar = filename[len(filename)-1]
         secondtolastchar = filename[len(filename)-2]
         thirdtolastchar = filename[len(filename)-3]
         lastcharandsecondtolastchar = str(secondtolastchar+lastchar)
         return lastchar, secondtolastchar, thirdtolastchar, lastcharandsecondtolastchar

      #check if the last two are hyphens. if there is more than one hypthen, remove the last character
until there is only one hyphen.
      def hyphencheck(filename,hyphen,lastchar, secondtolastchar, thirdtolastchar,
lastcharandsecondtolastchar):
         while lastchar == hyphen and secondtolastchar == hyphen: # if two hyphens at the end
            filename = filename[:-1] # remove last character
            incrementvars()
         return filename, lastchar, secondtolastchar, thirdtolastchar, lastcharandsecondtolastchar

      if filename == "": # default, if user tried to increment without inputting any varietyname,
plotname, or filename
         filename = datestring+","+GUI.timestring.get()

      lastchar, secondtolastchar, thirdtolastchar, lastcharandsecondtolastchar =
incrementvars(filename)
      filename, lastchar, secondtolastchar, thirdtolastchar, lastcharandsecondtolastchar =
hyphencheck(filename,hyphen,lastchar, secondtolastchar, thirdtolastchar,
lastcharandsecondtolastchar)

      if lastchar == hyphen: # if last character is a hyphen
         newName = str(filename+str("1"))
      elif secondtolastchar == hyphen and lastchar.isnumeric: # if single digit preceded by a hyphen
         #newName = str(filename+str(int(lastchar)+1))
         filename = filename[:-1] # remove last character
```

```
            newName = str(filename+str(int(lastchar)+1))
        elif thirdtolastchar == hyphen and lastcharandsecondtolastchar.isnumeric: # if double digits
preceded by a hyphen
            filename = filename[:-1] # remove last character
            filename = filename[:-1] # remove last character
            newName = str(filename+str(int(lastcharandsecondtolastchar)+1))
        elif filename == "":
            newName = date
        else:
            newName = str(filename+"_1")
        return newName
        #GUI.filename_force.set(newName)

''' Edge cases: Filenaming '''
def nameDirectionScrub(filename):
    if ("_side1" in filename):
        filename=filename.replace("_side1","")
        print(filename)
    if ("_side2" in filename):
        filename=filename.replace("_side2","")
    if ("_side3" in filename):
        filename=filename.replace("_side3","")
    if ("_forward" in filename):
        filename=filename.replace("_forward","")
    if ("_postTest" in filename):
        filename=filename.replace("_postTest","")
    return filename

def nameMissing(varietyname,plotname):
    if varietyname == "":
        varietyname = datestring
    if plotname == "":
        plotname = GUI.timestring.get() # plotname = GUI.timestring.get() # if you want the timestring
(serving at plotname) to not change...but then it will never change
    return varietyname, plotname

def nameBlackBox(direction,filename):
    varietyname = GUI.varietyname.get()
    plotname = GUI.plotname.get()
    check=GUI.passfilllednames_checkbox.get()
    if GUI.filename_force.get()=="" and check==1 and direction=='':
        varietyname, plotname = nameMissing(varietyname, plotname)
        #print(varietyname, plotname)
        filename = str(varietyname+str(",")+plotname)
    elif GUI.filename_force.get()=="" and check==1 and direction!='':
        varietyname, plotname = nameMissing(varietyname, plotname)
        filename = str(varietyname+str(",")+plotname+"_"+direction)
    elif GUI.filename_force.get()=="" and check==0 and direction !='':
```

```
      filename = datestring+","+time.strftime("%H%M")+"_"+direction
   elif GUI.filename_force.get()!="" and check==1 and direction !='':
      varietyname, plotname = nameMissing(varietyname, plotname)
      filename = str(varietyname+str(",")+plotname+str("_")+direction)
   elif GUI.filename_force.get()!="" and check==0 and direction !='':
      if ("side1" in filename) or ("side2" in filename) or ("side3" in filename) or ("forward" in
filename) or ("postTest" in filename):
         filename = nameDirectionScrub(GUI.filename_force.get())
         filename = filename+"_"+direction
      else:
         filename = filename+"_"+direction
   elif GUI.filename_force.get()=="" and check==0 and direction =='':
      filename = datestring+","+time.strftime("%H%M")
   elif GUI.filename_force.get()!="" and check==1 and direction =='':
      varietyname, plotname = nameMissing(varietyname, plotname)
      filename = str(varietyname+str(",")+plotname)
   elif GUI.filename_force.get()!="" and check==0 and direction =='':
      if ("side1" in filename) or ("side2" in filename) or ("side3" in filename) or ("forward" in
filename) or ("postTest" in filename):
         filename = nameDirectionScrub(GUI.filename_force.get())
         filename = filename
      else:
         filename = filename
   #GUI.filename_postTest.set(filename_postTest)
   return filename
''' end: Edge cases: Filenaming '''

''' Single XLSX workbook created from all expected CSV files for 9-cell study'''
def generateXSLXcombinedFile():
   writer = pd.ExcelWriter('default.xlsx')
   for csvfilename in sys.ar[1:]:
      df = pd.read.csv(csvfilename)
      #FIX df.to_excel(writer.sheet_names=os.path.splitext(csvfilename)[0]) # "keyword cannot be an
expression"
   writer.save()
def peakClickRunAndSave(filename):
   PeakClick() # you cannot put in counts first....because they haven't been collected yet!
   #ergo, run clicks after triggered XLSX workbook creation
''' trigger with button, on Initial Inputs page. Button also clears all data from stemberry, wait it dods
not triggers PeakClick.py, which saves to a separate CSV before all CSV's are wrapped into a xlsx
workbook.
'''

#Bebee legacy
# * # DATA COLLECTION FUNCTION - Acquires live data from Arduino # * #
def run(self, ser):
   try:
      started = 's'
```

```
      ser.write(started.encode()) #sends 's' to arduino, telling it to start
      print('send s to arduino, legacy')
   except:
      errors.append('serial com. (start data)') # label
      eCode = 'e2'
      errorCodes.append(eCode)
      popup('start data collect')

   ser.flush()
   time.sleep(.1)
   #Don't need this:
   #try:
   #ser_bytes = ser.readline()
   #decoded_bytes.insert(0,(ser_bytes[0:len(ser_bytes)-2].decode("utf-8")))#translates bytes to
string, inserts incoming data in decoded_bytes list
   #except:
    #  popup("communication")

   #DATA COLLECTION CODE

   if vis == 's':# data displayed in scrollbars (default)
      # Displays incoming data
      scroll = Scrollbar(self)

      RecordForce.timeLabel = tk.Label(self, text = "s",font = ("arial", 14, "bold"), fg = "dodgerblue2",
bg = "ghost white")
      RecordForce.timeLabel.place(x = 274, y = 70)
      RecordForce.Timelist = Listbox(self, yscrollcommand = scroll.set, bg = "ghost
white",highlightbackground = "gray2", width = 7, height = 1, font = ("arial", 14, "bold"), fg =
"dodgerblue2")
      RecordForce.Timelist.place(x = 240, y = 100)

      RecordForce.disLabel = tk.Label(self, text = "in.",font = ("arial", 14, "bold"), fg = "dodgerblue2",
bg = "ghost white")
      RecordForce.disLabel.place(x = 357, y = 70)
      RecordForce.Dislist = Listbox(self, yscrollcommand = scroll.set, bg = "ghost
white",highlightbackground = "gray2", width = 7, height = 1, font = ("arial", 14, "bold"), fg =
"dodgerblue2")
      RecordForce.Dislist.place(x = 330, y = 100)

      RecordForce.forceLabel = tk.Label(self, text = "lbs.",font = ("arial", 14, "bold"), fg =
"dodgerblue2", bg = "ghost white")
      RecordForce.forceLabel.place(x = 444, y = 70)
      RecordForce.Forcelist = Listbox(self, yscrollcommand = scroll.set, bg = "ghost
white",highlightbackground = "gray2", width = 7, height = 11, font = ("arial", 14, "bold"), fg =
"dodgerblue2")
      RecordForce.Forcelist.place(x = 420, y = 100)
```

```
   else:# user decided for no data display
      try:#clear scrollbars if they were there
         RecordForce.Dislist.place_forget()
         RecordForce.Forcelist.place_forget()
         RecordForce.Timelist.place_forget()
         RecordForce.disLabel.place_forget()
         RecordForce.forceLabel.place_forget()
         RecordForce.timeLabel.place_forget()
      except:# no scrollbars
         print("no scrollbars")
         pass

   i = 0
   print("i = 0")
   RecordForce.elapsed = []
   RecordForce.dis = []
   RecordForce.force = []
   string = list()

   #try:

   while RecordForce.collect == True: # GUI in fSerConnect()rontend controls value of collect to
start/stop loop
      if ser.inWaiting() > 0: #checks to see if Serial is available

         try: #make sure serial data can be read/is there
            ser_bytes = ser.readline()
         except:
            errors.append('serial read') # label
            eCode = 'e3'
            errorCodes.append(eCode)
            popup("serial read")


         if i == 0:
            start = time.time() #stopwatch starts

         #DELETE?
         #decoded_bytes.insert(i,(ser_bytes[0:len(ser_bytes)-2].decode("utf-8"))) # acquires &
decodes bytes (incoming Arduino data)
         #string.insert(i,str(decoded_bytes[i])) # inserts decoded bytes into string

         bytesDecoded = (ser_bytes[0:len(ser_bytes)-2].decode("utf-8"))
         #print("bytesDecoded = ",bytesDecoded)
         string.insert(i,str(bytesDecoded)) # inserts decoded bytes into string
         #print(' run ser read ', string[i]) # useful debugging tool
         split = string[i].split("|") # splits data at | (1st = distance, 2nd = force)
         print("split = ",split)
```

```
        if len(split) >= 2 and split[0] != "" and split[1] != "": #makes sure data is in proper formatting
before processing (else pair: A)
            inches = split[0]
            pounds = split[1]

            try:
               RecordForce.elapsed.insert(i, time.time() - start)# list of elapsed time
               RecordForce.dis.insert(i, float(inches))# list of inches traveled
               RecordForce.force.insert(i, float(pounds))# list of force traveled

            except:
               errors.append('data append') # label
               eCode = 'e4'
               errorCodes.append(eCode)
             #   popup("Arduino data error")
              #  print(string[i])

            '''Scrollbars Options'''
            '''
            # if scrollbars option = on:
            try: # puts data on GUI display by default (user can turn off)
               self.Dislist.insert(END, str(dis[i]))# inserts at end of listbox to actually display
               self.Dislist.see(END)# makes sure listbox is at end so it displays live data
               self.Forcelist.insert(END, str('%.2f' % force[i]))
               self.Forcelist.see(END)
               self.Timelist.insert(END, str('%.2f' % elapsed[i]))
               self.Timelist.see(END)

            #scrollbars options = off
            except:
               pass

            i = i+1

                  '''
        else: # skips incoming data if not in right format (if pair: A
            errors.append('data skip (incorrect format)') # label
            eCode = 'e5'
            errorCodes.append(eCode)
            '''
    except:
        if RecordForce.collect == True:
            errors.append('serial disconnect')
            eCode = 'e6'
            errorCodes.append(eCode)
        else:
            pass
            '''
```

```
'''Classes, Tkinter GUI'''
# GUI overarching class
class GUI(tk.Tk):
   def __init__(self, *args, **kwargs):# automatically runs

      tk.Tk.__init__(self, *args, **kwargs)

      GUI.initializeVarsGUI()
      GUI.refreshAll()

      container = tk.Frame(self)
      container.pack(side='top', fill='both',expand = True)
      container.grid_rowconfigure(0, weight=1)
      container.grid_columnconfigure(0, weight=1)

      # top menu configuration
      menubar = Menu(container)
      filemenu = Menu(menubar, tearoff=0)
      datamenu = Menu(menubar, tearoff=0)
      pagemenu = Menu(menubar, tearoff=0)

      filemenu.add_command(label='Serial Reconnect', command = lambda:SerReconnect())
      filemenu.add_command(label='Choose Output Folder', command =
lambda:popup_chooseFolder())
      filemenu.add_command(label='Errors', command = lambda:showErrors())
      filemenu.add_command(label='Save State', command = lambda:createBackupFile())
      filemenu.add_command(label="Exit", command = lambda:close())
      pagemenu.add_command(label="Guide", command=lambda:GUI.show_frame(Guide))
      pagemenu.add_command(label="Initial Inputs",
command=lambda:GUI.show_frame(InitialInputs))
      pagemenu.add_command(label="Record Force",
command=lambda:GUI.show_frame(RecordForce))
      pagemenu.add_command(label="Post Test Inputs",
command=lambda:GUI.show_frame(FinalInputs))
      pagemenu.add_command(label="Calibrate", command=lambda:GUI.show_frame(Calibrate))
      pagemenu.add_command(label="Stem Count PreTest, Classic",
command=lambda:GUI.show_frame(StemCountClassic))
      datamenu.add_command(label="Data Feed Display, On", command =
lambda:data_display(True))
      datamenu.add_command(label="Data Feed Display, Off", command =
lambda:data_display(False))

      menubar.add_cascade(label='File', menu=filemenu)
      menubar.add_cascade(label="Pages", menu=pagemenu)
      menubar.add_cascade(label="Livestream Data Recording", menu=datamenu)
```

```
        tk.Tk.config(self, menu=menubar)
        GUI.frames = {}# empty dictionary

        for F in (InitialInputs, RecordForce, FinalInputs, Calibrate, Guide, ErrorReport,
StemCountClassic):# must put all pages in here
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky='nsew')
            frame.configure(background = 'ghost white')

        GUI.show_frame(InitialInputs)

    def initializeVarsGUI():
        GUI.filename_force = StringVar()
        GUI.filename_preTest = StringVar()
        GUI.filename_postTest = StringVar()
        GUI.filename_all = StringVar()
        GUI.varietyname = StringVar()
        GUI.plotname = StringVar()
        GUI.stemheight = DoubleVar()
        GUI.currentdirection = StringVar()#
        GUI.barmiddle = DoubleVar() #
        GUI.barbottom = DoubleVar() #
        GUI.passfilllednames_checkbox =  IntVar() # revert
        GUI.timestring = StringVar()
        GUI.startRange1, GUI.startRange2, GUI.startRange3 =  DoubleVar(),  DoubleVar(),  DoubleVar() #
cm = StringVar()
        GUI.addressInput = StringVar()


GUI.cell1Mass,GUI.cell2Mass,GUI.cell3Mass,GUI.cell4Mass,GUI.cell5Mass,GUI.cell6Mass,GUI.cell7M
ass,GUI.cell8Mass,GUI.cell9Mass =  DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(),
DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar()

GUI.cell1Count,GUI.cell2Count,GUI.cell3Count,GUI.cell4Count,GUI.cell5Count,GUI.cell6Count,GUI.ce
ll7Count,GUI.cell8Count,GUI.cell9Count =  DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(),
DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar()

GUI.cell1Diameter1,GUI.cell2Diameter1,GUI.cell3Diameter1,GUI.cell4Diameter1,GUI.cell5Diameter1
,GUI.cell6Diameter1,GUI.cell7Diameter1,GUI.cell8Diameter1,GUI.cell9Diameter1 =  DoubleVar(),
DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(),
DoubleVar()

GUI.cell1Diameter2,GUI.cell2Diameter2,GUI.cell3Diameter2,GUI.cell4Diameter2,GUI.cell5Diameter2
,GUI.cell6Diameter2,GUI.cell7Diameter2,GUI.cell8Diameter2,GUI.cell9Diameter2 =  DoubleVar(),
DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(),
DoubleVar()
```

```
GUI.cell1Diameter3,GUI.cell2Diameter3,GUI.cell3Diameter3,GUI.cell4Diameter3,GUI.cell5Diameter3
,GUI.cell6Diameter3,GUI.cell7Diameter3,GUI.cell8Diameter3,GUI.cell9Diameter3 =  DoubleVar(),
DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(),
DoubleVar()

GUI.cell1Diameter4,GUI.cell2Diameter4,GUI.cell3Diameter4,GUI.cell4Diameter4,GUI.cell5Diameter4
,GUI.cell6Diameter4,GUI.cell7Diameter4,GUI.cell8Diameter4,GUI.cell9Diameter4 =  DoubleVar(),
DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(), DoubleVar(),
DoubleVar()

    ''' Non-tkinter GUI vars, initialize ''' # for nine cell assessment, save state
    # may as well keep everything here, for fun
    GUI.errors = [] # for tracking errors
    GUI.errorCodes = [] # for tracking errors
    GUI.ignoreserial = ignoreserial
    GUI.address = address

    GUI.forcePushed = []
    GUI.distanceTraveled = []
    GUI.timeElapsed = []
    GUI.travelvelocity = []
    GUI.samplingrate = []

    GUI.forcePushed_side1 = []
    GUI.forcePushed_side2 = []
    GUI.forcePushed_side3 = []
    GUI.forcePushed_forward = []
    GUI.distanceTraveled_side1 = []
    GUI.distanceTraveled_side2 = []
    GUI.distanceTraveled_side3 = []
    GUI.distanceTraveled_forward = []
    GUI.timeElapsed_side1 = []
    GUI.timeElapsed_side2 = []
    GUI.timeElapsed_side3 = []
    GUI.timeElapsed_forward =  []
    GUI.peaks_force_side1 = []
    GUI.peaks_force_side2 = []
    GUI.peaks_force_side3 = []
    GUI.peaks_force_forward = []
    GUI.peaks_distance_side1 = []
    GUI.peaks_distance_side2 = []
    GUI.peaks_distance_side3 = []
    GUI.peaks_distance_forward = []
    GUI.peaks_time_side1 = []
    GUI.peaks_time_side2 = []
    GUI.peaks_time_side3 = []
    GUI.peaks_time_forward = []
```

```
        GUI.peaks_force = []
        GUI.peaks_distance = []
        GUI.peaks_time = []

        peakclick.peaks_force = []
        peakclick.peaks_distance = []
        peakclick.peaks_time = []

        GUI.stemcounts = []

        GUI.peak_force_cell1, GUI.peak_force_cell2, GUI.peak_force_cell3, GUI.peak_force_cell4,
GUI.peak_force_cell5, GUI.peak_force_cell6, GUI.peak_force_cell7, GUI.peak_force_cell8,
GUI.peak_force_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
        GUI.peak_distance_cell1, GUI.peak_distance_cell2, GUI.peak_distance_cell3,
GUI.peak_distance_cell4, GUI.peak_distance_cell5, GUI.peak_distance_cell6,
GUI.peak_distance_cell7, GUI.peak_distance_cell8, GUI.peak_distance_cell9 =
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
        GUI.peak_time_cell1, GUI.peak_time_cell2, GUI.peak_time_cell3, GUI.peak_time_cell4,
GUI.peak_time_cell5, GUI.peak_time_cell6, GUI.peak_time_cell7, GUI.peak_time_cell8,
GUI.peak_time_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0

        GUI.data_preTest,GUI.data_recordForce,GUI.data_postTest,GUI.data_peaks,GUI.data_EI =
[],[],[],[],[]

    def refreshAll(): #clearall(self)?


