COMPUTATIONAL BONE MECHANICS MODELING WITH FREQUENCY DEPENDENT RHEOLOGICAL PROPERTIES AND CROSSLINKING

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Biomedical Engineering

by

Timothy Gray Moreno

February 2021

COMMITTEE MEMBERSHIP

TITLE: Computational Bone Mechanics Modeling with Frequency Dependent Rheological Properties and Crosslinking

AUTHOR: Timothy Gray Moreno

DATE SUBMITTED: February 2021

COMMITTEE CHAIR: Scott Hazelwood, Ph.D.
Professor of Biomedical Engineering

COMMITTEE MEMBER: Clifford Les, DVM, Ph.D., MRCVS
Chief Scientific Officer of Pedicaris Research

COMMITTEE MEMBER: Lanny Griffin, Ph.D.
Professor of Biomedical Engineering

ABSTRACT

Computational Bone Mechanics Modeling with Frequency Dependent Rheological Properties and Crosslinking

Timothy Gray Moreno

Bone is a largely bipartite viscoelastic composite. Its mechanical behavior is determined by strain rate and the relative proportions of its principal constituent elements, hydroxyapatite and collagen, but is also largely dictated by their geometry and topology. Collagen fibrils include many segments of tropocollagen in staggered, parallel sequences. The physical staggering of this tropocollagen allows for gaps known as hole-zones, which serve as nucleation points for apatite mineral. The distance between adjacent repeat units of tropocollagen is known as D-Spacing and can be measured by Atomic Force Microscopy (AFM). This D-Spacing can vary in length slightly within a bundle, but by an additional order of magnitude within the same specimen, and can significantly alter the proportion of hydroxyapatite. Previous researchers have built and refined a Finite Element Analysis "Complex Model" to capture the consequences of adjusting D-Spacing and the viscoelastic parameters. This will ultimately serve to elucidate and perhaps predict the mechanical consequences of biological events that alter these parameters. This study aims to further refine the model's precision by accounting for crosslinking between fibrils, the presence of which serves to add mechanical strength. This study also looks to refine the currently used rheological models by way of frequency dependent parameters in the hopes of improving model accuracy over a wider frequency range.

Hormonal factors such as estrogen can significantly determine the composition of bone. Menopause marks a significant reduction in circulating estrogen and has been shown to factor heavily in the development of conditions like osteoporosis. Because

sheep feature a hormonal cycle and skeletal structure similar to humans, three of six mature Columbia-Rambouillet ewes were randomly selected to undergo an ovariectomy, the remainder serving as sham-operated controls. Twelve months later twenty five beam samples were harvested from their radius bones for mechanical analysis and other testing, including atomic force microscopy (AFM) and dynamic mechanical analysis (DMA). The data gleaned from these tests provide an experimental basis of comparison with The Complex Model.

A 2-D Finite Element Analysis model in Abaqus was first created by Miguel Mendoza, which enforced viscoelasticity and a realistic proportion and placement of hydroxyapatite and collagen. The viscoelasticity was modeled using a Standard Linear Solid involving springs and a dashpot element. Crosslinks of varying number and location were arranged within the former model configuration as node to surface tie-constraints to explore the treatment of the FEA Model as a more realistic assembly of parts. Frequencies utilized for this model included 1, 3, 9 and 12 Hz. This approach is referred to in this research as the Intermolecular Forces (IMF) Scheme.

The model was subsequently refined by Christopher Ha and Austin Cummings. The model was characterized by 2x100 unit half-cells, the lengths of which were randomly generated by a Python script. This script ingested the mean and standard deviation D-Spacing length to generate a model geometrically similar to a real specimen bearing those dimensions. A frequency dependent value for the dashpot element in the rheological model used for tropocollagen was developed using this latter FEA model, named the Complex Model. Dashpot values explored for this variable dashpot included 0.0125, 0.125, 0.3125, 0.45, 0.5875, 0.725, 0.8625 and 1.25 GPa-s, some values chosen for their high performance in past studies and others to further narrow the search for the best performing dashpot. All dashpot values were investigated over the previously stated frequencies in addition to 2, 5, 7 and 12 Hz. The best fit dash-

pot values were plotted against the frequencies in which they best performed and a polynomial trend line was fitted to establish an equation, and that equation was used to modify an existing user material subroutine for tropocollagen to provide an automatic frequency dependent dashpot value to Abaqus. This approach is referred to in this research as the Variable Dashpot (VD) Scheme.

Results for the IMF scheme generally performed poorly, with the fully tie-constrained model performing best with 0.77 and 0.024 for R^2 and RMSE respectively. Of the randomized crosslink models, that with the lowest number (N=20) of randomly placed non-enzymatic crosslinks performed best with 0.81 and 0.051 for R^2 and RMSE respectively. Increasing the number of randomized crosslinks reduced model fit, and the remaining three variants exhibited mean R^2 and RMSE values of $0.66 - 0.67$ and 0.052 respectively. For the VD scheme, models running custom modified variable dashpot UMATs yielded R^2 and RMSE values of 0.87 and 0.012 for C2207, and 0.89 and 0.008 for C1809. This is a notable fit considering all other material property parameters are held constant throughout each frequency. In the rheological model, this research also found a striking difference between the frequency dependent viscous element values that made each model perform best. This indicates that differences in D-Spacing standard deviations between OVX and control may be associated with distinct strain-rate dependent mechanical responses.

ACKNOWLEDGMENTS

Thanks to:

- My advisors Dr. Hazelwood and Dr. Les, who were extremely supportive, patient and kind from the very beginning. Without their constant guidance and feedback I'd still be spinning my tires on the early minutia of this project. Dr. Hazelwood, hardly knowing me, vouched for me as a lateral transfer from the College of Science and Math and paved the path for me to take coursework in one of the greatest engineering colleges in the US, to which I owe my current employment as an engineer, where I frequently use many of the skills I learned.
- My wife Miki, for managing everything else so I could painstakingly debug python scripts and cautiously shift click elements in Abaqus over a capricious remote connection for hours on end. Let's take a vacation!
- Miguel Mendoza, Christopher Ha, Austin Cummings and Luke Thompson upon whose shoulders I stood to get this thing done.
- A contingent of sympathetic ME masters students who dragged me by the ear through certain graduate coursework including Matt Ichinose, Alec Henken and Danielle Loi.
- Maaz Hussaini, for taking many hours out of his weekend time to walk me step by step through the entire daunting experimental protocol.
- My military supervisors Nathan Foss, Danielle Hanke and Tyler Merritt for their encouragement, generous scheduling wizardry and irrational faith that I'd eventually graduate.
- Dr. Ashley McDonald, who sent me off with a good word and gave me a superb head start in terminal, python and MATLAB.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Purpose of Study

Bone is a dynamic organ that supports the body in various ways, such as providing mechanical structure, rigid points for ligament and tendon to attach to, storage for Ca^{2+}, and a housing for marrow, which performs the vital function of hematopoiesis [21]. Like many biological materials bone is much more complex than it initially appears, and any attempt at prosthetic mimicry can only hope to fulfill some portion of its functions. This study is of particularly interested in the role bone plays in bodily structural support, and therefore its mechanical properties.

Properties such as strength and stiffness are directly affected by many factors throughout life and development, and these changes can be induced through the vehicles of mechanotransduction and biochemistry. To complicate matters further, bone is an adaptive tissue capable of altering its structure in response to mechanical impetus, a concept which has been coined functional adaptation and is described in the much debated Wolff's Law [49].

> Every change in the form and function of...bone[s] or of their function alone is followed by certain definite changes in their internal architecture, and equally definite secondary alterations in their external conformation, in accordance with mathematical laws. -Julius Wolff

Throughout an individual's development and life, their bone responds to myriad pressures both in and out of their control such as genetics, diet, lifestyle and disease

[66]. While determining the mechanical properties of a bone are somewhat feasible, especially postmortem, establishing reasoning behind the current state of a bone's structure and characteristics is more difficult.

Characteristics such as density at the micro and nanoscale play a large role in the mechanical properties of bone such as Young's Modulus and compressive strength [13]. In the case of cancellous bone, some researchers suspect a cubic dependence of bone stiffness on density as follows [64]:

$$E \propto \dot{\epsilon}^{0.06}\rho^3 \tag{1.1}$$

While this relationship certainly does not hold for all types of bone, it does indicate that even slight changes in apparent density could have significant effects on certain bone under certain loading conditions. Proportions of composition, geometry and abundance of crosslinking are other large factors in mechanical outcome [49]

Osteoporosis, which literally means "porous bone", is a disease characterized by a systematic loss of bone density [31]. It affects roughly 54 million Americans and the National Osteoporosis Foundation suspects that one in two women and one in four men over age 50 will break a bone due to osteoporosis [58]. It is responsible for $19 Billion worth of damage in costs related to broken bones every year, and can severely limit the mobility of those affected [58]. In menopausal women, ovaries become less responsive to Luteinizing Hormone and Follicle-Stimulating Hormone, which serve to regulate key sex hormones such as estrogen, which in turn progressively diminishes in concentration throughout the remainder of life [48]. The most significant factor in accelerated bone loss is the deficiency of estrogen [66]. Further supporting this idea is the presence of bone loss in other conditions associated with premature estrogen deficiency such as anorexia nervosa, secondary amenorrhea and the use of inhibitors

of gonadotropin secretion. Estrogen deficiency can perturb the balance between bone formation and resorption [47] and correlates with an increase in the concentration of parathyroid hormone, further accelerating bone turnover [66].