        GUI.filename_force.set("")
        GUI.filename_preTest.set("")
        GUI.filename_postTest.set("")
        GUI.filename_all.set("")
        GUI.varietyname.set("")
        GUI.plotname.set("")
        GUI.startRange1.set(50)
        GUI.startRange2.set(150)
        GUI.startRange3.set(250) # centimeters
        GUI.stemheight.set(default_stemheight) # cm
        GUI.barbottom.set(round(GUI.stemheight.get()*initial_barbottomOverStemheight_coeff,3)) #
cm
        GUI.barmiddle.set(round(GUI.barbottom.get()+barradius,3)) # cm
        GUI.passfilllednames_checkbox.set(1)
        GUI.timestring.set(time.strftime("%H%M"))
        GUI.currentdirection.set("")
        GUI.addressInput.set("")

        ''' Set post test variables for mass, count, and diameter'''
```

```
GUI.cell1Mass.set(0),GUI.cell2Mass.set(0),GUI.cell3Mass.set(0),GUI.cell4Mass.set(0),GUI.cell5Mass.s
et(0),GUI.cell6Mass.set(0),GUI.cell7Mass.set(0),GUI.cell8Mass.set(0),GUI.cell9Mass.set(0)

GUI.cell1Count.set(0),GUI.cell2Count.set(0),GUI.cell3Count.set(0),GUI.cell4Count.set(0),GUI.cell5Co
unt.set(0),GUI.cell6Count.set(0),GUI.cell7Count.set(0),GUI.cell8Count.set(0),GUI.cell9Count.set(0)

GUI.cell1Diameter1.set(0),GUI.cell2Diameter1.set(0),GUI.cell3Diameter1.set(0),GUI.cell4Diameter1.
set(0),GUI.cell5Diameter1.set(0),GUI.cell6Diameter1.set(0),GUI.cell7Diameter1.set(0),GUI.cell8Diam
eter1.set(0),GUI.cell9Diameter1.set(0)

GUI.cell1Diameter2.set(0),GUI.cell2Diameter2.set(0),GUI.cell3Diameter2.set(0),GUI.cell4Diameter2.
set(0),GUI.cell5Diameter2.set(0),GUI.cell6Diameter2.set(0),GUI.cell7Diameter2.set(0),GUI.cell8Diam
eter2.set(0),GUI.cell9Diameter2.set(0)

GUI.cell1Diameter3.set(0),GUI.cell2Diameter3.set(0),GUI.cell3Diameter3.set(0),GUI.cell4Diameter3.
set(0),GUI.cell5Diameter3.set(0),GUI.cell6Diameter3.set(0),GUI.cell7Diameter3.set(0),GUI.cell8Diam
eter3.set(0),GUI.cell9Diameter3.set(0)

GUI.cell1Diameter4.set(0),GUI.cell2Diameter4.set(0),GUI.cell3Diameter4.set(0),GUI.cell4Diameter4.
set(0),GUI.cell5Diameter4.set(0),GUI.cell6Diameter4.set(0),GUI.cell7Diameter4.set(0),GUI.cell8Diam
eter4.set(0),GUI.cell9Diameter4.set(0)
    ''' end '''

    ''' Non-tkinter GUI vars, initialize ''' # for nine cell assessment, save state
    # may as well keep everything here, for fun
    GUI.errors = [] # for tracking errors
    GUI.errorCodes = [] # for tracking errors

    GUI.forcePushed = []
    GUI.distanceTraveled = []
    GUI.timeElapsed = []

    GUI.forcePushed_side1 = []
    GUI.forcePushed_side2 = []
    GUI.forcePushed_side3 = []
    GUI.forcePushed_forward = []
    GUI.distanceTraveled_side1 = []
    GUI.distanceTraveled_side2 = []
    GUI.distanceTraveled_side3 = []
    GUI.distanceTraveled_forward = []
    GUI.timeElapsed_side1 = []
    GUI.timeElapsed_side2 = []
    GUI.timeElapsed_side3 = []
    GUI.timeElapsed_forward =  []
    GUI.peaks_force_side1 = []
    GUI.peaks_force_side2 = []
    GUI.peaks_force_side3 = []
```

```
        GUI.peaks_force_forward = []
        GUI.peaks_distance_side1 = []
        GUI.peaks_distance_side2 = []
        GUI.peaks_distance_side3 = []
        GUI.peaks_distance_forward = []
        GUI.peaks_time_side1 = []
        GUI.peaks_time_side2 = []
        GUI.peaks_time_side3 = []

        GUI.peaks_force = []
        GUI.peaks_distance = []
        GUI.peaks_time = []

        GUI.stemcounts = []

        GUI.peak_force_cell1, GUI.peak_force_cell2, GUI.peak_force_cell3, GUI.peak_force_cell4,
GUI.peak_force_cell5, GUI.peak_force_cell6, GUI.peak_force_cell7, GUI.peak_force_cell8,
GUI.peak_force_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
        GUI.peak_distance_cell1, GUI.peak_distance_cell2, GUI.peak_distance_cell3,
GUI.peak_distance_cell4, GUI.peak_distance_cell5, GUI.peak_distance_cell6,
GUI.peak_distance_cell7, GUI.peak_distance_cell8, GUI.peak_distance_cell9 =
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
        GUI.peak_time_cell1, GUI.peak_time_cell2, GUI.peak_time_cell3, GUI.peak_time_cell4,
GUI.peak_time_cell5, GUI.peak_time_cell6, GUI.peak_time_cell7, GUI.peak_time_cell8,
GUI.peak_time_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0

        GUI.peak_EI_fullcontact_cell1, GUI.peak_EI_fullcontact_cell2, GUI.peak_EI_fullcontact_cell3,
GUI.peak_EI_fullcontact_cell4, GUI.peak_EI_fullcontact_cell5, GUI.peak_EI_fullcontact_cell6,
GUI.peak_EI_fullcontact_cell7, GUI.peak_EI_fullcontact_cell8, GUI.peak_EI_fullcontact_cell9 =
[],[],[],[],[],[],[],[],[]
        GUI.peak_EI_intermediatecontact_cell1, GUI.peak_EI_intermediatecontact_cell2,
GUI.peak_EI_intermediatecontact_cell3, GUI.peak_EI_intermediatecontact_cell4,
GUI.peak_EI_intermediatecontact_cell5, GUI.peak_EI_intermediatecontact_cell6,
GUI.peak_EI_intermediatecontact_cell7, GUI.peak_EI_intermediatecontact_cell8,
GUI.peak_EI_intermediatecontact_cell9 = [],[],[],[],[],[],[],[],[]
        GUI.peak_EI_nocontact_cell1, GUI.peak_EI_nocontact_cell2, GUI.peak_EI_nocontact_cell3,
GUI.peak_EI_nocontact_cell4, GUI.peak_EI_nocontact_cell5, GUI.peak_EI_nocontact_cell6,
GUI.peak_EI_nocontact_cell7, GUI.peak_EI_nocontact_cell8, GUI.peak_EI_nocontact_cell9 =
[],[],[],[],[],[],[],[],[]

        GUI.peaks_time_forward = []
        GUI.EI_fullcontact = []
        GUI.EI_intermediatecontact = []
        GUI.EI_nocontact = []
        GUI.AvgEI_intermediatecontact = []

        GUI.data_preTest,GUI.data_recordForce,GUI.data_postTest,GUI.data_peaks,GUI.data_EI =
[],[],[],[],[]
```

```
   def show_frame(cont):
      frame = GUI.frames[cont]
      frame.tkraise()
      frame.event_generate("<<ShowFrame>>") # event

# buttons that are the same for each page
#'''
class repeatPageButtons:
   def __init__(self, parent, controller): # automatically runs
      filler=1
   def showButtons(self, parent, controller):
      guide_button = Button(self, text = "Guide", font = ("arial", 14, "bold"), height = 2, width = 8, fg =
"ghost white", bg = "gray2",command=lambda:GUI.show_frame(Guide))
      initialInputs_button = Button(self, text = "Initial\nInputs", font = ("arial", 14, "bold"), height = 2,
width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(InitialInputs))
      recordForce_button = Button(self, text = "Record\nForce", font = ("arial", 14, "bold"), height =
2, width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(RecordForce))
      postInputs_button = Button(self, text = "Post Test\nInputs", font = ("arial", 14, "bold"), height =
2, width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(FinalInputs))

      guide_button.place(x = 0, y = 340)
      initialInputs_button.place(x = 375/3*1, y = 340)
      recordForce_button.place(x = 375/3*2, y = 340)
      postInputs_button.place(x = 375/3*3, y = 340)
      #'''
#Home page
class InitialInputs(tk.Frame):
   def __init__(self, parent, controller): # automatically runs
      # Once the program launches, the InitialInput screen will be shown for the first time, prompting
serial connection
      try:
         RecordForce.ser = SerConnect()
      except:
         GUI.ignoreserial = True
         print("Serial not connected.")

      tk.Frame.__init__(self, parent)

      ''' GUI design, non-frame '''
      pageButtons = repeatPageButtons.showButtons(self, parent, controller)
      homeheader = Label(self, text = "INITIAL INPUTS", font = ("arial", 17, "bold"), fg = "gray3",
bg="ghost white")
      unit_label = Label(self, text=str("Distance and height are in centimeters."), font = ("arial", 12,
"italic"), fg = "red4", bg="ghost white")
      savePreTestInputs_button = Button(self, text ="Save Initial Inputs", font = ("arial", 16, "bold"),
height = 1, width = 20, fg = "ghost white", bg = "dodgerblue3",
command=lambda:self.savePreTestInputs())
```

```
        variety_label = Label(self, text = "Variety: ", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white")
        varietyname_entryBox = Entry(self, textvariable=GUI.varietyname, font = ("arial", 14, "bold"), width="20", bg="white", fg="gray1")
        plotname_label = Label(self, text = "Plot: ", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white")
        plotname_entryBox = Entry(self, textvariable=GUI.plotname, font = ("arial", 14, "bold"), width="10", bg="white", fg="gray1")
        passfillednames_checkbox = Checkbutton(self, text = "Use variety & plot names", variable = GUI.passfillednames_checkbox, width=23, height=2, font = ("arial", 12), bg='ghost white')
        stemHeight_label = Label(self, text = "Avg. Stem Height (cm):", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white")
        stemHeight_entry = Entry(self, textvariable=GUI.stemheight, font = ("arial", 14, "bold"), width= 6, bg="white", fg="gray1")
        barHeight_label = Label(self, text = "Bar Middle Height (cm):", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white")
        barHeight_entry = Entry(self, textvariable=GUI.barmiddle, font = ("arial", 14, "bold"), width= 6, bg="white", fg="gray1")

        homeheader.place(x=275,y=0)
        unit_label.place(x=500,y=0)
        savePreTestInputs_button.place(x = 510, y = 340)
        variety_label.place(x=0,y=35)
        varietyname_entryBox.place(x = 80, y = 35)
        plotname_label.place(x=310,y=35)
        plotname_entryBox.place(x = 360, y = 35)
        passfillednames_checkbox.place(x = 540 , y = 25)
        stemHeight_label.place(x=0,y=240)
        stemHeight_entry.place(x = 220, y = 240)
        barHeight_label.place(x=0,y=280)
        barHeight_entry.place(x = 220, y = 280)

        ''' Frame: Range'''
        range_frame = tk.LabelFrame(self, text='Side Hit Ranges',font = ("arial", 14, "bold"), width= 10, bg="white", fg="gray1")
        range_frame.place(x = 20, y = 80)
        startRange1Dis_label = Label(range_frame, text = "Range 1 start (cm):", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=2, column=0)
        startRange2Dis_label = Label(range_frame, text = "Range 2 start (cm):", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=1, column=0)
        startRange3Dis_label = Label(range_frame, text = "Range 3 start (cm):", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=0, column=0)
        startRange1_entry = Entry(range_frame, textvariable=GUI.startRange1,font = ("arial", 14, "bold"), width= 4, bg="white", fg="gray1").grid(row=2, column=1)
        startRange2_entry = Entry(range_frame, textvariable=GUI.startRange2,font = ("arial", 14, "bold"), width= 4, bg="white", fg="gray1").grid(row=1, column=1)
        startRange3_entry = Entry(range_frame, textvariable=GUI.startRange3, font = ("arial", 14, "bold"), width= 4, bg="white", fg="gray1").grid(row=0, column=1)
```

```
        ''' end '''

        ''' Frame: Force Bar Quickset buttons'''
        barset_frame = tk.LabelFrame(self, text='Bar Bottom Quickset',font = ("arial", 14, "bold"),
width= 10, bg="white", fg="gray1")
        barset_frame.place(x = 340, y = 230)
        #button that calculates optimized force bar height
        height70percent_button = Button(barset_frame, text ="70%", font=("arial",14,"bold"),
height=1, width=6, fg="ghost white",
bg="red4",command=lambda:self.height70percent(GUI.stemheight.get()))
        height80percent_button = Button(barset_frame, text ="80%", font=("arial",14,"bold"),
height=1, width=6, fg="ghost white",
bg="red4",command=lambda:self.height80percent(GUI.stemheight.get()))
        height90percent_button = Button(barset_frame, text ="90%", font=("arial",14,"bold"),
height=1, width=6, fg="ghost white",
bg="red4",command=lambda:self.height90percent(GUI.stemheight.get()))
        height70percent_button.grid(row=0, column=0)
        height80percent_button.grid(row=0, column=1)
        height90percent_button.grid(row=0, column=2)
        ''' end '''

        ''' Frame: PreCount Buttons '' # Hide, access via menu
        precount_frame = tk.LabelFrame(self, text='Count First',font = ("arial", 10, "bold"), width= 4,
bg="white", fg="gray1")
        precount_frame.place(x = 650, y = 100)
        precount_button = tk.Button(precount_frame, text ="Don't", font=("arial",10,"bol;d"), height=1,
width=10, fg="ghost white",
bg="purple3",command=lambda:self.height70percent(GUI.stemheight.get()))
        precount_button.grid(row=0, column=4)
        '' end '''

        self.bind("<<ShowFrame>>", self.on_show_frame_InitialInputs) # why is this really here

    def height70percent(self, stemheight):
        coeff = 0.7
        GUI.barbottom.set(round(coeff*stemheight,3))
        GUI.barmiddle.set(round(GUI.barbottom.get()+barradius,3))
        print("70%: stemheight",GUI.stemheight.get(),"cm, barheight = ",GUI.barmiddle.get(),"cm,
barbottom = ",GUI.barbottom.get(),"cm")
    def height80percent(self, stemheight):
        coeff = 0.8
        GUI.barbottom.set(round(coeff*stemheight,3))
        GUI.barmiddle.set(round(GUI.barbottom.get()+barradius,3))
        print("80%: stemheight",GUI.stemheight.get(),"cm, barheight = ",GUI.barmiddle.get(),"cm,
barbottom = ",GUI.barbottom.get(),"cm")
    def height90percent(self, stemheight):
        coeff = 0.9
        GUI.barbottom.set(round(coeff*stemheight,3))
```

```
        GUI.barmiddle.set(round(GUI.barbottom.get()+barradius,3))
        print("90%: stemheight",GUI.stemheight.get(),"cm, barheight = ",GUI.barmiddle.get(),"cm,
barbottom = ",GUI.barbottom.get(),"cm")

    def savePreTestInputs(self):
        GUI.barbottom.set(round(GUI.barmiddle.get()-barradius,3)) # cm
        print(str(int(GUI.barbottom.get()/GUI.stemheight.get()*100)),"%:
stemheight",GUI.stemheight.get(),"cm, barheight = ",GUI.barmiddle.get(),"cm, barbottom =
",GUI.barbottom.get(),"cm")

        variety, plot, stemheight, barbottom, barmiddle, startRange1, startRange2, startRange3 =
['variety'], ['plot'], ['stemheight(cm)'], ['barbottom(cm)'], ['barmiddle(cm)'], ['startRange1(cm)'],
['startRange2(cm)'], ['startRange3(cm)']
        variety.append(GUI.varietyname.get())
        plot.append(GUI.plotname.get())
        stemheight.append(GUI.stemheight.get())
        barbottom.append(GUI.barbottom.get())
        barmiddle.append(GUI.barmiddle.get())
        startRange1.append(GUI.startRange1.get())
        startRange2.append(GUI.startRange2.get())
        startRange3.append(GUI.startRange3.get())

        update_filename_preTest()
        filename_preTest_csv = GUI.address + '/' + GUI.filename_preTest.get() + '.csv'

        if overwriteGuard(filename_preTest_csv) == True: # filename already exists, needs to be
renamed
            rename(GUI.filename_preTest.get()) # prompt user to rename file
        ''' write CSV'''
        GUI.data_preTest = [variety, plot, stemheight, barbottom, barmiddle, startRange1, startRange2,
startRange3]
        columns_data_preTest = zip_longest(*GUI.data_preTest)
        with open(filename_preTest_csv,'w',newline='') as f:
            writer = csv.writer(f)
            writer.writerows(columns_data_preTest)
        ''' end: write CSV '''
        print("filename_preTest_csv = "+filename_preTest_csv)

    def on_show_frame_InitialInputs(self, event):
        filler=1
        #print("show Initial Inputs screen")

# Data collection page
class RecordForce(tk.Frame):
    def __init__(self, parent, controller):# automatically runs

        RecordForce.peaks_force = []
        RecordForce.peaks_distance = []
```

```
        RecordForce.peaks_time = []

        self.legends = []

        tk.Frame.__init__(self, parent)
        self.controller = controller #

        RecordForce.container = tk.Frame(self)

        ''' GUI design, non-frame '''
        pageButtons = repeatPageButtons.showButtons(self, parent, controller)
        title = Label(self, text ="RECORD FORCE", font = ("arial", 17, "bold"), fg = "gray3", bg="ghost
white")
        filename_label = Label(self, text = "Filename: ", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white")
        filename_entryBox = Entry(self, textvariable=GUI.filename_force, font = ("arial", 14, "bold"),
width="32", bg="white", fg="gray1")
        self.checkAutoGraph =  IntVar() # on/off control of auto graph after stopping & saving data
        #self.checkAutoGraph.set(1)
        self.checkAutoGraph.set(0)
        graph_checkbox = Checkbutton(self, text = "Auto graph", variable = self.checkAutoGraph, width
= 13, height = 2, bg = 'ghost white')

        title.place(x=275,y=0)
        filename_label.place(x=0,y=80)
        filename_entryBox.place(x = 110, y = 80)
        graph_checkbox.place(x = 675 , y = 0)

        RecordForce.datafeed_frame = tk.LabelFrame(self, text='Data Feed',font = ("arial", 14, "bold"),
width= 10, bg="white", fg="gray1")
        RecordForce.datafeed_frame.place(x = 20, y = 0)
        clear_button = Button(RecordForce.datafeed_frame,text = "Clear",font = ("arial", 16, "bold"),
height = 1, width = 6, fg = "ghost white", bg = "red4",command=lambda:RecordForce.clearDisplay())
        clear_button.grid(row=0, column=0)
        RecordForce.msgbox =  tk.LabelFrame(self, text='',font = ("arial", 14, "bold"), width= 10,
bg="white", fg="gray1")

        RecordForce.msgbox.place(x = 5, y = 120)
        #forceSaved_label.place(x=5, y = 120)

        ''' Frame: Filename Quickset buttons'''
        nameset_frame = tk.LabelFrame(self, text='Filename\nQuickset',font = ("arial", 14, "bold"),
width= 10, bg="white", fg="gray1")
        nameset_frame.place(x = 570, y = 40)
        #button that calculates optimized force bar height
        side1TestButton = Button(nameset_frame, text = "Side 1", font = ("arial", 16, "bold"), height = 1,
width = 6, fg = "ghost white", bg = "red4",command=lambda:self.nameSide1())
```

```
        side2TestButton = Button(nameset_frame, text = "Side 2", font = ("arial", 16, "bold"), height = 1,
width = 6, fg = "ghost white", bg = "red4",command=lambda:self.nameSide2())
        side3TestButton = Button(nameset_frame, text = "Side 3", font = ("arial", 16, "bold"), height = 1,
width = 6,fg = "ghost white", bg = "red4",command=lambda:self.nameSide3())
        forwardTestButton = Button(nameset_frame, text = "Forward", font = ("arial", 16, "bold"),
height = 1, width = 6, fg = "ghost white", bg = "red4",command=lambda:self.nameForward())
        increment_button = Button(nameset_frame, text = "+1", font = ("arial", 16, "bold"), height = 1,
width = 6, fg = "ghost white", bg =
"purple4",command=lambda:self.incrementName_Force(GUI.filename_force.get()))

        side1TestButton.grid(row=0, column=0)
        side2TestButton.grid(row=1, column=0)
        side3TestButton.grid(row=2, column=0)
        forwardTestButton.grid(row=3, column=0)
        increment_button.grid(row=4, column=0)
        ''' end '''

        ''' Record Data Frame'''
        dataButtons_frame = tk.LabelFrame(self, text='',font = ("arial", 14, "bold"), width= 10,
bg="white", fg="gray1")
        dataButtons_frame.place(x = 675, y = 40)
        #tells Arduino to start collecting data
        start_button = Button(dataButtons_frame, text = "Start", font = ("arial", 16, "bold"), height = 3,
width = 8, fg = "ghost white", bg = "dodgerblue3",command=lambda:RecordForce.startCollect())
        #tells Arduino to stop collecting data & saves the data (calls filename function)
        stop_button = Button(dataButtons_frame, text = "Stop\n&\nSave", font = ("arial", 16, "bold"),
height = 3, width = 8, fg = "ghost white", bg =
"dodgerblue3",command=lambda:RecordForce.stopAndSave())
        #LEGACY, NOPE: stop_button = Button(dataButtons_frame, text = "Stop\n&\nSave", font =
("arial", 16, "bold"), height = 3, width = 8, fg = "ghost white", bg =
"dodgerblue3",command=lambda:RecordForce.stop())
        #tares/zeros load cell
        tare_button = Button(dataButtons_frame, text = "Tare", font = ("arial", 16, "bold"), height = 3,
width = 8, fg = "ghost white", bg = "dodgerblue3",command=lambda:RecordForce.tare())

        start_button.grid(row = 1, column = 0)
        stop_button.grid(row = 2, column = 0)
        tare_button.grid(row = 3, column = 0)
        ''' end frame'''

        self.bind("<<ShowFrame>>", self.on_show_frame_RecordForce)

    def nameForward(self):
        direction = "forward"
        filename_force = nameBlackBox(direction,GUI.filename_force.get())
        GUI.filename_force.set(filename_force)
        GUI.currentdirection.set(direction)
    def nameSide1(self):
```

```
    direction = "side1"
    filename_force = nameBlackBox(direction,GUI.filename_force.get())
    GUI.filename_force.set(filename_force)
    GUI.currentdirection.set(direction)
def nameSide2(self):
    direction = "side2"
    filename_force = nameBlackBox(direction,GUI.filename_force.get())
    GUI.filename_force.set(filename_force)
    GUI.currentdirection.set(direction)
def nameSide3(self):
    direction = "side3"
    filename_force = nameBlackBox(direction,GUI.filename_force.get())
    GUI.filename_force.set(filename_force)
    GUI.currentdirection.set(direction)
def nameFresh(varietyname,plotname):
    direction = ""
    filename_force = nameBlackBox(direction,GUI.filename_force.get())
    GUI.filename_force.set(filename_force)
    set(direction)

def clearDisplay():
    time.sleep(0.3)
    print('You hit the "Clear" button. Please develop clearDisplay().')
    GUI.refreshAll()
    '''
    try:
        RecordForce.Forcelist.delete(0, 'end')
        RecordForce.Dislist.delete(0, 'end')
        RecordForce.Timelist.delete(0, 'end')
        print('You hit the "Clear" button and deleted recorded data. This was not useful.')
    except:
        pass
        '''

def incrementName_Force(self,filename):
    newName = incrementName(filename)
    GUI.filename_force.set(newName)


# calls run function (for collecting Arduino data) to run in backend while GUI runs in frontend
def startCollect():
    now = datetime.datetime.now()
    unix_now = time.mktime(now.timetuple())
    time.sleep(0.4) # for visual effect
    #threading run function (simultaneously performs run function in backend)
    if GUI.ignoreserial == False:
        if RecordForce.ser.isOpen() == False:
            RecordForce.ser.open()
```

```
        RecordForce.legacy = False # bebee frankserial, lez go, 08/31/2022
        print("RecordForce.legacy = ",RecordForce.legacy)
        if RecordForce.legacy == False:
            runDataCollect()
        elif RecordForce.legacy == True:
            RecordForce.start()
        if visualizeDatastream == True:
            thread2_visualizeData = threading.Thread(target = datafeed,args=(RecordForce.container))
            thread2_visualizeData.start()
    else:
        print("Data collection not run, because GUI.ignoreserial ==",str(GUI.ignoreserial),"...")

# saves raw force data # Bebee legacy method
def start():
    RecordForce.collect = True
    if RecordForce.ser.isOpen() == False:
      RecordForce.ser.open()
    #threading run function (simultaneously performs run function in backend)
    t1 = threading.Thread(target = run,args=(RecordForce, RecordForce.ser))
    t1.start()

#bebee Legacy
def stop():
    RecordForce.ser.flushInput()# wait until all data is written
    RecordForce.collect = False

    try:
        stopped = 'x'
        RecordForce.ser.write(stopped.encode())# sends 'x' to Arduino to stop reading sensors
        time.sleep(.5)# for potential error protection?
        #ser.close()
    except:
        errors.append('serial com. (stopping data)') # error label
        eCode = 'e7'
        errorCodes.append(eCode)
    finally:

        GUI.timeElapsed = RecordForce.elapsed
        GUI.distanceTraveled = RecordForce.dis
        GUI.forcePushed = RecordForce.force
        RecordForce.saveForce()# run the Save Raw Data function


def sendStart():
    if RecordForce.ser.isOpen() == False:
      RecordForce.ser.open()

    started = 's'
```

```
        print("\nPython sent "+started+".")
        RecordForce.hasStarted = False
        RecordForce.hasSentStop = False
        RecordForce.hasStopped = False
        # wipe vars
        GUI.forcePushed = []
        GUI.distanceTraveled = []
        GUI.timeElapsed = []
        RecordForce.datastream = []
        #thread2_count_stop.start()
        while RecordForce.hasStarted == False: # len(line)==0
            RecordForce.ser.write(started.encode())
            time.sleep(sleepSend) # if this is on, it takes another two seconds to start, but the arduino
yells less.
            if RecordForce.ser.in_waiting > 0: # this does happen
                ser_bytes = RecordForce.ser.readline()
                line = ser_bytes.decode('utf-8').rstrip()
                if line =="Started!": #if line == started:
                    RecordForce.hasStarted = True
                    RecordForce.startTime = time.time() #stopwatch starts
                    RecordForce.i = 0
                    print(started+" received by arduino.")

    def sendStop():
        stopped = 'x'
        RecordForce.hasSentStop = True
        print("Python sent "+stopped+".")

        while RecordForce.hasStopped == False: # len(line)==0
            print("brake", end =" ")
            RecordForce.ser.write(stopped.encode())
            RecordForce.ser.flush()
            time.sleep(sleepSend)
            if RecordForce.ser.in_waiting > 0: # this does happen
                bytecount = RecordForce.ser.in_waiting
                ser_bytes = RecordForce.ser.read(bytecount)
                line = ser_bytes.decode('utf-8').rstrip()
                datapacket = line.splitlines()
                #print(line)
                if line =="Stopped!" or ("Stopped!" in datapacket): #if line == stopped:
                #if ("Stopped!" in ser_bytes): #if line == stopped:
                    RecordForce.hasStopped = True
                    print(stopped+" received by arduino.")
                    #RecordForce.ser.close()
                    #print(RecordForce.dataStream)
                    '''
                    RecordForce.allocateNineCellData() # what is this for, nine cell stuff?
                    '''
```

```
    RecordForce.ser.close()
    print("Test runtime: ",max(GUI.timeElapsed)," seconds.") # /1000

def stopAndSave():
    if RecordForce.legacy == True:
        RecordForce.stop()
    else:
        time.sleep(.1)
        if GUI.ignoreserial == False:

            testForNineCellFilename()
            if RecordForce.ser.isOpen():
                try:
                    RecordForce.ser.flushInput()# wait until all data is written
                    #print("Not flushing input.")
                except:
                    print("failed RecordForce.ser.flushInput()")
                RecordForce.sendStop()

            RecordForce.saveForce()
        else:
            print("File not saved. GUI.ignoreserial == True.")

def saveForce():
    createBackupFile()
    # force data filename
    filename_force = GUI.filename_force.get()
    filename_force_csv = GUI.address + '/' + (GUI.filename_force.get()) + '.csv'
    if overwriteGuard(filename_force_csv) == True: # filename already exists, needs to be renamed
        rename(filename_force) # prompt user to rename file

    #ave. SOCEM velocity
    avelocity = ["AvgTravelVelocity(cm/s)"]
    try:
        travelvelocity = max(GUI.distanceTraveled)/(max(GUI.timeElapsed)) #cm/s # /1000
        avelocity.append(travelvelocity)
    except:
        travelvelocity=0
        avelocity.append(travelvelocity)

    #Sampling Rate
    sampling=["SamplingRate(Hz)"]
    hz = list()
    try:
        for i in range(len(GUI.distanceTraveled)-1):
            change = GUI.timeElapsed[i+1] - GUI.timeElapsed[i] # ms
            hz.append(change)
        rate = sum(hz)/len(hz) # why flip?
```

```
            sampling.append(1/rate) # why flip?
        except:
            rate = 0
            sampling.append(0)

        GUI.travelvelocity = travelvelocity
        GUI.samplingrate = 1/rate # changed from 1/rate, to avoid a divide by zero error

        RecordForce.sidehitPeakClick()

        if GUI.ignoreserial == False and len(GUI.forcePushed)>0:
            GUI.distanceTraveled.insert(0, "Distance(cm)")
            GUI.forcePushed.insert(0, "Force(N)")
            GUI.timeElapsed.insert(0 , "Time(sec)")

            ''' write CSV'''
            GUI.data_recordForce = [GUI.timeElapsed,GUI.distanceTraveled, GUI.forcePushed, avelocity,
sampling,RecordForce.peaks_force,RecordForce.peaks_distance,RecordForce.peaks_time]
            columns_data_recordForce = zip_longest(*GUI.data_recordForce)
            with open(filename_force_csv,'w',newline='') as f:
                writer = csv.writer(f)
                writer.writerows(columns_data_recordForce)
            ''' end: write CSV '''
            print("filename_force_csv = "+filename_force_csv)

            # tell user raw data was saved
            #print("File saved: "+GUI.filename_force.get()+".csv\n")
            try:
                forceSaved_label = Label(RecordForce.msgbox, text = "Force data saved.", font = ("arial",
14, "bold"), fg = "dodgerblue3", bg = "ghost white")
                #forceSaved_label = Label(RecordForce.msgbox, text = "Force data saved.", font = ("arial",
14, "bold"), fg = "dodgerblue3", bg = "ghost white").grid(row=0, column=0)
            except:
                print("attempt to generate in-window forceSaved_label messsage. fail, dave.")
        else:
            print("Force data not saved. GUI.ignoreserial = "+str(GUI.ignoreserial)+".
len(GUI.forcePushed) = ",len(GUI.forcePushed))

        #RecordForce.clearDisplay()
        '''
        self.instantGraph()
        '''
        '''
        Why clear?
        GUI.timeElapsed.clear()
        GUI.forcePushed.clear()
        GUI.distanceTraveled.clear()
        avelocity.clear()
```

```
        hz.clear()
        sampling.clear()
        '''

    '''
    def overwriteGuard(self, filename):# prevents overwriting by checking if filename already exists in
saving folder
        return path.exists(filename) # True = already exits, False = doesn't exist
    '''
    '''
    #auto graph feature
    def instantGraph(self):
        try:
        #if self.dataset-1 <= 1:
            self.legends = []
        except:
            pass

        if not plt.get_fignums():#if graph figure was closed, reset legend
            self.legends.clear()
            #print("new fig who dis")
        self.legends.append(GUI.filename_force.get())#add current filename to legend
        #fig = plt.figure(figsize=(8,4.8)) #fig size control
        #plots force displacement graph
        print("len(GUI.distanceTraveled) = ",len(GUI.distanceTraveled))
        if self.checkAutoGraph.get() == 1 and len(GUI.distanceTraveled)>5 and GUI.ignoreserial ==
False:
            plt.plot(GUI.distanceTraveled, GUI,forcePushed)
            plt.xlabel("Distance (cm)")
            plt.ylabel("Force (N)")
            plt.title(filename.get())
            plt.legend(self.legends)
            plt.axis = ([min(distance), max(distance), min(force), max(force)])
            plt.show()
        else:
            print("There is no data to graph. Try GUI.ignoreserial = False, in StemBerry.")
        '''
    def sidehitPeakClick():
        RecordForce.peaks_force,RecordForce.peaks_distance,RecordForce.peaks_time= [],[],[]
        # currently only lauches click assessment for side1, side2, side3
        #print("GUI.currentdirection = ",GUI.currentdirection.get())
        #print("len(GUI.forcePushed) = ",len(GUI.forcePushed))
        if (assessAllTests == True) or (GUI.currentdirection.get() == "side1") or
(GUI.currentdirection.get() == "side2") or (GUI.currentdirection.get() == "side3"):
        #if True:
            if len(GUI.forcePushed)>0:
                varietyAndPlotnameAndDetail = GUI.filename_force.get()
                RecordForce.plotshown = True
```

```
            RecordForce.closedplt = False
            RecordForce.thread3_plotchecker = threading.Thread(target = RecordForce.plotchecker)
            RecordForce.thread3_plotchecker.start()

peakclick.peakclick(GUI.forcePushed,GUI.distanceTraveled,GUI.timeElapsed,GUI.filename_force.get(
),GUI.address,GUI.travelvelocity)
            #RecordForce.peaks_force,RecordForce.peaks_distance,RecordForce.peaks_time =
peakclick.peaks_force,peakclick.peaks_distance,peakclick.peaks_time

            #
RecordForce.sortClicks(RecordForce.peaks_force,RecordForce.peaks_distance,RecordForce.peaks_ti
me)
            print("Delete this note about side hit completing")
         else:
            print("PeaksClick figure not triggered because len(GUI.forcePushed) = 0.")
   def plotchecker():
      while RecordForce.plotshown == True:
         time.sleep(.1)
         if RecordForce.closedplt == True:
            time.sleep(.1)
            RecordForce.sortClicks()
            RecordForce.plotshown = False
         '''
         else:
            print("loop while")
            '''
   def allocateNineCellData():
      print("allocate")

   def sortClicks():
      if GUI.currentdirection.get() == "side1":
         if len(RecordForce.peaks_force) == 3:
            GUI.peak_force_cell1, GUI.peak_force_cell2, GUI.peak_force_cell3 =
RecordForce.peaks_force[0],RecordForce.peaks_force[1],RecordForce.peaks_force[2]
         elif len(RecordForce.peaks_force) == 4:
            GUI.peak_force_cell1, GUI.peak_force_cell2, GUI.peak_force_cell3 =
RecordForce.peaks_force[1],RecordForce.peaks_force[2],RecordForce.peaks_force[3]
         if len(RecordForce.peaks_distance) == 3:
            GUI.peak_distance_cell1, GUI.peak_distance_cell2, GUI.peak_distance_cell3 =
RecordForce.peaks_distance[0],RecordForce.peaks_distance[1],RecordForce.peaks_distance[2]
         elif len(RecordForce.peaks_distance) == 4:
            GUI.peak_distance_cell1, GUI.peak_distance_cell2, GUI.peak_distance_cell3 =
RecordForce.peaks_distance[1],RecordForce.peaks_distance[2],RecordForce.peaks_distance[3]
         if len(RecordForce.peaks_time) == 3:
            GUI.peak_time_cell1, GUI.peak_time_cell2,
GUI.peak_time_cell3=RecordForce.peaks_time[0],RecordForce.peaks_time[1],RecordForce.peaks_ti
me[2]
         elif len(RecordForce.peaks_time) == 4:
```

```
            GUI.peak_time_cell1, GUI.peak_time_cell2,
GUI.peak_time_cell3=RecordForce.peaks_time[1],RecordForce.peaks_time[2],RecordForce.peaks_ti
me[3]

    elif GUI.currentdirection.get() == "side2":
        if len(RecordForce.peaks_force) == 3:
            GUI.peak_force_cell4, GUI.peak_force_cell5, GUI.peak_force_cell6 =
RecordForce.peaks_force[0],RecordForce.peaks_force[1],RecordForce.peaks_force[2]
        elif len(RecordForce.peaks_force) == 4:
            GUI.peak_force_cell4, GUI.peak_force_cell5, GUI.peak_force_cell6 =
RecordForce.peaks_force[1],RecordForce.peaks_force[2],RecordForce.peaks_force[3]
        if len(RecordForce.peaks_distance) == 3:
            GUI.peak_distance_cell4, GUI.peak_distance_cell5, GUI.peak_distance_cell6 =
RecordForce.peaks_distance[0],RecordForce.peaks_distance[1],RecordForce.peaks_distance[2]
        elif len(RecordForce.peaks_distance) == 4:
            GUI.peak_distance_cell4, GUI.peak_distance_cell5, GUI.peak_distance_cell6 =
RecordForce.peaks_distance[1],RecordForce.peaks_distance[2],RecordForce.peaks_distance[3]
        if len(RecordForce.peaks_time) == 3:
            GUI.peak_time_cell4, GUI.peak_time_cell5,
GUI.peak_time_cell6=RecordForce.peaks_time[0],RecordForce.peaks_time[1],RecordForce.peaks_ti
me[2]
        elif len(RecordForce.peaks_time) == 4:
            GUI.peak_time_cell4, GUI.peak_time_cell5,
GUI.peak_time_cell6=RecordForce.peaks_time[1],RecordForce.peaks_time[2],RecordForce.peaks_ti
me[3]
        #GUI.peak_distance_cell4, GUI.peak_distance_cell5, GUI.peak_distance_cell6 =
RecordForce.peaks_distance[0],RecordForce.peaks_distance[1],RecordForce.peaks_distance[2]
        #GUI.peak_time_cell4, GUI.peak_time_cell5,
GUI.peak_time_cell6=RecordForce.peaks_time[0],RecordForce.peaks_time[1],RecordForce.peaks_ti
me[2]
    elif GUI.currentdirection.get() == "side3":
        if len(RecordForce.peaks_force) == 3:
            GUI.peak_force_cell7, GUI.peak_force_cell8, GUI.peak_force_cell9 =
RecordForce.peaks_force[0],RecordForce.peaks_force[1],RecordForce.peaks_force[2]
        elif len(RecordForce.peaks_force) == 4:
            GUI.peak_force_cell7, GUI.peak_force_cell8, GUI.peak_force_cell9 =
RecordForce.peaks_force[1],RecordForce.peaks_force[2],RecordForce.peaks_force[3]
        if len(RecordForce.peaks_distance) == 3:
            GUI.peak_distance_cell7, GUI.peak_distance_cell8, GUI.peak_distance_cell9 =
RecordForce.peaks_distance[0],RecordForce.peaks_distance[1],RecordForce.peaks_distance[2]
        elif len(RecordForce.peaks_distance) == 4:
            GUI.peak_distance_cell7, GUI.peak_distance_cell8, GUI.peak_distance_cell9 =
RecordForce.peaks_distance[1],RecordForce.peaks_distance[2],RecordForce.peaks_distance[3]
        if len(RecordForce.peaks_time) == 3:
            GUI.peak_time_cell7, GUI.peak_time_cell8,
GUI.peak_time_cell9=RecordForce.peaks_time[0],RecordForce.peaks_time[1],RecordForce.peaks_ti
me[2]
        elif len(RecordForce.peaks_time) == 4:
```

```
            GUI.peak_time_cell7, GUI.peak_time_cell8,
GUI.peak_time_cell9=RecordForce.peaks_time[1],RecordForce.peaks_time[2],RecordForce.peaks_ti
me[3]
        #GUI.peak_distance_cell7, GUI.peak_distance_cell8, GUI.peak_distance_cell9 =
RecordForce.peaks_distance[0],RecordForce.peaks_distance[1],RecordForce.peaks_distance[2]
        #GUI.peak_time_cell7, GUI.peak_time_cell8,
GUI.peak_time_cell9=RecordForce.peaks_time[0],RecordForce.peaks_time[1],RecordForce.peaks_ti
me[2]

  #zeroes load cell measurement

  def tare():
    if GUI.ignoreserial == False:
      print("Tare")
      RecordForce.ser.flush()#wait until all data is written

      tare = 't'
      RecordForce.ser.write(tare.encode()) #sends 't' to arduino, telling it to tare
      time.sleep(0.3)#wait x seconds for Arduino to tare load cell (for smoothing)
    else:
      print("\nYou hit the 'tare' button while GUI.ignoreserial == True.\nLoadcell cannot be tared
because it is neither connected nor sought.")
      RecordForce.message_connectArduino()

  def message_connectArduino():
    #print("\nYou hit the 'tare' button while GUI.ignoreserial == True.\nLoadcell cannot be tared
because it is neither connected nor sought.\n\nConnect an arduino.\nFlash Ardunio with
serialConnection_v11.ino(&+).\n\nIn StemBerry header variables:\nGUI.ignoreserial =
False.\nMatch dev_manual port ID with ID on Arduino IDE.\n\nSigned, Clayton Bennett, August 25,
2022.")
    print("\n\nConnect an arduino.\nFlash Ardunio with serialConnection_v11.ino(&+).\n\nIn
StemBerry header variables:\nGUI.ignoreserial = False.\nMatch dev_manual port ID with ID on
Arduino IDE.\n\nSigned, Clayton Bennett, August 25, 2022.")

  def on_show_frame_RecordForce(self, event):
    #Flip to data collection screen, GUI variables
    if (GUI.varietyname.get()!="" or GUI.plotname.get()!="") and
(GUI.passfillednames_checkbox.get()==1): # checks if a varietyname or plotname has been given
      RecordForce.nameFresh(GUI.varietyname.get(),GUI.plotname.get()) # if so, autopopulate the
basic filestructure
    filename_force = nameBlackBox("",GUI.filename_force.get())
    GUI.filename_force.set(filename_force)
    GUI.currentdirection.set("") # so that sortClicks will funtion properly, if a new name is assigned
# this is non-deal coding

class StemCountClassic(tk.Frame):
  def __init__(self, parent, controller): # automatically runs
```

```
        tk.Frame.__init__(self, parent)

        header_label = Label(self, text = "STEM COUNT INITIAL INPUT", font = ("arial", 17, "bold"), fg =
"gray3", bg="ghost white")
        construction_label = Label(self, text = "Under Construction.\nWill allow user to input sample
density data before pushing SOCEM,\nrather than use the nine-cell post test count input fields.",
font = ("arial", 17, "bold"), fg = "red4", bg="ghost white")
        header_label.place(x=235,y=0)
        construction_label.place(x=10,y=100)

        pageButtons = repeatPageButtons.showButtons(self, parent, controller)

# Load cell calibration page
class Calibrate(tk.Frame):

    def __init__(self, parent, controller): # automatically runs

        tk.Frame.__init__(self, parent)

        ''' GUI design, non-frame '''
        pageButtons = repeatPageButtons.showButtons(self, parent, controller)
        header_label = Label(self, text = "FORCE SENSOR CALIBRATION", font = ("arial", 17, "bold"), fg =
"gray3", bg="ghost white")
        tareIt_label = Label(self, text = "1. Tare w/ no weight", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white")
        inputWeight_label = Label(self, text = "2. Input weight (kg)", font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white")
        caliIt_label = Label(self, text = '3. Place weight', font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white")
        caliIt4_label = Label(self, text = '4. Optimize so Diff. = 0', font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white")
        testWeight_label = Label(self, text = "Weight:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white")

        header_label.place(x=235,y=0)
        tareIt_label.place(x=5,y=43)
        inputWeight_label.place(x=5,y=73)
        caliIt_label.place(x=5,y=103)
        caliIt4_label.place(x=5,y=133)
        testWeight_label.place(x=5,y=183)


        self.knownWeight = DoubleVar() # know weight textvariable
        self.knownWeight.set(0.0) # initially = 1.0 kg (assuming 1.0 kg will be used)
        knownW_entry = Entry(self, textvariable=self.knownWeight, font = ("arial", 14, "bold"), width=
5, bg="white", fg="gray1").place(x = 80, y =183)
```

```
        kg = Label(self, text = "kg", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost
white").place(x=140,y=183)

        self.force = self.knownWeight.get() * convert_KgToN # convert known weight kg to N
        self.strWeight = str('%.3f' % self.force) # store as string
        self.strForce = StringVar() # for displaying & updating on GUI
        self.strForce.set(self.strWeight) # initial value = self.knownWeight

        eq_label = Label(self, text = '= ', font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white")
        force_label = Label(self, textvariable = self.strForce, font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white")
        unit_label = Label(self, text = " N", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white")
        cali_label = Label(self, text = "Cali. Factor:", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost
white")

        eq_label.place(x=170,y=183)
        force_label.place(x=187,y=183)
        unit_label.place(x=249,y=183)
        cali_label.place(x=5,y=223)

        self.calibra =  DoubleVar()
        #self.calibra.set(199750) # initial calibration num. Has been working well. AB.
        self.calibra.set(calibrationFactor) # initial calibration num. Has been working well. AB.
        #self.calibra.set(1997500) # death to the infidels. CB.
        self.factor = self.calibra.get()
        self.calibra_entry = Entry(self, textvariable=self.calibra, font = ("arial", 14, "bold"), width= 10,
bg="white", fg="gray1")


        #tares/zeros load cell
        tare_button = Button(self, text = "Tare", font = ("arial", 16, "bold"), height = 3, width = 8, fg =
"ghost white", bg = "gray2",command=lambda:RecordForce.tare) # confirm this works
        # updates cali factor & starts/continues cali. process
        cali_button = Button(self, text ="Update\nCali.\nFactor", font = ("arial", 16, "bold"), height = 3,
width = 8, fg = "ghost white", bg = "gray2", command=lambda:self.caliThread())
        # stops cali. process
        done_button = Button(self, text ="Done", font = ("arial", 16, "bold"), height = 3, width = 8, fg =
"ghost white", bg = "gray2", command=lambda:self.doneCali())
        # + 1000 to calibra
        p1000_button = Button(self, text ="+1000", font = ("arial", 16, "bold"), height = 1, width = 8, fg =
"ghost white", bg = "gray2", command=lambda:self.updateCali(1000))
        # - 1000 to calibra
        n1000_button = Button(self, text ="-1000", font = ("arial", 16, "bold"), height = 1, width = 8, fg =
"ghost white", bg = "gray2", command=lambda:self.updateCali(-1000))
        # + 100
        p100_button = Button(self, text ="+100", font = ("arial", 16, "bold"), height = 1, width = 8, fg =
"ghost white", bg = "gray2", command=lambda:self.updateCali(100))
        # - 100
```

```
        n100_button = Button(self, text ="-100", font = ("arial", 16, "bold"), height = 1, width = 8, fg =
"ghost white", bg = "gray2", command=lambda:self.updateCali(-100))

        scroll = Scrollbar(self)

        self.LC_label = Label(self, text = "N",font = ("arial", 14, "bold"), fg = "dodgerblue3", bg = "ghost
white")
        self.LClist = Listbox(self, yscrollcommand = scroll.set, bg = "ghost white",highlightbackground =
"gray2", width = 7, height = 10, font = ("arial", 14, "bold"), fg = "dodgerblue3")
        self.Diff_label = Label(self, text = "Diff.",font = ("arial", 14, "bold"), fg = "dodgerblue3", bg =
"ghost white")
        self.Difflist = Listbox(self, yscrollcommand = scroll.set, bg = "ghost white",highlightbackground =
"gray2", width = 7, height = 10, font = ("arial", 14, "bold"), fg = "dodgerblue3")

        self.calibra_entry.place(x = 125, y = 223)
        tare_button.place(x = 559, y = 44)
        cali_button.place(x = 675, y = 44)
        done_button.place(x = 675, y = 224)
        p1000_button.place(x = 559, y = 136)
        n1000_button.place(x = 559, y = 136+44)
        p100_button.place(x = 675, y = 136)
        n100_button.place(x = 675, y = 136+44)

        self.LC_label.place(x = 330, y = 43)
        self.LClist.place(x = 310, y = 73)
        self.Diff_label.place(x = 420, y = 43)
        self.Difflist.place(x = 400, y = 73)

    def updateCali(self, cali): # update calibration factor
        self.factor = self.calibra.get() + cali
        self.calibra_entry.delete(0, 'end')
        self.calibra_entry.insert(0, self.factor)
        return self.factor

    def tare(self):
        RecordForce.ser.flush()#wait until all data is written
        tare = 't'
        RecordForce.ser.write(tare.encode()) #sends 't' to arduino, telling it to tare
        print("Tare.")
        time.sleep(0.3)#wait x seconds for Arduino to tare load cell (for smoothing)

    def caliFactor(self):
        self.force = self.knownWeight.get() * convert_KgToN # convert known weight kg to N
        self.strW = str('%.3f' % self.force) # store as string
        self.strForce.set(self.strW) # update GUI text

        scroll = Scrollbar(self)
        self.factor = self.calibra.get() # get user input calibration factor
```

```
        self.doneCali() # if Arduino sending force data, this will momentarily stop it

        strFactor = str(self.factor) # cali factor as string
        RecordForce.ser.write(strFactor.encode()) # send cali factor to Arduino
        RecordForce.ser.flush() # make sure it gets it before proceeding

        global caliLoop
        caliLoop = True

        while caliLoop == True: # loop to continuously print Arduino force readings

            if RecordForce.ser.inWaiting() > 0: #checks to see if Serial is available

                try: #make sure serial data can be read/is there
                    ser_bytes = RecordForce.ser.readline()
                except:
                    GUI.errors.append('serial read')
                    eCode = 'e8'
                    GUI.errorCodes.append(eCode)
                    print("eCode = "+eCode)
                    #popup("serial read")

                bytesDecoded = (ser_bytes[0:len(ser_bytes)-2].decode("utf-8")) # force reading bytes
                try:
                    reading = float(bytesDecoded) # convert bytes to float
                    diff = self.force - reading # difference between reading & known weight
                    self.LClist.insert(END, str('%.2f' % reading)) # scrollbar list for force readings
                    self.Difflist.see(END)
                    self.Difflist.insert(END, str('%.1f' % diff)) # scrollbar list for forcebar - known weight
                    self.LClist.see(END)
                except:
                    pass


    def caliThread(self): #threading calibrate function (simultaneously performs caliFactor function in
backend)
        thread = threading.Thread(target = Calibrate.caliFactor,args=(self,))
        thread.start()

    def doneCali(self): # stops calibration process
        # RecordForce.ser.reset_input_buffer()# clear the input buffer # suppressed 9/6/22 CB
        global caliLoop
        caliLoop = False # stop loop asking for data

        send = 'd' # stop Arduino sending
        RecordForce.ser.write(send.encode()) # send 'd' to stop Arduino sending data

# error page for displaying errors
```

```
class ErrorReport(tk.Frame):

   def __init__(self, parent, controller): # automatically runs
      tk.Frame.__init__(self, parent)

      # button that returns to Geo. Inputs page
      initialInputs_button = Button(self, text ="Initial\nInputs", font = ("arial", 16, "bold"), height = 3,
width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(InitialInputs))
      initialInputs_button.place(x = 675, y = 316)
      # button that returns to RecordForce page
      recordForce_button = Button(self, text = "Record\nForce",font = ("arial", 16, "bold"), height = 3,
width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(RecordForce))
      recordForce_button.place(x = 675, y = 225)

      scroll = Scrollbar(self)

      self.ErrorCode_label = Label(self, text = "Error Code\n(Location)",font = ("arial", 14, "bold"), fg =
"gray3", bg = "ghost white").place(x = 179, y = 50)
      self.ErrorCodeList = Listbox(self, yscrollcommand = scroll.set, bg = "ghost
white",highlightbackground = "gray2", width = 10, height = 13, font = ("arial", 14, "bold"), fg =
"dodgerblue3")
      self.ErrorCodeList.place(x = 175, y = 100)

      self.Error_label = Label(self, text = "Description",font = ("arial", 14, "bold"), fg = "gray3", bg =
"ghost white")
      self.Error_label.place(x = 400, y = 75)
      self.ErrorDesc = Listbox(self, yscrollcommand = scroll.set, bg = "ghost
white",highlightbackground = "gray2", width = 30, height = 13, font = ("arial", 14, "bold"), fg =
"dodgerblue3")
      self.ErrorDesc.place(x = 289, y = 100)

   def showErrors2(self):

      self.ErrorCodeList.delete(0, 'end')
      self.ErrorDesc.delete(0, 'end')

      for e in range(len(GUI.errorCodes)):
         self.ErrorCodeList.insert(END, GUI.errorCodes[e])# inserts at end of listbox to actually display
         self.ErrorCodeList.see(END)# makes sure listbox is at end so it displays live data
         self.ErrorDesc.insert(END, GUI.errors[e])
         self.ErrorDesc.see(END)
'''
class Heights(tk.Frame):
   destroyed. see StemBerry_v13.
'''
 # Guide page
class Guide(tk.Frame):
   def __init__(self, parent, controller): # automatically runs
```

```
        tk.Frame.__init__(self, parent)

        pageButtons = repeatPageButtons.showButtons(self, parent, controller)

        # button that enters Calibrate page/class
        calibrate_button = Button(self, text = "Calibrate\nForce\nSensor", font = ("arial", 16, "bold"),
height = 3, width = 8, fg = "ghost white", bg = "gray2",
command=lambda:GUI.show_frame(Calibrate)) #tares/zeros load cell
        calibrate_button.place(x = 510, y = 340)

        # instruction steps:
        '''
        Nine-cell scheme design:
        '''
        guide_frame = tk.LabelFrame(self, text='Nine-Cell Scheme',font = ("arial", 14, "bold"), width=
10, bg="white", fg="gray1")
        guide_frame.place(x = 0, y = 20)
        #guideHeader = Label(self, text = "Nine-cell scheme design", font = ("arial", 17, "bold"), fg =
"gray3", bg="ghost white").place(x=350,y=0)
        one = Label(guide_frame, text = '1.  Equalize stem heights', font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white").grid(row=0, column=0)
        two = Label(guide_frame, text = '2.  Record Variety and Plot names', font = ("arial", 14, "bold"),
fg = "gray3", bg="ghost white").grid(row=1, column=0)
        three = Label(guide_frame, text = '3.  Enter stem height and cell location data', font = ("arial",
14, "bold"), fg = "gray3", bg="ghost white").grid(row=2, column=0)
        four = Label(guide_frame, text = '4.  Perform four SOCEM tests (3 side hits, 1 forward hit)', font
= ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=3, column=0)
        five = Label(guide_frame, text = '5.  Collect stems for mass, count, and diameters.', font =
("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=4, column=0)
        six = Label(guide_frame, text = '6.  Press compile to complete nine-cell data object.\nGo on to
the next small plot!', font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=5,
column=0)

        try:
            # SOCEM diagram of use #
            load = Image.open(directory+'/'+'GuideSOCEM_2022.png')
            load = load.resize((275,275))
            render = ImageTk.PhotoImage(load)
            img = Label(self, image=render)
            img.image = render
            img.place(x = 520, y = 35)
        except:
            print("Guide image not found.")

class FinalInputs(tk.Frame):

    def __init__(self, parent, controller): # automatically runs
```

```
        FinalInputs.mass1 = [] # TypeError: 'float' object is not iterable
        FinalInputs.mass2 = []
        FinalInputs.mass3 = []
        FinalInputs.mass4 = []
        FinalInputs.mass5 = []
        FinalInputs.mass6 = []
        FinalInputs.mass7 = []
        FinalInputs.mass8 = []
        FinalInputs.mass9 = []
        FinalInputs.count1 = []
        FinalInputs.count2 = []
        FinalInputs.count3 = []
        FinalInputs.count4 = []
        FinalInputs.count5 = []
        FinalInputs.count6 = []
        FinalInputs.count7 = []
        FinalInputs.count8 = []
        FinalInputs.count9 = []

        FinalInputs.diam1 = []
        FinalInputs.diam2 = []
        FinalInputs.diam3 = []
        FinalInputs.diam4 = []
        FinalInputs.diam5 = []
        FinalInputs.diam6 = []
        FinalInputs.diam7 = []
        FinalInputs.diam8 = []
        FinalInputs.diam9 = []

        tk.Frame.__init__(self, parent)

        ''' GUI design, non-frame '''
        pageButtons = repeatPageButtons.showButtons(self, parent, controller)
        unit_label = Label(self, text=str("Mass unit is grams, diameter unit is millimeters."), font =
("arial", 12, "italic"), fg = "red4", bg="ghost white")
        #backupFinalInputs_button = Button(self, text ="Create Backup File", font = ("arial", 14, "bold"),
height = 1, width = 20, fg = "ghost white", bg = "dodgerblue3",
command=lambda:createBackupFile())
        savePostTestInputs_button = Button(self, text ="Save Post Test Inputs", font = ("arial", 14,
"bold"), height = 1, width = 20, fg = "ghost white", bg = "dodgerblue3",
command=lambda:self.savePostTestInputs())
        compileNineCellData_button = Button(self, text ="Compile Nine-Cell Data", font = ("arial", 14,
"bold"), height = 1, width = 20, fg = "ghost white", bg = "dodgerblue3",
command=lambda:self.compileNineCellData())

        unit_label.place(x=400+30,y=0)
        #backupFinalInputs_button.place(x = 510, y = 340+38)
        compileNineCellData_button.place(x = 510, y = 340+38)
```

```
        savePostTestInputs_button.place(x = 510, y = 340)

        ''' Frame: Cell 1 '''
        cell1_frame = tk.LabelFrame(self, text='Cell 1',font = ("arial", 14, "bold"), width= 10, bg="white",
fg="gray1")
        cell1_frame.place(x = 0, y = 230)
        cell1Mass_label = Label(cell1_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=0, column=0)
        cell1Count_label = Label(cell1_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=1, column=0)
        cell1Diameters_label = Label(cell1_frame, text = "Diam:", font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white").grid(row=2, column=0)
        cell1Mass_entry = Entry(cell1_frame, textvariable=GUI.cell1Mass, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell1Count_entry = Entry(cell1_frame, textvariable=GUI.cell1Count, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=1, column=1)
        cell1Diameter1_entry = Entry(cell1_frame, textvariable=GUI.cell1Diameter1, font = ("arial", 14,
"bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
        cell1Diameter2_entry = Entry(cell1_frame, textvariable=GUI.cell1Diameter2, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
        cell1Diameter3_entry = Entry(cell1_frame, textvariable=GUI.cell1Diameter3, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
        cell1Diameter4_entry = Entry(cell1_frame, textvariable=GUI.cell1Diameter4, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        ''' Frame: Cell 2 '''
        cell2_frame = tk.LabelFrame(self, text='Cell 2',font = ("arial", 14, "bold"), width= 10, bg="white",
fg="gray1")
        #cell2_frame.place(x = 250, y = 230)
        cell2_frame.place(x = 0, y = 125)
        cell2Mass_label = Label(cell2_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=0, column=0)
        cell2Count_label = Label(cell2_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=1, column=0)
        cell2Diameters_label = Label(cell2_frame, text = "Diam:", font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white").grid(row=2, column=0)
        cell2Mass_entry = Entry(cell2_frame, textvariable=GUI.cell2Mass, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell2Count_entry = Entry(cell2_frame, textvariable=GUI.cell2Count, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=1, column=1)
        cell2Diameter1_entry = Entry(cell2_frame, textvariable=GUI.cell2Diameter1, font = ("arial", 14,
"bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
        cell2Diameter2_entry = Entry(cell2_frame, textvariable=GUI.cell2Diameter2, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
        cell2Diameter3_entry = Entry(cell2_frame, textvariable=GUI.cell2Diameter3, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
```

```
        cell2Diameter4_entry = Entry(cell2_frame, textvariable=GUI.cell2Diameter4, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        ''' Frame: Cell 3 '''
        cell3_frame = tk.LabelFrame(self, text='Cell 3',font = ("arial", 14, "bold"), width= 10, bg="white",
fg="gray1")
        #cell3_frame.place(x = 500, y = 230)
        cell3_frame.place(x = 0, y = 20)
        cell3Mass_label = Label(cell3_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=0, column=0)
        cell3Count_label = Label(cell3_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=1, column=0)
        cell3Diameters_label = Label(cell3_frame, text = "Diam:", font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white").grid(row=2, column=0)
        cell3Mass_entry = Entry(cell3_frame, textvariable=GUI.cell3Mass, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell3Count_entry = Entry(cell3_frame, textvariable=GUI.cell3Count, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=1, column=1)
        cell3Diameter1_entry = Entry(cell3_frame, textvariable=GUI.cell3Diameter1, font = ("arial", 14,
"bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
        cell3Diameter2_entry = Entry(cell3_frame, textvariable=GUI.cell3Diameter2, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
        cell3Diameter3_entry = Entry(cell3_frame, textvariable=GUI.cell3Diameter3, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
        cell3Diameter4_entry = Entry(cell3_frame, textvariable=GUI.cell3Diameter4, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        ''' Frame: Cell 4 '''
        cell4_frame = tk.LabelFrame(self, text='Cell 4',font = ("arial", 14, "bold"), width= 10, bg="white",
fg="gray1")
        #cell4_frame.place(x = 0, y = 125)
        cell4_frame.place(x = 250, y = 230)
        cell4Mass_label = Label(cell4_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=0, column=0)
        cell4Count_label = Label(cell4_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=1, column=0)
        cell4Diameters_label = Label(cell4_frame, text = "Diam:", font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white").grid(row=2, column=0)
        cell4Mass_entry = Entry(cell4_frame, textvariable=GUI.cell4Mass, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell4Count_entry = Entry(cell4_frame, textvariable=GUI.cell4Count, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=1, column=1)
        cell4Diameter1_entry = Entry(cell4_frame, textvariable=GUI.cell4Diameter1, font = ("arial", 14,
"bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
        cell4Diameter2_entry = Entry(cell4_frame, textvariable=GUI.cell4Diameter2, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
```

```
        cell4Diameter3_entry = Entry(cell4_frame, textvariable=GUI.cell4Diameter3, font = ("arial", 14, "bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
        cell4Diameter4_entry = Entry(cell4_frame, textvariable=GUI.cell4Diameter4, font = ("arial", 14, "bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        ''' Frame: Cell 5 '''
        cell5_frame = tk.LabelFrame(self, text='Cell 5',font = ("arial", 14, "bold"), width= 10, bg="white", fg="gray1")
        cell5_frame.place(x = 250, y = 125)
        cell5Mass_label = Label(cell5_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=0, column=0)
        cell5Count_label = Label(cell5_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=1, column=0)
        cell5Diameters_label = Label(cell5_frame, text = "Diam:", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=2, column=0)
        cell5Mass_entry = Entry(cell5_frame, textvariable=GUI.cell5Mass, font = ("arial", 14, "bold"), width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell5Count_entry = Entry(cell5_frame, textvariable=GUI.cell5Count, font = ("arial", 14, "bold"), width=4, bg="white", fg="gray1").grid(row=1, column=1)
        cell5Diameter1_entry = Entry(cell5_frame, textvariable=GUI.cell5Diameter1, font = ("arial", 14, "bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
        cell5Diameter2_entry = Entry(cell5_frame, textvariable=GUI.cell5Diameter2, font = ("arial", 14, "bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
        cell5Diameter3_entry = Entry(cell5_frame, textvariable=GUI.cell5Diameter3, font = ("arial", 14, "bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
        cell5Diameter4_entry = Entry(cell5_frame, textvariable=GUI.cell5Diameter4, font = ("arial", 14, "bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        ''' Frame: Cell 6 '''
        cell6_frame = tk.LabelFrame(self, text='Cell 6',font = ("arial", 14, "bold"), width= 10, bg="white", fg="gray1")
        #cell6_frame.place(x = 500, y = 125)
        cell6_frame.place(x = 250, y = 20)
        cell6Mass_label = Label(cell6_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=0, column=0)
        cell6Count_label = Label(cell6_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=1, column=0)
        cell6Diameters_label = Label(cell6_frame, text = "Diam:", font = ("arial", 14, "bold"), fg = "gray3", bg="ghost white").grid(row=2, column=0)
        cell6Mass_entry = Entry(cell6_frame, textvariable=GUI.cell6Mass, font = ("arial", 14, "bold"), width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell6Count_entry = Entry(cell6_frame, textvariable=GUI.cell6Count, font = ("arial", 14, "bold"), width=4, bg="white", fg="gray1").grid(row=1, column=1)
        cell6Diameter1_entry = Entry(cell6_frame, textvariable=GUI.cell6Diameter1, font = ("arial", 14, "bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
```

```
        cell6Diameter2_entry = Entry(cell6_frame, textvariable=GUI.cell6Diameter2, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
        cell6Diameter3_entry = Entry(cell6_frame, textvariable=GUI.cell6Diameter3, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
        cell6Diameter4_entry = Entry(cell6_frame, textvariable=GUI.cell6Diameter4, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        ''' Frame: Cell 7 '''
        cell7_frame = tk.LabelFrame(self, text='Cell 7',font = ("arial", 14, "bold"), width= 10, bg="white",
fg="gray1")
        #cell7_frame.place(x = 0, y = 20)
        cell7_frame.place(x = 500, y = 230)
        cell7Mass_label = Label(cell7_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=0, column=0)
        cell7Count_label = Label(cell7_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=1, column=0)
        cell7Diameters_label = Label(cell7_frame, text = "Diam:", font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white").grid(row=2, column=0)
        cell7Mass_entry = Entry(cell7_frame, textvariable=GUI.cell7Mass, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell7Count_entry = Entry(cell7_frame, textvariable=GUI.cell7Count, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=1, column=1)
        cell7Diameter1_entry = Entry(cell7_frame, textvariable=GUI.cell7Diameter1, font = ("arial", 14,
"bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
        cell7Diameter2_entry = Entry(cell7_frame, textvariable=GUI.cell7Diameter2, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
        cell7Diameter3_entry = Entry(cell7_frame, textvariable=GUI.cell7Diameter3, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
        cell7Diameter4_entry = Entry(cell7_frame, textvariable=GUI.cell7Diameter4, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        ''' Frame: Cell 8 '''
        cell8_frame = tk.LabelFrame(self, text='Cell 8',font = ("arial", 14, "bold"), width= 10, bg="white",
fg="gray1")
        #cell8_frame.place(x = 250, y = 20)
        cell8_frame.place(x = 500, y = 125)
        cell8Mass_label = Label(cell8_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=0, column=0)
        cell8Count_label = Label(cell8_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=1, column=0)
        cell8Diameters_label = Label(cell8_frame, text = "Diam:", font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white").grid(row=2, column=0)
        cell8Mass_entry = Entry(cell8_frame, textvariable=GUI.cell8Mass, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell8Count_entry = Entry(cell8_frame, textvariable=GUI.cell8Count, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=1, column=1)
```

```
        cell8Diameter1_entry = Entry(cell8_frame, textvariable=GUI.cell8Diameter1, font = ("arial", 14,
"bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
        cell8Diameter2_entry = Entry(cell8_frame, textvariable=GUI.cell8Diameter2, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
        cell8Diameter3_entry = Entry(cell8_frame, textvariable=GUI.cell8Diameter3, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
        cell8Diameter4_entry = Entry(cell8_frame, textvariable=GUI.cell8Diameter4, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        ''' Frame: Cell 9 '''
        cell9_frame = tk.LabelFrame(self, text='Cell 9',font = ("arial", 14, "bold"), width= 10, bg="white",
fg="gray1")
        cell9_frame.place(x = 500, y = 20)
        cell9Mass_label = Label(cell9_frame, text = "Mass:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=0, column=0)
        cell9Count_label = Label(cell9_frame, text = "Count:", font = ("arial", 14, "bold"), fg = "gray3",
bg="ghost white").grid(row=1, column=0)
        cell9Diameters_label = Label(cell9_frame, text = "Diam:", font = ("arial", 14, "bold"), fg =
"gray3", bg="ghost white").grid(row=2, column=0)
        cell9Mass_entry = Entry(cell9_frame, textvariable=GUI.cell9Mass, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=0, column=1)
        cell9Count_entry = Entry(cell9_frame, textvariable=GUI.cell9Count, font = ("arial", 14, "bold"),
width=4, bg="white", fg="gray1").grid(row=1, column=1)
        cell9Diameter1_entry = Entry(cell9_frame, textvariable=GUI.cell9Diameter1, font = ("arial", 14,
"bold"), width=4, bg="white", fg="gray1").grid(row=2, column=1)
        cell9Diameter2_entry = Entry(cell9_frame, textvariable=GUI.cell9Diameter2, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=2)
        cell9Diameter3_entry = Entry(cell9_frame, textvariable=GUI.cell9Diameter3, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=3)
        cell9Diameter4_entry = Entry(cell9_frame, textvariable=GUI.cell9Diameter4, font = ("arial", 14,
"bold"), width=3, bg="white", fg="gray1").grid(row=2, column=4)
        ''' end '''

        self.bind("<<ShowFrame>>", self.on_show_frame_FinalInputs) # need this?

    def savePostTestInputs(self):

        filename_postTest_csv = GUI.address + '/' + (GUI.filename_postTest.get()) + '.csv'

        FinalInputs.mass1 = [GUI.cell1Mass.get()] # TypeError: 'float' object is not iterable
        FinalInputs.mass2 = [GUI.cell2Mass.get()]
        FinalInputs.mass3 = [GUI.cell3Mass.get()]
        FinalInputs.mass4 = [GUI.cell4Mass.get()]
        FinalInputs.mass5 = [GUI.cell5Mass.get()]
        FinalInputs.mass6 = [GUI.cell6Mass.get()]
        FinalInputs.mass7 = [GUI.cell7Mass.get()]
        FinalInputs.mass8 = [GUI.cell8Mass.get()]
```

```
        FinalInputs.mass9 = [GUI.cell9Mass.get()]
        FinalInputs.count1 = [GUI.cell1Count.get()] # TypeError: 'float' object is not iterable
        FinalInputs.count2 = [GUI.cell2Count.get()]
        FinalInputs.count3 = [GUI.cell3Count.get()]
        FinalInputs.count4 = [GUI.cell4Count.get()]
        FinalInputs.count5 = [GUI.cell5Count.get()]
        FinalInputs.count6 = [GUI.cell6Count.get()]
        FinalInputs.count7 = [GUI.cell7Count.get()]
        FinalInputs.count8 = [GUI.cell8Count.get()]
        FinalInputs.count9 = [GUI.cell9Count.get()]

        FinalInputs.diam1 =
[GUI.cell1Diameter1.get(),GUI.cell1Diameter2.get(),GUI.cell1Diameter3.get(),GUI.cell1Diameter4.ge
t()]
        FinalInputs.diam2 =
[GUI.cell2Diameter1.get(),GUI.cell2Diameter2.get(),GUI.cell2Diameter3.get(),GUI.cell2Diameter4.ge
t()]
        FinalInputs.diam3 =
[GUI.cell3Diameter1.get(),GUI.cell3Diameter2.get(),GUI.cell3Diameter3.get(),GUI.cell3Diameter4.ge
t()]
        FinalInputs.diam4 =
[GUI.cell4Diameter1.get(),GUI.cell4Diameter2.get(),GUI.cell4Diameter3.get(),GUI.cell4Diameter4.ge
t()]
        FinalInputs.diam5 =
[GUI.cell5Diameter1.get(),GUI.cell5Diameter2.get(),GUI.cell5Diameter3.get(),GUI.cell5Diameter4.ge
t()]
        FinalInputs.diam6 =
[GUI.cell6Diameter1.get(),GUI.cell6Diameter2.get(),GUI.cell6Diameter3.get(),GUI.cell6Diameter4.ge
t()]
        FinalInputs.diam7 =
[GUI.cell7Diameter1.get(),GUI.cell7Diameter2.get(),GUI.cell7Diameter3.get(),GUI.cell7Diameter4.ge
t()]
        FinalInputs.diam8 =
[GUI.cell8Diameter1.get(),GUI.cell8Diameter2.get(),GUI.cell8Diameter3.get(),GUI.cell8Diameter4.ge
t()]
        FinalInputs.diam9 =
[GUI.cell9Diameter1.get(),GUI.cell9Diameter2.get(),GUI.cell9Diameter3.get(),GUI.cell9Diameter4.ge
t()]

         # Labels for Excel
        FinalInputs.diam1.insert(0,"diameters_cell1(mm)")
        FinalInputs.diam2.insert(0,"diameters_cell2(mm)")
        FinalInputs.diam3.insert(0,"diameters_cell3(mm)")
        FinalInputs.diam4.insert(0,"diameters_cell4(mm)")
        FinalInputs.diam5.insert(0,"diameters_cell5(mm)")
        FinalInputs.diam6.insert(0,"diameters_cell6(mm)")
        FinalInputs.diam7.insert(0,"diameters_cell7(mm)")
        FinalInputs.diam8.insert(0,"diameters_cell8(mm)")
```

```
        FinalInputs.diam9.insert(0,"diameters_cell9(mm)")
        FinalInputs.mass1.insert(0,"mass_cell1(g)")
        FinalInputs.mass2.insert(0,"mass_cell2(g)")
        FinalInputs.mass3.insert(0,"mass_cell3(g)")
        FinalInputs.mass4.insert(0,"mass_cell4(g)")
        FinalInputs.mass5.insert(0,"mass_cell5(g)")
        FinalInputs.mass6.insert(0,"mass_cell6(g)")
        FinalInputs.mass7.insert(0,"mass_cell7(g)")
        FinalInputs.mass8.insert(0,"mass_cell8(g)")
        FinalInputs.mass9.insert(0,"mass_cell9(g)")
        FinalInputs.count1.insert(0,"count_cell1")
        FinalInputs.count2.insert(0,"count_cell2")
        FinalInputs.count3.insert(0,"count_cell3")
        FinalInputs.count4.insert(0,"count_cell4")
        FinalInputs.count5.insert(0,"count_cell5")
        FinalInputs.count6.insert(0,"count_cell6")
        FinalInputs.count7.insert(0,"count_cell7")
        FinalInputs.count8.insert(0,"count_cell8")
        FinalInputs.count9.insert(0,"count_cell9")

        if overwriteGuardPage(filename_postTest_csv) == True: # filename already exists, needs to be
renamed
            renamePage(GUI.filename_postTest.get()) # prompt user to rename file
        ''' write CSV'''

        GUI.data_postTest =
[FinalInputs.diam1,FinalInputs.diam2,FinalInputs.diam3,FinalInputs.diam4,FinalInputs.diam5,FinalIn
puts.diam6,FinalInputs.diam7,FinalInputs.diam8,FinalInputs.diam9,FinalInputs.mass1,FinalInputs.m
ass2,FinalInputs.mass3,FinalInputs.mass4,FinalInputs.mass5,FinalInputs.mass6,FinalInputs.mass7,Fi
nalInputs.mass8,FinalInputs.mass9,FinalInputs.count1,FinalInputs.count2,FinalInputs.count3,FinalIn
puts.count4,FinalInputs.count5,FinalInputs.count6,FinalInputs.count7,FinalInputs.count8,FinalInputs
.count9]
        columns_data_postTest = zip_longest(*GUI.data_postTest)

        with open(filename_postTest_csv,'w',newline='') as f:
            writer = csv.writer(f)
            writer.writerows(columns_data_postTest)
        ''' end: write CSV '''
        print("filename_postTest_csv = "+filename_postTest_csv)

    def saveEIs():
        GUI.EI_fullcontact.insert(0 , "EI_fullcontact(N*cm^2)")
        GUI.EI_intermediatecontact.insert(0 , "EI_intermediatecontact(N*cm^2)")
        GUI.EI_nocontact.insert(0 , "EI_nocontact(N*cm^2")
        GUI.AvgEI_intermediatecontact.insert(0 , "AvgEI_intermediatecontact(N*cm^2)")
        ''' write CSV'''
        filename_EI_csv = GUI.address + '/' + GUI.filename_force.get() + '_EI.csv'
```

```
      GUI.data_EI =
[GUI.EI_fullcontact,GUI.EI_intermediatecontact,GUI.EI_nocontact,GUI.AvgEI_intermediatecontact]
      columns_data_EI = zip_longest(*GUI.data_EI)
      with open(filename_EI_csv,'w',newline='') as f:
         writer = csv.writer(f)
         writer.writerows(columns_data_EI)
      ''' end: write CSV '''
      print("filename_EI_csv = "+filename_EI_csv)
      #print("saved:", filename_EI_csv)

   def compileNineCellData(self):
      createBackupFile() # fix below # numbers are for lbs, not newtons
      GUI.peaks_force = [GUI.peak_force_cell1, GUI.peak_force_cell2, GUI.peak_force_cell3,
GUI.peak_force_cell4, GUI.peak_force_cell5, GUI.peak_force_cell6, GUI.peak_force_cell7,
GUI.peak_force_cell8, GUI.peak_force_cell9]
      GUI.peaks_distance = [GUI.peak_distance_cell1, GUI.peak_distance_cell2,
GUI.peak_distance_cell3, GUI.peak_distance_cell4, GUI.peak_distance_cell5,
GUI.peak_distance_cell6, GUI.peak_distance_cell7, GUI.peak_distance_cell8,
GUI.peak_distance_cell9]
      GUI.peaks_time = [GUI.peak_time_cell1, GUI.peak_time_cell2, GUI.peak_time_cell3,
GUI.peak_time_cell4, GUI.peak_time_cell5, GUI.peak_time_cell6, GUI.peak_time_cell7,
GUI.peak_time_cell8, GUI.peak_time_cell9]

GUI.stemcounts=[GUI.cell1Count.get(),GUI.cell2Count.get(),GUI.cell3Count.get(),GUI.cell4Count.get(
),GUI.cell5Count.get(),GUI.cell6Count.get(),GUI.cell7Count.get(),GUI.cell8Count.get(),GUI.cell9Count
.get()]

      # changed from 10 units long to 9 units long, troubleshoot, CB 9/8/22
      #GUI.stemspacing_average, GUI.EI_fullcontact, GUI.EI_nocontact, GUI.EI_intermediatecontact =
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],[0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0],[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
      GUI.stemspacing_average, GUI.EI_fullcontact, GUI.EI_nocontact, GUI.EI_intermediatecontact =
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0],[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
      print("GUI.stemcounts = ",GUI.stemcounts)
      #for i in range(0,9): #1,10
      for i in range(0,8): #1,10 # FIX THIS AFTER DAQ, for EI post processing
         try:
            GUI.stemspacing_average[i], GUI.EI_fullcontact[i], GUI.EI_nocontact[i] =
FinalInputs.calculateEI(float(GUI.peaks_force[i]), GUI.stemheight.get(), GUI.barbottom.get(),
GUI.stemcounts[i])
         except:
            GUI.stemspacing_average[i], GUI.EI_fullcontact[i], GUI.EI_nocontact[i] = 0,0,0
      GUI.AvgEI_intermediatecontact = [0.0]
      GUI.AvgEI_intermediatecontact[0] =
round(sum(GUI.EI_intermediatecontact)/len(GUI.EI_intermediatecontact),3)
      FinalInputs.saveEIs()
      FinalInputs.saveCompiled()
```

```
        time.sleep(1) # pause x seconds
        if refreshAllAuto == True:
            GUI.refreshAll() # refresh all variables
        #except:
        #else:
        #    print("The nine cell scheme requires exactly 9 clips, 3 from each side hit.")



    def calculateEI(peak_force, stemheight, barbottom, stemcount):
        #try:
        stemspacing_average = 1/(stemcount/barlength)
        EI_fullcontact = EI_Interaction_Fx.EI_Interaction(peak_force, stemheight,
barbottom,stemspacing_average) # uses clicked forces (Y axis), force bar height, horizontal plot
heights, and count density
        EI_nocontact = EI_No_Interaction_Fx.EI_NoInteraction(peak_force, stemheight, barbottom,
stemspacing_average) # the x value of the click does nothing other than find the nearest height from
horz. It is not factored in to the number of beams or the character of the beams.
        EI_intermediatecontact = (EI_fullcontact + EI_nocontact)/2
        EI_Interaction_Fx.clearAll()
        EI_No_Interaction_Fx.clearAll()
        #except:
        '''
        stemspacing_average = 0 # if count is zero
        EI_fullcontact = 0
        EI_nocontact = 0
        EI_intermediatecontact = 0
        '''

        return stemspacing_average, EI_fullcontact, EI_nocontact, EI_intermediatecontact;

    def saveCompiled():
        #print("Compiled Data button does not currently work correctly. Please develop.")
        # ned to change the way I handle filename_force : we need a base name, that is variety and plot
        # same for the peakcliclk setion.
        ''' write XSLX'''
        filename_compiled_xlsx = GUI.address + '/' + GUI.filename_force.get() + '_compiled.xlsx'

        filename_preTest_csv = GUI.address + '/' + GUI.filename_preTest.get() + '.csv'
        filename_forceSide1_csv = GUI.address + '/' + (GUI.filename_force.get()) + '_side1.csv'
        filename_forceSide2_csv = GUI.address + '/' + (GUI.filename_force.get()) + '_side2.csv'
        filename_forceSide3_csv = GUI.address + '/' + (GUI.filename_force.get()) + '_side3.csv'
        filename_forceForward_csv = GUI.address + '/' + (GUI.filename_force.get()) + '_forward.csv'
        filename_EI_csv = GUI.address + '/' + GUI.filename_force.get() + '_EI.csv'
        filename_postTest_csv = GUI.address + '/' + (GUI.filename_postTest.get()) + '.csv'
```

```
        filenames_CSV_all =
[filename_preTest_csv,filename_postTest_csv,filename_EI_csv,filename_forceSide1_csv,filename_f
orceSide2_csv,filename_forceSide3_csv,filename_forceForward_csv]
        sheetnames_XLSX_all =
['preTest','postTest','EI','force_side1','force_side2','force_side3','force_forward']
        sheetnames_XLSX_most = ['preTest','postTest','EI','force_side1','force_side2','force_side3']

        ''' test names, sans arduino'''
        filenames_CSV_some = [filename_preTest_csv,filename_postTest_csv,filename_EI_csv]
        sheetnames_XLSX_some = ['preTest','postTest','EI']

        writer = pd.ExcelWriter(filename_compiled_xlsx, engine='xlsxwriter')

        try:
            i=0
            for csvfilename in filenames_CSV_all:
                df = pd.read_csv(csvfilename)
                df.to_excel(writer,sheet_name=sheetnames_XLSX_all[i])
                i+=1
            writer.save()
            print("filename_compiled_xlsx = "+filename_compiled_xlsx)
        except:
            try:
                i=0
                for csvfilename in filenames_CSV_some:
                    df = pd.read_csv(csvfilename)
                    df.to_excel(writer,sheet_name=sheetnames_XLSX_most[i])
                    i+=1
                writer.save()
                print("filename_compiled_xlsx = "+filename_compiled_xlsx)
                print("Forward hit data not included in compilation file.")
            except:
                try:
                    i=0
                    for csvfilename in filenames_CSV_some:
                        df = pd.read_csv(csvfilename)
                        df.to_excel(writer,sheet_name=sheetnames_XLSX_some[i])
                        i+=1
                    writer.save()
                    print("filename_compiled_xlsx = "+filename_compiled_xlsx)
                    print("Raw force data not included in compilation file.")
                except:
                    print("Generate at least a pre-test and post-test CSV file before trying to compile data.")


    def on_show_frame_FinalInputs(self, event):
        #print("Flip to FinalInputs screen.")
```

```
        background_box = Label(self, text="This is hidden text meant to cover up old text.", font =
("arial", 14, "bold"), fg = "ghost white", bg="ghost white")
        background_box.place(x=0,y=0)
        # Update stringname of postTest file, based on filename_force, if it exists.
        filename_postTest = nameBlackBox("postTest",GUI.filename_postTest.get())
        GUI.filename_postTest.set(filename_postTest)

        # update filename text field, even if force page is never activated
        if (GUI.varietyname.get()!="" or GUI.plotname.get()!="") and
(GUI.passfillednames_checkbox.get()==1): # checks if a varietyname or plotname has been given
            RecordForce.nameFresh(GUI.varietyname.get(),GUI.plotname.get()) # if so, autopopulate the
basic filestructure
        filename_force = nameBlackBox("",GUI.filename_force.get())
        GUI.filename_force.set(filename_force)



        filename_label = Label(self, text="Filename:"+filename_postTest, font = ("arial", 14, "bold"), fg
= "dodgerblue3", bg="ghost white")
        filename_label.place(x=0,y=0)

''' Figure Interation classes '''

class SnaptoCursor(object):
    '''
    Cursor crossshair snaps to nearest x, y point.
    '''
    def __init__(self, ax, x, y):
        self.ax = ax
        #self.lx = ax.axhline(color='gold') # horizontal line
        self.ly = ax.axvline(color='orange', linewidth=1, linestyle="--") # vertical line
        self.x = x
        self.y = y
        # text location in axes coords
        self.txt = ax.text(0.7, 0.9, '', transform=ax.transAxes)

    def mouse_move(self, event):
        if not event.inaxes:
            return

        x, y = event.xdata, event.ydata
        indx = min(np.searchsorted(self.x, x), len(self.x)-1)
        x = self.x[indx]
        y = self.y[indx]
        # update the line positions
        #self.lx.set_ydata(y) # why is this commented out
        self.ly.set_xdata(x)

        self.txt.set_text('x=%1.2f, y=%1.2f' % (x, y))
```

```
        #print('x=%1.2f, y=%1.2f' % (x, y))
        self.ax.figure.canvas.draw()

class Cursor(object):
    '''
    Cursor crosshair that follows mouse
    '''

    def __init__(self, ax):
        self.ax = ax
        self.lx = ax.axhline(color='orange', linewidth=1, linestyle="--")
        self.ly = ax.axvline(color='orange', linewidth=1, linestyle="--")
        #text location in axes coords
        self.txt = ax.text(0.7, 0.9, '',transform=ax.transAxes)

    def mouse_move(self, event):
        if not event.inaxes:
            return
        x, y = event.xdata, event.ydata
        #update line positions
        self.lx.set_ydata(y)
        self.ly.set_xdata(x)

        self.txt.set_text('x=%1.2f, y=%1.2f' % (x, y))
        self.ax.figure.canvas.draw()

''' Peak click plotter methods'''
def initialPlot(distanceTraveled, forcePushed, timeElapsed, encoderWorked,
varietyAndPlotnameAndDetail, documentationFolder,averageVelocity):
    fig, ax = plt.subplots()
    encoderWorked = encoderWorked_override
    ''' vertical lines, for suggesting edge effect regions for forward tests '''
    if encoderWorked == True:
        start = 50 # cm , cut off 1st 50cm = 20" usually #
        end = 305 # cm, cut off after 120 inches = 305 cm usually
    else:
        startDis = 50 # cut off 1st 50cm = 20" usually
        try:
            speed = averageVelocity # assume cm/ms . would need encode to work.....
            start = startDis/speed # CB edit # 20/speed # find t where SOCEM ~ 20" into plot
        except:
            speed = 50 # assume 50 cm/s
            start = startDis/speed # CB edit # 20/speed # find t where SOCEM ~ 20" into plot
        endDis = 305 # cut of after 120" usually
        end = endDis/speed # CB edit # time[-1] - (20/speed) # find t where SOCME ~ 20" before end of
plot

        disReferenced_input = str(input('Would you like to type in known distance points (y/n)? '))
```

```
        if disReferenced_input == 'y':
            disReferenced_PeakClick = True
        elif disReferenced_input == 'n':
            disReferenced_PeakClick = False
        else: # just in case
            disReferenced_PeakClick = False

        if disReferenced_PeakClick == True:
            startDisRef = (int(input('Start point x (in): ')))
            endDisRef = (int(input('End point x (in): ')))

        ''' '''
    maxPt = max(forcePushed)
    # draw cut off lines
    cutStart = [start, start]
    cutEnd = [end, end]
    cutLine = [0, maxPt]
    ax.plot(cutStart,cutLine, color = 'red') # start cut off line
    ax.plot(cutEnd, cutLine, color = 'red') # end cut off line


    # plot dis vs forcePushed
    if encoderWorked == True:
        ax.plot(distanceTraveled, forcePushed, color='midnightblue')
        ax.set_xlabel('Distance (cm)')
        snap_cursor = SnaptoCursor(ax, distanceTraveled, forcePushed) # create snap cursor object
    # plot time vs forcePushed
    else:
        ax.plot(timeElapsed, forcePushed, color='midnightblue')
        ax.set_xlabel('Time (sec)')
        snap_cursor = SnaptoCursor(ax, timeElapsed, forcePushed) # create snap cursor object

    title3 = '\n*click outside plot if red lines good*'
    fig.suptitle(GUI.filename_force.get() + '\nCut off edges: click start x pt, then end x pt.' + title3)
    #ax.set_title('*click outside plot if red lines good*')
    ax.set_ylabel('Force (N)')

    snap = fig.canvas.mpl_connect('motion_notify_event', snap_cursor.mouse_move) # update snap
cursor upon mouse movement

    xCut = [] # stores where to cut off ends of plot (eliminate edge effects)
    def click(event): # get x coord once mouse is pressed
        x = event.xdata
        if x is None:
            print('red lines')
            xCut.append((start))
            xCut.append((end))
        else:
```

```
            print('x clicked = %1.2f' % x)
            xCut.append((x))
          #  self.fig.canvas.mpl_disconnect(cid)
        if len(xCut) >= 2:
            fig.canvas.mpl_disconnect(cid)
            fig.canvas.mpl_disconnect(snap)
            fig.canvas.set_window_title('InitialPlot')
            if encoderWorked == True:
                savename = GUI.filename_force.get() + '_' + str(round(xCut[0],1)) + '-' +
str(round(xCut[1],1)) + '_raw.PNG'
            elif encoderWorked == False and disReferenced_PeakClick == True:
                savename = GUI.filename_force.get() + '_' + str(round(xCut[0],1)) + '-' +
str(round(xCut[1],1)) + '_disref' + '_raw.PNG'
            elif encoderWorked == False and disReferenced_PeakClick == False:
                savename = GUI.filename_force.get() + '_' + str(round(xCut[0],1)) + '-' +
str(round(xCut[1],1)) + '_timebased' + '_raw.PNG'
            else:
                savename = GUI.filename_force.get() + '_' + str(round(xCut[0],1)) + '-' +
str(round(xCut[1],1)) + '_else' + '_raw.PNG'
            ax.plot([xCut[0],xCut[0]],cutLine, color = 'orange', linewidth=1, linestyle="--") # start cut off
line
            ax.plot([xCut[1],xCut[1]], cutLine, color = 'orange', linewidth=1, linestyle="--") # end cut off
line
            #print("Initial plot show...")
            #plt.show()
            #print("Initial plot shown.")
            if not os.path.exists(documentationFolder):
              os.makedirs(documentationFolder) # Create documentationFolder because it does not exist
            print("did")
            plt.savefig(documentationFolder + '/' + savename)
            print("did2")
            plt.close(block)
            print("xCut = ",xCut)
        return xCut # return start & end x (dis. or time) pts

    cid = fig.canvas.mpl_connect('button_press_event', click) # connects click event
    print("cid")

    #clicked = snap_cursor.click()
    #print('clicked', snap_cursor.click.coords)
    plt.draw()
    plt.ion()
    plt.show() # never getting past this
    #plt.show(block=False) # never getting past this
    print("did4")
    print("encoderWorked = ",encoderWorked)
    print("disReferenced_PeakClick = ",disReferenced)
    if encoderWorked == True:
```

```
        return xCut
    elif encoderWorked == False and disReferenced_PeakClick == True:
        return xCut, disReferenced, disNew, i, j
    elif disReferenced_PeakClick == False:
        disNew = []
        return xCut, disReferenced, disNew, i, j
    print("Complete initial plot.")
    print("xCut, disReferenced, disNew,i,j = ", xCut, disReferenced, disNew,i,j)


################################################################################

def choosePeaks(xData, forcePushed, xCut, varietyAndPlotnameAndDetail, encoderWorked,
disReferenced, documentationFolder):
    #please: EMBED THE MATPLOTLIB PLOT INTO A TKINTER WINDOW< WHICH CAB BE A POPUP<
LEADING TO POPUP.MAINLOOP()
    encoderWorked = encoderWorked_override
    def nearest_pt(pt): # get nearest dis index to starting pt in disCut
        idx = (np.abs(np.asarray(xData)- pt)).argmin()
        #print('idx ', idx)
        return idx
    peakclick.peaks_force = [] # force peaks
    peakclick.peaks_xaxis = [] # x (distance) pt of force peak

    startIdx = nearest_pt(xCut[0]) # starting index
    print('startIdx = ',startIdx) # , dis[startIdx])
    endIdx = nearest_pt(xCut[1])
    print('Closest distance pts: ', [xData[startIdx] , xData[endIdx]])
    xCenter = xData[startIdx:endIdx]
    fCenter = forcePushed[startIdx:endIdx]

    fig, ax = plt.subplots()
    fig.canvas.set_window_title('ChoosePeaks')
    fig.suptitle(GUI.filename_force.get() + '\nSelect Force Peaks, *click outside when done*')
    #fig.suptitle(GUI.filename_force.get() + '\nCut off edges: click start x pt, then end x pt.' + title3)
    #ax.set_title('*click outside when done*')
    ax.plot(xCenter, fCenter) # needed?
    maxPt = max(forcePushed)
    ax.set_xlim(min(xCenter)-5, max(xCenter)+5)
    ax.set_ylabel('Force (Newtons)')
    ''' # set secondary vertical axis
    xold = np.asarray(xCenter)
    xnew = xold*convert_NToLbs
    def forward(x):
        return np.interp(x, xold, xnew)
    def inverse(x):
        return np.interp(x, xnew, xold)
    axis_pounds = ax.secondary_yaxis('right', functions=(forward,inverse))
```

```
    axis_pounds.set_ylabel('Force (pounds)')
    '''
    if encoderWorked == True or disReferenced_PeakClick == True:
        ax.set_xlabel('Distance (cm)')
    else:
        ax.set_xlabel('Time (sec)')

    cursor = Cursor(ax) # create snap cursor object
    cursorMove = fig.canvas.mpl_connect('motion_notify_event', cursor.mouse_move) # update snap
cursor upon mouse movement

    closeplt = False
    def click(event): # get x coord once mouse is pressed
        y, x = event.ydata, event.xdata
        #if y is None and len(peakclick.peaks_force)>2: # requires 3 clicks, or the window wont close
        #if y is None and len(peakclick.peaks_force)>0: # requires 1 click, or the window wont close
        if y is None: # window will close whenever you click out of the axes frame.
            cursorMove = fig.canvas.mpl_connect('motion_notify_event', cursor.mouse_move) # update
snap cursor upon mouse movement
            fig.canvas.mpl_disconnect(cid)
            fig.canvas.mpl_disconnect(cursorMove)
            # auto save file
            # example: CF452_24hr_4_23-156_disref_clicks.PNG
            if encoderWorked == True:
                savename = GUI.filename_force.get() + '_' + str(round(xCut[0],1)) + '-' +
str(round(xCut[1],1)) + '_clicks.PNG'
            elif encoderWorked == False and disReferenced_PeakClick == True:
                savename = GUI.filename_force.get() + '_' + str(round(xCut[0],1)) + '-' +
str(round(xCut[1],1)) + '_disref' + '_clicks.PNG'
            elif encoderWorked == False and disReferenced_PeakClick == False:
                savename = GUI.filename_force.get() + '_' + str(round(xCut[0],1)) + '-' +
str(round(xCut[1],1)) + '_timebased' + '_clicks.PNG'
            plt.savefig(documentationFolder + '/' + savename)
            print("choosePeaks: ",savename)

            print('peaks_xaxis = ', peakclick.peaks_xaxis)

            # lists of numbers, from analysis choice
            xCutL = ['xCut(in)'] # Distance, analysis range
            tCutL = ['tCut(sec)']# Time, analysis range
            print("test")
            plt.close()

            if encoderWorked == True:
                peakclick.peaks_distance = peakclick.peaks_xaxis
                peakclick.peaks_time =
peakclick.findmatchtime(forcePushed,distanceTraveled,timeElapsed,peaks_distance)
                print("peaks_force =",peakclick.peaks_force)
```

```
            elif encoderWorked == False and disReferenced_PeakClick == True:
                peakclick.peaks_distance = peakclick.peaks_xaxis
                peakclick.peaks_time =
peakclick.findmatchtime(forcePushed,distanceTraveled,timeElapsed,peaks_distance)
                print("peaks_force =",peakclick.peaks_force)
            else: #elif encoderWorked == False and disReferenced_PeakClick == False: # possible issue
                peakclick.peaks_time = peakclick.peaks_xaxis
                peakclick.peaks_distance = [0, 0, 0] # might error, if there are not three clicks
                print("peaks_force =",peakclick.peaks_force)

            peakclick.saveCSV(GUI.filename_force.get(),GUI.address)
            RecordForce.closedplt = True
        else:
            # print('Force clicked = %1.2f at %1.2f' % (y, x)) # hide, CB
            peakclick.peaks_force.append((y))
            peakclick.peaks_xaxis.append((x))
            '''
            peaks_force.append((y))
            peaks_xaxis.append((x))
            '''
            ax.scatter(x, y, color='red')
            cursorMove = fig.canvas.mpl_connect('motion_notify_event', cursor.mouse_move) # update
snap cursor upon mouse movement
            fig.canvas.draw()

        if peakclick.peaks_force == []:
            quit()

    #plt.draw() # the magic ingredient
    plt.ion()
    #fig.canvas.draw()
    cid = fig.canvas.mpl_connect('button_press_event', click) # connects click event
    #fig.canvas.draw()
    plt.show()

# MAIN
class peakclick:
    '''
    - Finish autoclicker by setting plt.show() into an inset tkinter gui popup, and then mainloop.
        Use: FigureCanvasTkAgg,NavigationToolbar2Tk,plt,Cursor.
    '''
    def __init__():
        peakclick.peaks_force = []
        peakclick.peaks_distance = []
        peakclick.peaks_time = []

    def findmatchtime(forcePushed,distanceTraveled,timeElapsed,peaks_distance):
        i=0
```

```
    for peaks_distance_i in peaks_distance:
        peaks_time = timeElapsed[distanceTraveled.find(peaks_axis_i)]
        i+=1
    return peaks_time

  # peaks_force,peaks_distance,peaks_time =
peakclick.input(GUI.forcePushed,GUI.distanceTraveled,GUI.timeElapsed,GUI.filename_force.get(),G
UI.address)
  def
peakclick(forcePushed,distanceTraveled,timeElapsed,varietyAndPlotnameAndDetail,address,averag
eVelocity):

    #documentationFolder = GUI.address + '/' + 'documentation'
    documentationFolder = GUI.address # for PNG and raw data to go to the same place.
    if max(distanceTraveled) > 10: # Assess if the encoder worked or not. Assuems that if it worked,
the max value would exceeed 1 inch.
        encoderWorked = True #
    else: encoderWorked = False
    encoderWorked = encoderWorked_override # leaving this here means all graphs will be shown
in force vs time

    print('Encoder? ', encoderWorked)
    print('max(distanceTraveled) = ', str(max(distanceTraveled)))
    print(GUI.filename_force.get())

    if useInitialPlot_PeackClick == True:
        if encoderWorked == False:
            xCut, disReferenced, disNew,i,j = initialPlot(distanceTraveled, forcePushed, timeElapsed,
encoderWorked, GUI.filename_force.get(), documentationFolder,averageVelocity)
        elif encoderWorked == True:
            xCut = initialPlot(distanceTraveled, forcePushed, timeElapsed,
encoderWorked,GUI.filename_force.get(), documentationFolder,averageVelocity)
            disReferenced_PeakClick = False
    else:
        xCut = [min(distanceTraveled),max(distanceTraveled)]
        tCut = [min(timeElapsed),max(timeElapsed)]
        disReferenced_PeakClick = False

    if encoderWorked == True:
        print('Distance cut at: ', xCut) # cut forcePushed and horz!!!
        #peakclick.peaks_force,peakclick.peaks_xaxis = choosePeaks(distanceTraveled, forcePushed,
xCut,GUI.filename_force.get(),encoderWorked, disReferenced_PeakClick,documentationFolder)
        choosePeaks(distanceTraveled, forcePushed, xCut,GUI.filename_force.get(),encoderWorked,
disReferenced_PeakClick,documentationFolder)
    elif encoderWorked == False and disReferenced_PeakClick == True:
        print('troubleshoot702')
        print('Distance cut at: ', xCut)
```

```
            #peakclick.peaks_force,peakclick.peaks_xaxis = choosePeaks(disNew, forcePushed,
xCut,GUI.filename_force.get(),encoderWorked, disReferenced_PeakClick, documentationFolder)
            choosePeaks(disNew, forcePushed, xCut,GUI.filename_force.get(),encoderWorked,
disReferenced_PeakClick, documentationFolder)
        else: #elif encoderWorked == False and disReferenced_PeakClick == False: # possible issue dave
            xCut=tCut
            print('Time cut at: ', xCut)
            #peakclick.peaks_force,peakclick.peaks_xaxis = choosePeaks(timeElapsed, forcePushed,
xCut,GUI.filename_force.get(),encoderWorked, disReferenced_PeakClick, documentationFolder)
            choosePeaks(timeElapsed, forcePushed, xCut,GUI.filename_force.get(),encoderWorked,
disReferenced_PeakClick, documentationFolder)

        #peakclick.saveCSV(GUI.filename_force.get(),GUI.address)
        #return peakclick.peaks_force,peakclick.peaks_distance,peakclick.peaks_time

    def saveCSV(varietyAndPlotnameAndDetail,address):
        #print("not yet saved. develop.")
        filename_peaks_csv = GUI.address + "/" + GUI.filename_force.get() + "_peaks.csv"
        ''' write CSV'''
        GUI.data_peaks = [peakclick.peaks_force,peakclick.peaks_distance,peakclick.peaks_time]
        RecordForce.peaks_force = peakclick.peaks_force
        RecordForce.peaks_distance = peakclick.peaks_distance
        RecordForce.peaks_time = peakclick.peaks_time
        RecordForce.peaks_distance.insert(0, "PeaksDistance(cm)")
        RecordForce.peaks_force.insert(0, "PeaksForce(N)")
        RecordForce.peaks_time.insert(0 , "PeaksTime(sec)")
        columns_data_peaks = zip_longest(*GUI.data_peaks)
        with open(filename_peaks_csv,'w',newline='') as f:
            writer = csv.writer(f)
            writer.writerows(columns_data_peaks)
        ''' end: write CSV '''
        print("filename_peaks_csv = "+filename_peaks_csv)
        RecordForce.peaks_force = peakclick.peaks_force
        RecordForce.peaks_distance = peakclick.peaks_distance
        RecordForce.peaks_time = peakclick.peaks_time

class EI_Interaction_Fx:
    '''
    Closed-form solution for calculating EI via the Multiple Inline Interacting Cantilever Beam Model
    Author: Austin Bebee
    Last updated: 7/6/2020
    Require input values for peak force (f), force bar height (h), beam length (l), beam-to-beam
spacing (s).
    Assumes the system contains the full/max number of beams (full interaction) at the first beam's
max deflection.
    If this is not the case, set the variable "finite_beam_num" to True and set the variable "beam-
num" to the number
    of beams in a row.
```

```
    Additional assumptions:
        - beams deflect linearly inline
        - force bar force always perpendicular to 1st beam's end angle
        - each beam has same K & KO
    '''
    # INPUT PARAMETERS. EI will be calculated in units of f*(l^2)
    # example
    #f = 5  # peak force
    #h = 8  # force bar height
    #l = 10  # beam length
    #s = 1  # beam-to-beam spacing
    global definite_beam_num, beams
    definite_beam_num = False  # if False, assumes max number of beams (full interaction) at the first beam's max deflection
    beams = 8  # num. of beams in a row (only used if "definite_beam_num" set to True)

    # Model lists/arrays - each index is a beam's attribute (0 index = 1st beam, last index = last beam)
    global EI_Interaction_Fx_theta
    global EI_Interaction_Fx_betaA
    global EI_Interaction_Fx_x_deflection
    global EI_Interaction_Fx_dist_horz
    global EI_Interaction_Fx_phi
    global EI_Interaction_Fx_forces
    global EI_Interaction_Fx_q_len
    global EI_Interaction_Fx_KO
    global EI_Interaction_Fx_gamma_length

    EI_Interaction_Fx_theta = list()  # PRBM angle (radians)
    EI_Interaction_Fx_betaA = list()  # math.pi/2 - theta (radians)
    EI_Interaction_Fx_x_deflection = list()  # x deflection
    EI_Interaction_Fx_dist_horz = list()  # horizontal distance a beam extends past the next (x - s)
    EI_Interaction_Fx_phi = list()  # force vector angle w/ respect to undeflected axis (vertical axis in this case)
    EI_Interaction_Fx_forces = list()  # individual reaction forces
    EI_Interaction_Fx_q_len = list()  # effective beam lengths
    EI_Interaction_Fx_KO = list()  # stiffness coefficient
    EI_Interaction_Fx_gamma_length = list()  # gamma*l (longer rigid link length)

    def clearAll(): # Clears variables for new simulation
        EI_Interaction_Fx_betaA.clear()
        EI_Interaction_Fx_x_deflection.clear()
        d.clear()
        EI_Interaction_Fx_phi.clear()
        EI_Interaction_Fx_forces.clear()
        EI_Interaction_Fx_q_len.clear()
        EI_Interaction_Fx_KO.clear()
        EI_Interaction_Fx_gamma_length.clear()
```

```
   def MultiPhiCor(h, l, s, phi): # Phi correctoin for multiple beams (exp. developed) used when h/l < 0.7
      if h/l < 0.7:
         mphi = 244.7802*(s/l) - 683.4973*((s/l)**2) - 165.1557*((s/l)*(h/l)) + 43.4227*((h/l)**2)
         mphi = mphi * ((math.pi) / 180)
      else:
         mphi = phi

      return mphi

   def Parametric_angle_coefficient(n): # returns c (parametric angle coefficient) when given n
      if -4 < n <= -1.5:
         c = 1.238945 + 0.012035*n + 0.00454*(n**2)
      elif -0.5 < n: # <= 10: # some conditions yield n = 10.25, which is just beyond the defined limits of n. Can either remove n <= 10 boundary or set n = 10 if n > 10.
         c = 1.238845 + 0.009113*n - 0.001929*(n**2) + 0.000191*(n**3) - 0.000007*(n**4)
      else:
         c = 1.238845 + 0.009113*n - 0.001929*(n**2) + 0.000191*(n**3) - 0.000007*(n**4) # added 8/9/2022
      return c

   def gammaUpdate(n):# returns gamma value when give n
      if n > 10:
         n = 10
      if n > .5: # <= 10: # some conditions yield n = 10.25, which is just beyond the defined limits of n. Can either remove n <= 10 boundary or set n = 10 if n > 10.
         gamma = .841655 - 0.0067807 * n + .000438 * (n ** 2)
      elif n > -1.8316 and n < 0.5:
         gamma = .852144 - 0.0182867 * n
      elif n > -5 and n < -1.8316:
         gamma = .912364 + .0145928 * n # added 9/8/2022
      else: # added 9/8/2022
         gamma = .912364 + .0145928 * n # added 9/8/2022

      return gamma

   def KOupdate(n):# returns Ko (stiffness coefficient) when given n
      if n > -5 and n <= -2.5:
         Ko = (3.024112 + 0.121290 * n + 0.003169 * (n ** 2))
      elif n > -2.5 and n <= -1:
         Ko = (1.967647 - 2.616021 * n - 3.738166 * (n ** 2) - 2.649437 * (n ** 3) - 0.891906 * (n ** 4) - 0.113063 * (n ** 5))
      elif n > -1: # and n <= 10: # <= 10: # some conditions yield n = 10.25, which is just beyond the defined limits of n. Can either remove n <= 10 boundary or set n = 10 if n > 10.
         Ko = (2.654855 - 0.509896 * (10 ** -1) * n + 0.126749 * (10 ** -1) * (n ** 2) - 0.142039 * (10 ** -2) * (n ** 3) + 0.584525 * (10 ** -4) * (n ** 4))
      else: # added 8/9/2022
```

```
        Ko = (2.654855 - 0.509896 * (10 ** -1) * n + 0.126749 * (10 ** -1) * (n ** 2) - 0.142039 * (10
** -2) * (n ** 3) + 0.584525 * (10 ** -4) * (n ** 4))

      return Ko

  # MAIN FUNCTION THAT RETURNS THE SYSTEM'S MEAN EI
  def EI_Interaction(f, h, l, s):
    #f, h, l, s = peak_force, stemheight, barbottom,stemspacing_average
    print(f, h, l, s)
    f=f/4.44822 # N to lbs
    h=h/2.54 # cm to inches
    l=l/2.54 # cm to inches
    s=s/76*30 # adjust barlength by cm to by inches
    ## DETERMINE 1ST BEAM'S KINEMATICS ##

    #initial assumption that force bar applies a horizontal force yields:
    n = 0 # initial n value
    gamma = EI_Interaction_Fx.gammaUpdate(n)  # initial gamma

    # Gamma Convergence Cycle - update gamma for better estimate
    # issue is here, Nans are coming out
    j = 0
    while j < 100:  # gamma converges well before 100 iterations
      b = (1 - gamma) * l  # length below torsional spring (in.)
      # betaV needs edge case management - the solution should have been programmed already
on the laptop, or the desktop
      betaV = (np.arcsin((h - b) / (gamma * l)))  # beta angle value (rads)
      if np.isnan(betaV):
        betaV = 0
      thetaV = (math.pi / 2) - betaV  # PRBM theta value (rads)
      c = EI_Interaction_Fx.Parametric_angle_coefficient(n)  # get parametric angle coefficient
      beam_end_angle = thetaV * c  # corrected beam end angle
      phiV = (math.pi / 2) - beam_end_angle  # phi value (perpendicular to beam end angle)
      n = (1 / np.tan(phiV))  # update n
      #  Option below: if n > 10 (beyond PRBM defined limits) set n = 10 (note: negligible change if
used)
      if n > 10:
        n = 10
      gamma = EI_Interaction_Fx.gammaUpdate(n)  # update gamma
      j += 1

    b1 = b # stores final b of 1st beam
    gamma1 = gamma  # stores final gamma of 1st beam

    # 1st beam's geometry & attributes:
    EI_Interaction_Fx_q_len.insert(0, l)  # effective beam length
    EI_Interaction_Fx_gamma_length.insert(0, gamma * l)  # stores longer rigid link length
    Ko = EI_Interaction_Fx.KOupdate(n)  # update Ko (stiffness coefficient)
```

```
    EI_Interaction_Fx_KO.insert(0, Ko)  # stores 1st beam's stiffness coefficient
    EI_Interaction_Fx_betaA.insert(0, betaV)  # stores beta angle
    EI_Interaction_Fx_x_deflection.insert(0, EI_Interaction_Fx_gamma_length[0] *
np.cos(EI_Interaction_Fx_betaA[0]))  # stores x deflection
    EI_Interaction_Fx_dist_horz.insert(0, EI_Interaction_Fx_x_deflection[0] - s)  # stores horizontal
distance beam extends past the next

    phiCorrection = EI_Interaction_Fx.MultiPhiCor(h, l, s, phiV) # correct phi (if h/l < 0.7)
    EI_Interaction_Fx_phi.insert(0, phiCorrection)  # force vector angle w/ respect to undeflected
axis

    # ALL OTHER BEAM'S KINEMATICS
    def otherBeams(i): # called after determining # of beams
        EI_Interaction_Fx_betaA.insert(i + 1, np.arctan((EI_Interaction_Fx_gamma_length[i] *
np.sin(EI_Interaction_Fx_betaA[i])) / EI_Interaction_Fx_dist_horz[i]))
        phiV = (EI_Interaction_Fx_betaA[i])  # initially assume phi = previous beta
        n = (1 / np.tan(phiV))  # determine n
        c = EI_Interaction_Fx.Parametric_angle_coefficient(n) # determine c
        phiV = (math.pi / 2) - (math.pi / 2 - EI_Interaction_Fx_betaA[i]) * c # correct & update phi
        phiCorrection = EI_Interaction_Fx.MultiPhiCor(h, l, s, phiV) # correct phi (if h/l < 0.7)
        EI_Interaction_Fx_phi.insert(i + 1, phiCorrection) # store beam's phi
        Ko = EI_Interaction_Fx.KOupdate(n)  # update Ko
        EI_Interaction_Fx_KO.insert(i+1, Ko)  # store beam's Ko
        gamma = EI_Interaction_Fx.gammaUpdate(n)  # update gamma
        b = (1 - gamma) * l  # update base length
        EI_Interaction_Fx_q_len.insert(i + 1, b + math.sqrt(EI_Interaction_Fx_dist_horz[i] ** 2 +
(EI_Interaction_Fx_gamma_length[i] * np.sin(EI_Interaction_Fx_betaA[i])) ** 2))  # effective
cantilever beam length (base to applied force from previous beam or force bar)
        EI_Interaction_Fx_gamma_length.append(gamma * l)  # store beam's gamma*length
        EI_Interaction_Fx_x_deflection.insert(i + 1, EI_Interaction_Fx_gamma_length[i + 1] *
np.cos(EI_Interaction_Fx_betaA[i + 1]))  # x deflection at end of beam
        #y.insert(i+1, EI_Interaction_Fx_gamma_length[i+1] * np.sin(beta[i + 1])) # y deflection at end
of beam (not needed for EI calculation)
        EI_Interaction_Fx_dist_horz.insert(i + 1, EI_Interaction_Fx_x_deflection[i + 1] - s)# stores
horizontal distance beam extends past the next


        return EI_Interaction_Fx_dist_horz[i]  # returns distance to check to continue looping or not

    # Determine # of other beams:
    if definite_beam_num == False:  # full interaction @ 1st beam's max deflection
        i = 0
        while EI_Interaction_Fx_dist_horz[i] > 0:  # will previous beam hit next beam? If so, run that
beam through otherBeams()
            EI_Interaction_Fx.otherBeams(i)
            i += 1
    else:  # definite number of beams in a row (may not be full/max interaction possible)
        i = 0
```

```
        while EI_Interaction_Fx_dist_horz[i] > 0 and i < beams-1:  # will previous beam hit next beam
& does that beam exist? If so, run that beam through otherBeams()
            EI_Interaction_Fx.otherBeams(i)
            i += 1

    # BACKSOLVE TO GET ALL FORCES/K EXCEPT 1ST FORCE/K

    # Last beam force/k
    if len(EI_Interaction_Fx_betaA) > 1:  # check if more than 1 beam
        num = ((math.pi / 2) - EI_Interaction_Fx_betaA[-1])  # numerator = theta
        den = ((EI_Interaction_Fx_x_deflection[-2] - s) * np.cos(EI_Interaction_Fx_phi[-1]) +
(EI_Interaction_Fx_gamma_length[-2] * np.sin(EI_Interaction_Fx_betaA[-2]) *
np.sin(EI_Interaction_Fx_phi[-1])))  # denominator
        EI_Interaction_Fx_forces.insert(-1, num / den)  # last force/k
    else:  # if only one beam, skip
        pass

    # middle beam forces/k
    j = -2
    if len(EI_Interaction_Fx_betaA) > 1:  # check if more than 1 beam
        while j > -(len(EI_Interaction_Fx_betaA)):  # loop until reaching 1st force/K
            num1 = ((math.pi / 2) - EI_Interaction_Fx_betaA[j])  # 1st numerator term = theta
            num2 = EI_Interaction_Fx_forces[j + 1] * (EI_Interaction_Fx_gamma_length[j] *
np.sin(EI_Interaction_Fx_betaA[j]) * np.sin(EI_Interaction_Fx_phi[j + 1]) + x[j] *
np.cos(EI_Interaction_Fx_phi[j + 1]))  # force due to previous beam
            den1 = (EI_Interaction_Fx_x_deflection[j - 1] - s) * np.cos(EI_Interaction_Fx_phi[j])  #
denominator term 1
            den2 = EI_Interaction_Fx_gamma_length[j - 1] * np.sin(EI_Interaction_Fx_betaA[j - 1]) *
np.sin(EI_Interaction_Fx_phi[j])  # denominator term 2
            EI_Interaction_Fx_forces.insert(j, (num1 + num2) / (den1 + den2))  # forces/k

            j = j - 1  # increment backwards

    # 1st Beam calculations
    print("EI_Interaction_Fx_phi[0] = ",EI_Interaction_Fx_phi[0])
    print("np.sin(EI_Interaction_Fx_phi[0]) = ",np.sin(EI_Interaction_Fx_phi[0]))
    print("f = ",f)
    fx = f/np.sin(EI_Interaction_Fx_phi[0])
    EI_Interaction_Fx_forces.insert(0, fx)  # store force bar applied force in 0 index

    # calculate K (torsional spring constant) of 1st beam
    knum1 = fx * (EI_Interaction_Fx_x_deflection[0] * np.cos(EI_Interaction_Fx_phi[0]) + (h - b1) *
np.sin(EI_Interaction_Fx_phi[0]))  # numerator
    kden1 = ((math.pi / 2) - EI_Interaction_Fx_betaA[0])  # denominator 1st term

    if len(EI_Interaction_Fx_betaA) > 1:  # if more than 1 beam (i.e., interacting beams)
```

```
        kden2 = EI_Interaction_Fx_forces[1] * ((h - b1) * np.sin(EI_Interaction_Fx_phi[1]) +
EI_Interaction_Fx_x_deflection[0] * np.cos(EI_Interaction_Fx_phi[1])) # denominator 2nd term due
to interactions
      else: # only 1 beam, no interactions
        kden2 = 0

      K = (knum1) / (kden1 + kden2)  # compute K (torsional spring constant)

      ## COMPUTE EI ##
      EI = (l * K) / (EI_Interaction_Fx_KO[0] * gamma1)

      #t_num = len(EI_Interaction_Fx_betaA)  # total number of interacting beams @ 1st beam's
deflection
      # EI here is in lb*in^2
      EI = EI*4.44822*2.54*2.54 # convert lb*in^2 to N*cm*2
      return EI

   def test(f, h, l, s):
      clearAll()
      EI = EI_Interaction(f, h, l, s)
      print(EI)
   #test(7, 5, 10, 1)

class EI_No_Interaction_Fx:
   '''
   No-Interaction Closed-form solution for calculating EI via the Multiple Inline Non-Interacting
Cantilever Beam Model
   Author: Austin Bebee
   Last updated: 7/6/2020
   Require input values for peak force (f), force bar height (h), beam length (l), beam-to-beam
spacing (s).
   Assumes the system contains the full/max number of beams at the first beam's max deflection.
   If this is not the case, set the variable "finite_beam_num" to True and set the variable "beam-
num" to the number
   of beams in a row.
   Additional assumptions:
      - no contact between beams. Beams only contact the force bar
      - force bar force always perpendicular to 1st beam's end angle
      - each beam may have different K & KO
   '''
   # INPUT PARAMETERS (EI will be calculated in units of f*l^2)
   # f = 5 # peak force
   # h = 8 # force bar height
   # l = 10 # beam length
   # s = 1 # beam-to-beam spacing
   global definite_beam_num, beams
   definite_beam_num = False # # if False, assumes max number of beams at the first beam's max
deflection
```

```
    beams = 8 # num. of beams in a row (only used if "definite_beam_num" set to True)

    # Model lists/arrays - each index is a beam's attribute (0 index = 1st beam, last index = last beam)
    global EI_No_Interaction_Fx_theta
    global EI_No_Interaction_Fx_betaA
    global EI_No_Interaction_Fx_x_deflection
    global EI_No_Interaction_Fx_dist_horz
    global EI_No_Interaction_Fx_phi
    global EI_No_Interaction_Fx_q_len
    global EI_No_Interaction_Fx_KO
    global EI_No_Interaction_Fx_gamma_length
    global EI_No_Interaction_Fx_dist_horzens

    EI_No_Interaction_Fx_theta = list() # PRBM angle (rads)
    EI_No_Interaction_Fx_betaA = list() # math.pi/2 - theta (rads)
    EI_No_Interaction_Fx_x_deflection = list() # x deflection
    EI_No_Interaction_Fx_dist_horz = list() # horizontal distance a beam extends past the next (x - s)
    EI_No_Interaction_Fx_phi = list() # 180 deg. - alpha
    EI_No_Interaction_Fx_q_len = list()# effective l term in following k's equation
    EI_No_Interaction_Fx_KO = list() # stiffness coefficient
    EI_No_Interaction_Fx_gamma_length = list() # gamma*l (longer rigid link length)
    EI_No_Interaction_Fx_dist_horzens = list() # denominator terms for EI

    def clearAll(): # clears variables for new simulation
        EI_No_Interaction_Fx_betaA.clear()
        EI_No_Interaction_Fx_theta.clear()
        EI_No_Interaction_Fx_x_deflection.clear()
        EI_No_Interaction_Fx_dist_horz.clear()
        EI_No_Interaction_Fx_phi.clear()
        EI_No_Interaction_Fx_q_len.clear()
        EI_No_Interaction_Fx_KO.clear()
        EI_No_Interaction_Fx_gamma_length.clear()
        dens.clear()

    def Parametric_angle_coefficient(n): # returns c (parametric angle coefficient) when given n
        if -4 < n <= -1.5:
            c = 1.238945 + 0.012035*n + 0.00454*(n**2)
        elif -0.5 < n: # <= 10: # some conditions yield n = 10.25, which is just beyond the defined limits
of n. Can either remove n <= 10 boundary or set n = 10 if n > 10.
            c = 1.238845 + 0.009113*n - 0.001929*(n**2) + 0.000191*(n**3) - 0.000007*(n**4)
        else:
            c = 1.238845 + 0.009113*n - 0.001929*(n**2) + 0.000191*(n**3) - 0.000007*(n**4) # added
8/9/2022
        return c

    def gammaUpdate(n):# returns gamma value when give n
        if n > 10:
            n = 10
```

```
        if n > .5: # <= 10: # some conditions yield n = 10.25, which is just beyond the defined limits of n. Can either remove n <= 10 boundary or set n = 10 if n > 10.
            gamma = .841655 - 0.0067807 * n + .000438 * (n ** 2)
        elif n > -1.8316 and n < 0.5:
            gamma = .852144 - 0.0182867 * n
        elif n > -5 and n < -1.8316:
            gamma = .912364 + .0145928 * n
        else: # added 9/8/2022
            gamma = .912364 + .0145928 * n # added 9/8/2022

        return gamma

    def KOupdate(n):# returns Ko (stiffness coefficient) when given n
        if n > -5 and n <= -2.5:
            Ko = (3.024112 + 0.121290 * n + 0.003169 * (n ** 2))
        elif n > -2.5 and n <= -1:
            Ko = (1.967647 - 2.616021 * n - 3.738166 * (n ** 2) - 2.649437 * (n ** 3) - 0.891906 * (n ** 4) - 0.113063 * (n ** 5))
        elif n > -1: # and n <= 10: # <= 10: # some conditions yield n = 10.25, which is just beyond the defined limits of n. Can either remove n <= 10 boundary or set n = 10 if n > 10.
            Ko = (2.654855 - 0.509896 * (10 ** -1) * n + 0.126749 * (10 ** -1) * (n ** 2) - 0.142039 * (10 ** -2) * (n ** 3) + 0.584525 * (10 ** -4) * (n ** 4))
        else: # added 8/9/2022
            Ko = (2.654855 - 0.509896 * (10 ** -1) * n + 0.126749 * (10 ** -1) * (n ** 2) - 0.142039 * (10 ** -2) * (n ** 3) + 0.584525 * (10 ** -4) * (n ** 4))

        return Ko

    # MAIN FUNCTION THAT RETURNS THE SYSTEM'S MEAN EI
    def EI_NoInteraction(f, h, l, s):
        #f, h, l, s = peak_force, stemheight, barbottom,stemspacing_average
        f=f/4.44822 # N to lbs
        h=h/2.54 # cm to inches
        l=l/2.54 # cm to inches
        s=s/76*30 # adjust barlength by cm to by inches
        ## DETERMINE 1ST BEAM'S KINEMATICS ##

        #initial assumption that force bar applies a horizontal force yields:
        n = 0 # initial n value
        gamma = EI_No_Interaction_Fx.gammaUpdate(n)  # initial gamma

        # Gamma Convergence Cycle - update gamma for better estimate
        j = 0

        while j < 100:  # gamma converges well before 100 iterations
            b = (1 - gamma) * l  # length below torsional spring (in.)
            betaV = (np.arcsin((h - b) / (gamma * l)))  # beta angle value (rads)
            if np.isnan(betaV):
```

```
            betaV = 0
        thetaV = (math.pi / 2) - betaV  # PRBM theta value (rads)
        c = EI_No_Interaction_Fx.Parametric_angle_coefficient(n)  # get parametric angle coefficient
        beam_end_angle = thetaV * c  # corrected beam end angle
        phiV = (math.pi / 2) - beam_end_angle  # phi value (perpendicular to beam end angle)
        n = (1 / np.tan(phiV))  # update n
        #  Option below: if n > 10 (beyond PRBM defined limits) set n = 10 (note: negligible change if
used)
        if n > 10:
            n = 10
        gamma = EI_No_Interaction_Fx.gammaUpdate(n)  # update gamma
        j += 1

    b1 = b # stores final b of 1st beam
    gamma1 = gamma # stores final gamma of 1st beam

    # 1st beam's geometry & attributes:
    EI_No_Interaction_Fx_q_len.insert(0, l)  # effective beam length
    EI_No_Interaction_Fx_gamma_length.insert(0, gamma1 * l)  # stores longer rigid link length
    if np.isnan(EI_No_Interaction_Fx_gamma_length[0]):
        print("np.isnan(EI_No_Interaction_Fx_gamma_length[0])
=",np.isnan(EI_No_Interaction_Fx_gamma_length[0]))
        EI_No_Interaction_Fx_gamma_length[0]= 1 # fix this to a more realistic value
        print("EI_No_Interaction_Fx_gamma_length[0]) =",EI_No_Interaction_Fx_gamma_length[0])
    Ko = EI_No_Interaction_Fx.KOupdate(n)  # update Ko (stiffness coefficient)
    EI_No_Interaction_Fx_KO.insert(0, Ko)  # stores 1st beam's stiffness coefficient
    EI_No_Interaction_Fx_betaA.insert(0, betaV)  # stores beta angle
    EI_No_Interaction_Fx_x_deflection.insert(0, EI_No_Interaction_Fx_gamma_length[0] *
np.cos(EI_No_Interaction_Fx_betaA[0]))  # stores x deflection

    EI_No_Interaction_Fx_dist_horz.insert(0, EI_No_Interaction_Fx_x_deflection[0] - s)  # stores
horizontal distance beam extends past the next
    EI_No_Interaction_Fx_phi.insert(0, phiV)  # force vector angle w/ respect to undeflected axis

    # ALL OTHER BEAM'S KINEMATICS
    i = 0
    def otherBeams(i): # called after determining # of beams
        EI_No_Interaction_Fx_betaA.insert(i+1, np.arctan((EI_No_Interaction_Fx_gamma_length[0] *
np.sin(EI_No_Interaction_Fx_betaA[0])) / (EI_No_Interaction_Fx_dist_horz[i])))
        phiV = EI_No.Interaction_Fx.beta[i] # initialy assum phi = previous beta
        n = (1/np.tan(phiV)) # determine n
        c = EI_No_Interaction_Fx.Parametric_angle_coefficient(n) # get parametric angle coefficient
        phiV = (math.pi / 2) - (math.pi / 2 - EI_No_Interaction_Fx_betaA[i]) * c # correct & update phi
        EI_No_Interaction_Fx_phi.insert(i+1, phiV) # store beam's phi
        Ko = EI_No_Interaction_Fx.KOupdate(n) # update Ko
        EI_No_Interaction_Fx_KO.insert(i+1, Ko) # store Ko
        gamma = EI_No_Interaction_Fx.gammaUpdate(n) # update gamma
        EI_No_Interaction_Fx_gamma_length.append(gamma * l) # store g*l
```

```
        b = ((1 - gamma) * l) # update base length
        EI_No_Interaction_Fx_x_deflection.insert(i+1, EI_No_Interaction_Fx_dist_horz[i]) # x distance
from base of beam to force bar's positon @ 1st beam's max deflection
        EI_No_Interaction_Fx_dist_horz.insert(i+1, EI_No_Interaction_Fx_x_deflection[i+1] - s) #
horizontal distance beam extends past the next
        EI_No_Interaction_Fx_q_len.insert(i+1, b +
math.sqrt(EI_No_Interaction_Fx_x_deflection[i]**2 +
(EI_No_Interaction_Fx_gamma_length[0]*np.sin(EI_No_Interaction_Fx_betaA[0]))**2)) # effective
cantilever beam length (base to applied force)

        return i

    # Determine # of other beams:
    numBeams = int(round(EI_No_Interaction_Fx_x_deflection[0]/s)) # number of additional beams
hitting force bar at 1st one's max deflection

    if definite_beam_num == False:  # full number of beams @ 1st beam's max deflection
        i = 0
        while i <= (numBeams-2):  # will beam hit force bar? If so, run that beam through
otherBeams()
        # numBeams - 2: -2 because 1st beam already computed, i starts at 0
            EI_No_Interaction_Fx.otherBeams(i)
            i += 1
    else:  # definite number of beams in a row (may expect more beams than the system actually
has)
        i = 0
        while i <= (numBeams-2) and i < beams-1:  # will beam hit force bar & does that beam exist?
If so, run that beam through otherBeams()
            EI_No_Interaction_Fx.otherBeams(i)
            i += 1

    # Calculations required to compute EI
    for i in range(len(EI_No_Interaction_Fx_betaA)):
        EI_No_Interaction_Fx_theta.insert(i, ((math.pi/2)-EI_No_Interaction_Fx_betaA[i])) # PRBM
Theta
        #s.insert(i, deltaTheta[i]/(gamma*(q[i]**2)))
        dens.insert(i, (EI_No_Interaction_Fx_KO[i]*theta[i])/(EI_No_Interaction_Fx_q_len[i]**2)) #
denominator terms for EI
    EIden = sum(dens) # denominator sum term for EI
    # Compute EI
    print("np.sin(EI_No_Interaction_Fx_phi[0]) = ",np.sin(EI_No_Interaction_Fx_phi[0]))
    print("f = ",f)
    Ftot = f/np.sin(EI_No_Interaction_Fx_phi[0]) # estimate F total from Fx
    EI = Ftot/EIden
    # EI here is in lb*in^2
    EI = EI*4.44822*2.54*2.54 # convert lb*in^2 to N*cm*2
    return EI
```

```
    def test(f, h, l, s):
        clearAll()
        EI = EI_NoInteraction(f, h, l, s)
        print(EI)

    #test(5, 8, 10, 1)


''' Main '''
print("StemBerry is loading.....")
print("output: address = "+address)
print("script = "+script)
print("directory = "+directory)
print("ignoreserial = "+str(ignoreserial))
app = GUI() # INITIATES GUI TO START
app.title("StemBerry")
app.geometry("800x480+0+0")
app.aspect()
#app.geometry("700x700+0+0")
#fig = plt.figure()
#app.iconbitmap(s'/home/pi/Desktop/SOCEM Code')
#app.geometry("{0}x{1}+0+0".format(app.winfo_screenwidth()-3,app.winfo_screenheight()-3)) #full
screen:
app.mainloop()
''' End '''
```

Appendix B2: spreadsheetsToTable_v3.m

```
% Title: spreadsheetsToTable_v3.m
% Author: Clayton Bennett
% Created: 16 March 2022
% Last edited: 27 September 2022
% Purpose:
% - Import any xlsx file!! Expect a single row for single values. Any
% columns longer than 1 row will be imported as a cells.
% - Checks that the filename is present as a data column. If not, two
% columns will be added, one for the short name and one for the entire file
% location.

% SOCEM specific:
% - Pull in data from analyzed SOCEM files, in "EI, Analyzed" and "EI,
%     Analyzed_timebased" folders.
% - This data does not include things like run numbers and hour label,
% though these are discernible from description and filename text.
savestuff = 'y';
format compact
%% Import data
% Edit this section for your specific needs.
% names key folders

directory_script = ''; % ENTER script location
dir_compiledData = ''; % ENTER target file directory

if (directory_script(end)~='\')
   directory_script = strcat(directory_script,'\');
end

% folders for dividing up the 2021 data
level1names = {'August5','August6','August10','August13'};
%level1names = {'August28'};
%level1names = {'correctedHeights'}
level2names = {'EI_outputFiles'}; % Use both of these?
%level2names = {'cleanRaw'}; % Use both of these?
list_xlsxfiles = {};

for i_level1name = 1:numel(level1names) % i_level1name=4;
   for j_level2name = 1:numel(level2names) % j_level2name=1;
      activefolder =
strcat(directory_data,level1names{i_level1name},'\',level2names{j_level2name},'\');
      cd(activefolder)
      % files files in active folder, then remove files that are not of the
```

```
      % proper filetype
      desired_import_filetype = '.xlsx';
      list_xlsxfiles_active = strtrim(string(ls())); %string(ls()) % cellstr(ls())

      i=1;
      while i<=numel(list_xlsxfiles_active)
         if not(contains(list_xlsxfiles_active(i),desired_import_filetype))
            list_xlsxfiles_active(i)=[];
         else
            list_xlsxfiles_active(i) = strcat(activefolder,list_xlsxfiles_active(i));
            i=i+1;
         end
      end

      for i=1:numel(list_xlsxfiles_active)
         list_xlsxfiles{end+1}=list_xlsxfiles_active(i);
      end
   end % end level 1 names loop
end % end level 2 names loop
n_files = numel(list_xlsxfiles);

%% Import data from each file
% Look one column at a time. Record the column header name. If the column
% is greater than one row long, package it as a cell

% Some numeric data saved by Python was output as string. Find which.
stringToDouble=[];
cellTrack=[];
list_filepath= {};
list_filedetail = {};


handle_waitbar=waitbar(0,strcat('Data is being imported from ',string(n_files),' SOCEM files.
Computer speed, go!'));
for i=1:n_files

   filename = list_xlsxfiles{i};
   strfile=filename;
   sheets = sheetnames(strfile);

   [filepath,filedetail,ext] = fileparts(filename);
   list_filepath{end+1} = filepath;
   list_filedetail{end+1} = filedetail;
```

```
T_active = [];
for s = 1:numel(sheets) % loop through all sheets in the same file
if s>1
   opts_sheetPrevious = opts_sheet;
end

opts_sheet = detectImportOptions(strfile,'Sheet',sheets{s},'PreserveVariableNames',1);

% Remove duplicates from sheet 2 and above, if they data has been
% represented in a previous sheet.
if s>1
   % find duplicates, to remove from second sheet
   k=1;
   while k<=numel(opts_sheet.VariableNames)
      if sum(opts_sheet.VariableNames{k}==string(opts_sheetPrevious.VariableNames))>0
         opts_sheet.VariableNames(k)=[];
      else
         k=k+1;
      end
   end
end

charIdx=string(opts_sheet.VariableTypes)=='char';
charNames=opts_sheet.VariableNames(charIdx);
opts_sheet=setvartype(opts_sheet,charNames,'string');


%create table
T_active_sheet =
table('Size',[size(opts_sheet.VariableNames)],'VariableNames',opts_sheet.VariableNames,'VariableT
ypes',opts_sheet.VariableTypes);

% get raw data
t_active_sheet = readtable(filename,opts_sheet,'Sheet',sheets{s});

% allocate raw data into a single row for active sheet
for k = 1:width(t_active_sheet)
   columnData=rmmissing(t_active_sheet.(k));
   if isempty(columnData)
      if sum(cellTrack==k)>0
         columnData = [];
         T_active_sheet.(k)={columnData};
      elseif sum(cellTrack==k)==0 && sum(stringToDouble==k)>0
         columnData = NaN;
```

```
            T_active_sheet.(k)=columnData;
        end
    elseif numel(columnData)>1
        if isstring(columnData) && logical(mean(not(isnan(double(columnData)))))
            columnData = double(columnData);
            if sum(stringToDouble==k)==0
                stringToDouble(end+1)=k;
            end

        end

        if sum(cellTrack==k)==0
            cellTrack(end+1)=k;
        end
        T_active_sheet.(k)={columnData};
    elseif numel(columnData)==1
        if isstring(columnData) && not(isnan(double(columnData))) % some numeric data was saved
by Python as a string. This fixes it.
            columnData = double(columnData);
            if sum(stringToDouble==k)==0
                stringToDouble(end+1)=k;
            end
        end
        T_active_sheet.(k)=columnData;
    end
  end


  T_active = [T_active,T_active_sheet];

  end % loop through all sheets in the same file
  % Once all sheets have been imported,
  % check to see if filename was stored in the table, in final row.
  % If not, add filepath and filedetail.
  if i==1
    hay_filedetail = 0;
    for checkcolumn = 1:width(T_active)
        if isstring(table2array(T_active(end,checkcolumn))) ||
ischar(table2array(T_active(end,checkcolumn)))
            if 1==numel(table2array(T_active(end,checkcolumn)))
                if contains(table2array(T_active(end,checkcolumn)),filedetail)
                    hay_filedetail = 1; % "hay" is spanish for "there is". "1" is binary for "yes, there is".
                    break % leave this for loop, what is sought has been found
                else
```

```
                hay_filedetail = -99; % to show that the first row was already checked
              end
            else
              hay_filedetail = -98;
            end
          else hay_filedetail = -97;
          end
        end
    end
    % if hay_filedetail is still zero, and no record of the filename was
    % found, then record the full filename and the file detail, in two new
    % columns.
    if not(hay_filedetail == 1) % file name details were not found in the first row of the table. Beware
of mixed directories with files from various stages of development.
        T_active.("File detail") = filedetail;
        T_active.Filename = filename;
    end


    if i==1
        T = T_active;
    elseif string(T.Properties.VariableNames)==string(T_active.Properties.VariableNames)
        T = [T;T_active];
    else % different variable names, different number of columns
        msg= strcat("Your files have columns that differ.", newline(), filename);
        disp(msg)
        % In this case, items are not stored?
        % if class(string), use <missing>
        % if class(double), use NaN
        % if other, use ....?
    end

    fprintf('%d.', i)
    waitbar(i/n_files,handle_waitbar)

end

% If file details were just added because they weren't present in the
% imported data, put the filedetail column first, so that it's easy to read.
if not(hay_filedetail==1)
    T = [T(:,end-1), T(:,1:end-2) ,T(:,end)];
end

close(handle_waitbar)
```

```
fprintf('\n')
cd(directory_script)

%% prep for CSV, remove columns with cells
Tcsv=T;
[rows,cols]=size(T);
c=1;
while c<=cols
   if string(class(Tcsv.(c)))==string('cell')
      Tcsv.(c)=[];
   else
      c=c+1;
   end
   [rows,cols]=size(Tcsv);
end

%% CSV creation
filename_detail = 'wheat2021_SOCEM_directImport';
filenameCSV_withCells =
strcat('T_',filename_detail,'_withCells_',cell2mat(level1names),'_',cell2mat(level2names),'_',date,'.c
sv');
filenameCSV =
strcat('T_',filename_detail,'_',cell2mat(level1names),'_',cell2mat(level2names),'_',date,'.csv');

if savestuff == 'y'
cd(dir_compiledData)
writetable(T,filenameCSV_withCells,'FileType','Text','WriteVariableNames',1);
writetable(Tcsv,filenameCSV,'FileType','Text','WriteVariableNames',1);
end
cd(directory_script
```

Appendix C: Glossary of Terms

Term	Definition
Cell	A portion of a row of crop that is the length of the SOCEM force bar and that has been side hit by the SOCEM.
Edge effect	A phenomenon that causes the mechanical properties of plants along the edge of a plot to differ from the rest of the plot, because these plants experience less competition for nutrients and sunlight.
Flexural rigidity (EI)	A value used to approximate stalk lodging resistance and that is derived from beam bending equations which correlates with stiffness and characteristic size.
Interrow	The space between rows of crops.
Morphology	The geometric factors of the physical form and structure of a biological specimen.
Phenotyping	Collecting data for measurable traits of biological specimens to associate expressed traits with genetic character in the DNA common to the breed (i.e., variety) of a specimen.
Push	A SOCEM push refers to a period of active data collection, as the device is pushed through a small plot of wheat. A side hit is a push. A forward hit is a push.
Side hit	Testing from a side direction, perpendicular to the direction of rows, rather than the standard forward direction, parallel to rows. The outcome is that one force peak is generated for each row contacted.
Slenderness ratio	The radio of the diameter of a stem over the length of the stem.
Small plot	Experimental plots of cereal crops, meant for studying genetic variation.
Stalk lodging resistance	The complex characteristic impacted by many factors with the result that a given genetic variety of a crop will be less likely to break along the length of its stalk.

Appendix D: Table of Equations

Bibliography

[1] B. Shiferaw, M. Smale, H.-J. Braun, E. Duveiller, M. Reynolds, and G. Muricho, "Crops that feed the world 10. Past successes and future challenges to the role played by wheat in global food security," *Food Sec.*, vol. 5, no. 3, pp. 291–317, Jun. 2013, doi: 10.1007/s12571-013-0263-y.

[2] Observatory of Economic Complexity, "Wheat and Meslin," 2022. https://oec.world/en/profile/hs/wheat

[3] International Grains Council, "Market Information," 2022. https://www.igc.int/en/markets/marketinfo-sd.aspx

[4] Y. Luo *et al.*, "Accurately mapping global wheat production system using deep learning algorithms," *International Journal of Applied Earth Observation and Geoinformation*, vol. 110, p. 102823, Jun. 2022, doi: 10.1016/j.jag.2022.102823.

[5] USDA, "All Wheat Acres, United States," *National Agriculture Statistics Service*, Jun. 30, 2020. https://www.nass.usda.gov/Charts_and_Maps/Field_Crops/awac.php

[6] W. F. Schillinger and R. I. Papendick, "Then and Now: 125 Years of Dryland Wheat Farming in the Inland Pacific Northwest," *Agronomy Journal*, vol. 100, no. S3, May 2008, doi: 10.2134/agronj2007.0027c.

[7] Hannah Kammeyer, "Pacific Northwest 2022 Seed Guide." Limagrain Seed Company, 2022. [Online]. Available: https://limagraincerealseeds.com/pacific-northwest/

[8] H. Wenholz, "William Farrer: Australia's Greatest Benefactor," *The Australian Quarterly*, vol. 2, no. 6, p. 91, 1930, doi: 10.2307/20628860.

[9] W. Tadesse *et al.*, "Genetic Gains in Wheat Breeding and Its Role in Feeding the World," *Crop Breeding, Genetics and Genomics*, vol. 1:e190005, Jul. 2019, doi: 10.20900/cbgg20190005.

[10] P. M. Berry *et al.*, "Understanding and Reducing Lodging in Cereals," in *Advances in Agronomy*, vol. Volume 84, L. S. Donald, Ed. Academic Press, 2004, pp. 217–271.

[11] S.C. Salmon, "An Instrument For Determining The Breaking Strength Of Straw, And A Preliminary Report On The Relation Between Breaking Strength And Lodging," *Journal of Agricultural Research*, vol. 43, no. 1, pp. 73–82, 1931.

[12] L. Shah *et al.*, "Improving Lodging Resistance: Using Wheat and Rice as Classical Examples," *IJMS*, vol. 20, no. 17, p. 4211, Aug. 2019, doi: 10.3390/ijms20174211.

[13] H. Matsuyama and T. Ookawa, "The effects of seeding rate on yield, lodging resistance and culm strength in wheat," *Plant Production Science*, vol. 23, no. 3, pp. 322–332, Jul. 2020, doi: 10.1080/1343943X.2019.1702469.

[14] M. Sterling, C. J. Baker, P. M. Berry, and A. Wade, "An experimental investigation of the lodging of wheat," *Agricultural and Forest Meteorology*, vol. 119, no. 3–4, pp. 149–165, Nov. 2003, doi: 10.1016/S0168-1923(03)00140-0.

[15] M. J. Crook and A. R. Ennos, "The effect of nitrogen and growth regulators on stem and root characteristics associated with lodging in two cultivars of winter wheat," *Journal of Experimental Botany*, vol. 46, no. 8, pp. 931–938, 1995.

[16] A. Muhammad *et al.*, "Survey of wheat straw stem characteristics for enhanced resistance to lodging," *Cellulose*, pp. 1–16, 2020.

[17] Y. A. Oduntan, C. J. Stubbs, and D. J. Robertson, “High throughput phenotyping of cross-sectional morphology to assess stalk lodging resistance,” *Plant Methods*, vol. 18, no. 1, p. 1, Jan. 2022, doi: 10.1186/s13007-021-00833-3.

[18] D. Robertson, Z. Benton, S. Kresovich, and D. Cook, “Stalk architecture is a stronger predictor of stalk lodging resistance than chemical composition,” *Biosystems Engineering*, 2022.

[19] D. J. Robertson, M. Julias, S. Y. Lee, and D. D. Cook, “Maize Stalk Lodging: Morphological Determinants of Stalk Strength,” *Crop Science*, vol. 57, no. 2, pp. 926–934, 2017, doi: 10.2135/cropsci2016.07.0569.

[20] D. J. Robertson, Z. W. Brenton, S. Kresovich, and D. D. Cook, “Maize lodging resistance: Stalk architecture is a stronger predictor of stalk bending strength than chemical composition,” *Biosystems Engineering*, vol. 219, pp. 124–134, Jul. 2022, doi: 10.1016/j.biosystemseng.2022.04.010.

[21] P. M. Berry and S. T. Berry, “Understanding the genetic control of lodging-associated plant characters in winter wheat (Triticum aestivum L.),” *Euphytica*, vol. 205, no. 3, pp. 671–689, Oct. 2015, doi: 10.1007/s10681-015-1387-2.

[22] B. C. Helmick, “A Method for Testing the Breaking Strength of Straw [1],” *Agronomy Journal*, vol. 7, no. 3, pp. 118–120, May 1915, doi: 10.2134/agronj1915.00021962000700030003x.

[23] M. A. Willis, “An Apparatus for Testing the Breaking Strength of Straw [1],” *Agronomy Journal*, vol. 17, no. 6, pp. 334–335, Jun. 1925, doi: 10.2134/agronj1925.00021962001700060005x.

[24] P. M. Berry, J. Spink, M. Sterling, and A. A. Pickett, “Methods for Rapidly Measuring the Lodging Resistance of Wheat Cultivars,” *Journal of Agronomy and Crop Science*, vol. 189, pp. 390–401, 2003.

[25] D. Jo Heuschele, J. Wiersma, L. Reynolds, A. Mangin, Y. Lawley, and P. Marchetto, “The Stalker: An open source force meter for rapid stalk strength phenotyping,” *HardwareX*, vol. 6, p. e00067, Oct. 2019, doi: 10.1016/j.ohx.2019.e00067.

[26] L. Erndwein, D. Cook, D. Robertson, and E. Sparks, “Field-based mechanical phenotyping of cereal crops to assess lodging resistance,” *arXiv:1909.08555*, 2019.

[27] Q. Guo *et al.*, “A Non-Destructive and Direction-Insensitive Method Using a Strain Sensor and Two Single Axis Angle Sensors for Evaluating Corn Stalk Lodging Resistance,” *Sensors*, vol. 18, no. 6, p. 1852, 2018.

[28] D. D. Cook, W. de la Chapelle, T.-C. Lin, S. Y. Lee, W. Sun, and D. J. Robertson, “DARLING: a device for assessing resistance to lodging in grain crops,” *Plant methods*, vol. 15, no. 1, p. 102, 2019.

[29] A. Mangin, J. Heuschele, A. Brûlé-Babel, D. Flaten, J. Wiersma, and Y. Lawley, “Rapid in situ non-destructive evaluation of lodging risk in dryland agronomic wheat research,” *Agronomy Journal*, 2022, doi: doi: 10.1002/agj2.21173.

[30] D. J. Heuschele, J. Wiersma, L. Reynolds, A. Mangin, Y. Lawley, and P. Marchetto, “The Stalker: An open source force meter for rapid stalk strength phenotyping,” *HardwareX*, p. e00067, 2019.

[31] C. J. Stubbs, K. Seegmiller, C. McMahan, R. S. Sekhon, and D. J. Robertson, “Diverse maize hybrids are structurally inefficient at resisting wind induced bending forces that cause stalk lodging,” *Plant Methods*, vol. 16, pp. 1–15, 2020.

[32] R. G. Budynas, J. K. Nisbett, and J. E. Shigley, *Shigley’s mechanical engineering design*, Tenth edition. New York, NY: McGraw-Hill Education, 2015.

[33] R.M. Khurmi, *Textbook of Engineering Mechanics*. Ram Nagar, New Delhi: S Chand & Co Ltd, 2010.

[34] Christian Lorbach, Wolfgang J. Fischer, Adriana Gregorova, Ulrich Hirn, and Wolfgang Bauer, “Pulp Fiber Bending Stiffness in Wet and Dry State Measured from Moment of Inertia and Modulus of Elasticity,” *BioResources*, vol. 9, no. 3, pp. 5511–5528, 2014.

[35] C. J. Collins, B. Yang, T. D. Crenshaw, and H.-L. Ploeg, “Evaluation of experimental, analytical, and computational methods to determine long-bone bending stiffness,” *Journal of the Mechanical Behavior of Biomedical Materials*, vol. 115, p. 104253, Mar. 2021, doi: 10.1016/j.jmbbm.2020.104253.

[36] Austin Bebee, “A Model and Device for High Throughput Measurements of Stem Flexural Stiffness in Grains,” University of Idaho, 2020.

[37] R. B. Austin and R. D. Blackwell, “Edge and neighbour effects in cereal yield trials,” *J. Agric. Sci.*, vol. 94, no. 3, pp. 731–734, Jun. 1980, doi: 10.1017/S0021859600028720.

[38] M. Romani, B. Borghi, R. Alberici, G. Delogu, J. Hesselbach, and F. Salamini, “Intergenotypic competition and border effect in bread wheat and barley,” *Euphytica*, vol. 69, no. 1–2, pp. 19–31, Jan. 1993, doi: 10.1007/BF00021722.

[39] A. Hadjichristodoulou, “Edge effects on yield, yield components and other traits in mechanized durum wheat and barley trials,” *J. Agric. Sci.*, vol. 101, no. 2, pp. 383–387, Oct. 1983, doi: 10.1017/S0021859600037709.

[40] C. Stubbs, R. Larson, and D. Cook, “Maize stalk stiffness and strength are primarily determined by morphological Factors,” *Scientific Reports (in review)*.

[41] D. Robertson, J. Cornwall, C. Stubbs, and McMahan Christopher, “The Overlooked Biomechanical Role of the Clasping Leaf Sheath in Wheat Stalk Lodging,” *Frontiers in Plant Science*, p. 1774, doi: doi: 10.3389/fpls.2021.617880.

[42] Kurtis Schroeder and Doug Finkelnburg, “2014 Small Grain and Grain Legume Report,” University of Idaho, College of Agricultural and Life Sciences, Department of Plant Sciences, Cereal Yield Trials Research Bulletin 187, Jun. 2015. [Online]. Available: https://www.lib.uidaho.edu/digital/uiext/items/uiext33195.html

[43] Kurtis Schroeder, Doug Finkelnburg, and David White, “2015 Small Grain and Grain Legume Report,” University of Idaho, College of Agricultural and Life Sciences, Department of Plant Sciences, Cereal Yield Trials Research Bulletin 190, Jun. 2016. [Online]. Available: https://dokumen.tips/documents/2015-small-grain-and-grain-legume-report-small-grain-and-grain-legume-report-.html?page=3

[44] Kurtis Schroeder, David White, and Andrew McGinnis, “2018 Small Grain and Grain Legume Report,” University of Idaho, College of Agricultural and Life Sciences, Department of Plant Sciences, Cereal Yield Trials Research Bulletin 201, Nov. 2019. [Online]. Available: https://www.uidaho.edu/-/media/UIdaho-Responsive/Files/Extension/topic/cereals/north/reports/2018-smallgrain-and-grainlegumereport-approved.pdf

[45] Kurtis Schroeder and David White, “2017 Small Grain and Grain Legume Report,” University of Idaho, College of Agricultural and Life Sciences, Department of Plant Sciences, Cereal Yield Trials Research Bulletin 194, Jun. 2018. [Online]. Available: http://www.extension.uidaho.edu/cereals

[46] F. Galton, “Regression Towards Mediocrity in Hereditary Stature.,” *The Journal of the Anthropological Institute of Great Britain and Ireland*, vol. 15, p. 246, 1886, doi: 10.2307/2841583.

[47] D. S. Moore, W. I. Notz, and M. A. Flinger, *The Basic Practice of Statistics*, 6th ed. New York, NY: W. H. Freeman and Company, 2013.

[48] J. Lee Rodgers and W. A. Nicewander, “Thirteen Ways to Look at the Correlation Coefficient,” *The American Statistician*, vol. 42, no. 1, pp. 59–66, Feb. 1988, doi: 10.1080/00031305.1988.10475524.

[49] D. U. Shah, T. P. S. Reynolds, and M. H. Ramage, “The strength of plants: theory and experimental methods to measure the mechanical properties of stems,” *Journal of Experimental Botany*, vol. 68, pp. 4497–4516, Jul. 2017, doi: 10.1093/jxb/erx245.

[50] Z. Z. Sun *et al.*, “The Viscoelasticity Model of Corn Straw under the Different Moisture Contents,” *Mathematical Problems in Engineering*, 2013, doi: Artn 320207 10.1155/2013/320207.

[51] “EAS Air Suspension Ride Height Sensor LR020159, Left Rear, Original Equipment, For LR3 And Range Rover Sport, 2005 - 2009,” *Roverparts.com*. https://www.roverparts.com/suspension/relays-sensors/LR020159OE/

[52] “Python Pickle Module,” *Real Python*. https://realpython.com/python-pickle-module/

[53] S. de Bossoreille de Ribou, F. Douam, O. Hamant, M. W. Frohlich, and I. Negrutiu, “Plant science and agricultural productivity: Why are we hitting the yield ceiling?,” *Plant Science*, vol. 210, pp. 159–176, May 2013, doi: http://dx.doi.org/10.1016/j.plantsci.2013.05.010.