This study hopes to shed light on the effects that a distinct hormonal change can have on the structure of bone by evaluating it on the nanoscale level through the utilization of Finite Element Analysis (FEA) composed of collagen and hydroxyapatite and experimental data. Significant work has already been accomplished in the development of a two-dimensional FEA model that accounts for the bi-composite, viscoelastic nature of bone. In order to further develop the biological relevance and accuracy of the model, this study aims to account for the presence of crosslinking. Enzymatic and non-enzymatic crosslinking has been modeled before [69], and its inclusion in this model should serve to help it better resemble the experimental results and allow for a better understanding of the consequences of altering bone structure. This research also seeks to fine tune parameters relating to the viscous elements in bone, and to develop a relationship between the behavior of those elements and the rate with which they are strained.

1.2 Bone Tissue Background

As a connective tissue and organ in its own right, the chief function of bone is to bear loads for the body while resisting deformation [49]. As an optimization problem, a synthetic replacement for bone would have to optimize strength and stiffness without introducing too much weight and brittleness all while allowing for self-maintenance, remodeling and repair. To understand how bone manages to accomplish these feats simultaneously, bone needs to be evaluated on multiple scales. It is tempting to guess at the purpose of whole bone from its geometry in terms of the direction and

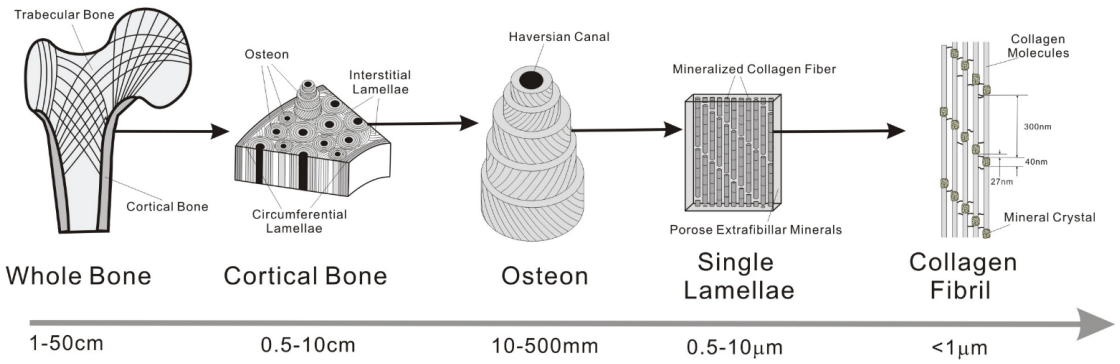

Figure 1.1: Hierarchy of Cortical Bone [27]

magnitudes of the loads it's meant to bear, but this simplistic look belies some of the more interesting functions of bone, like how it manages to contend with those loads over the course of a lifetime.

It is helpful to consider the form and function of bone in millimeters, microns and nanometers separately at first. Bone on the visible scale fits one of two categories: compact and cancellous, two distinct "builds" of bone that will be expanded on later. To the naked eye, bone appears to be quite homogeneous, or even as Frost noted, unified as though "poured into a mold", but a deeper look reveals quite the opposite, as can be seen in the wildly different organizations shown in figure 1.1 [29]. At the micron scale, bone takes the form of osteons, which are about 200 μm in diameter and lengths of 1-3 mm. Osteons compose the fiber portion of the fiber-reinforced composite view of bone [49]. These fibers are often oriented along directions of principal stress, and are separated from their embedding matrix by a cement line, a thin layer of calcified mucopolysaccharides [82]. The presence of this cement line aids greatly in mitigating the advances of fractures and promotes energy absorption because it is a relatively weak interface and succumbs easily to shear.

The nanoscale of bone is of particular interest to this study, and is dominated by the interactions between collagen and mineral. Collagen fibers running alongside

each other are mineralized by hydroxyapatite and connected via enzymatic and non-enzymatic crosslinking [69]. The arrangement of these few constituents can have large effects on every succeeding rung on the hierarchy of bone, and deviations in the quantity or quality of these fundamental building blocks are responsible for several diseases including Osteogensis Imperfecta, which pertains to collagen dysfunction, and Osteomalacia, which pertains to a lack of mineralization [49].

1.2.1 Compact and Cancellous Bone

Compact and cancellous bone have many differences which make sense in light of their mechanical function.

Compact bone (sometimes referred to as cortical or dense bone) often forms the outside of long bones and is typically dominated by lamellar, rather than woven bone. It is significantly less porous (5% to 15%) than cancellous bone and includes several unique structures such as Haversian canals, Volkmann's canals, and lucunar-canalicular networks, as seen in figure 1.2. Haversian canals, named after the English physician Clopton Havers, can be found in the center of each osteon and contain capillaries and nerve fibers. Volkmann's canals, named after the German physiologist Alfred Volkmann, are often orthogonal to Haversian canals, connecting them to each other and further facilitating fluid flow throughout the bone [48]. As seen in the figure 1.2, lacunae are small spaces that contain osteocytes, and each osteocyte features multiple dendritic processes that flow through small gaps called canaliculi which lead to gap junctions. This is suspected to play a role in bone's adaptation to strain via mechanotransduction [82].

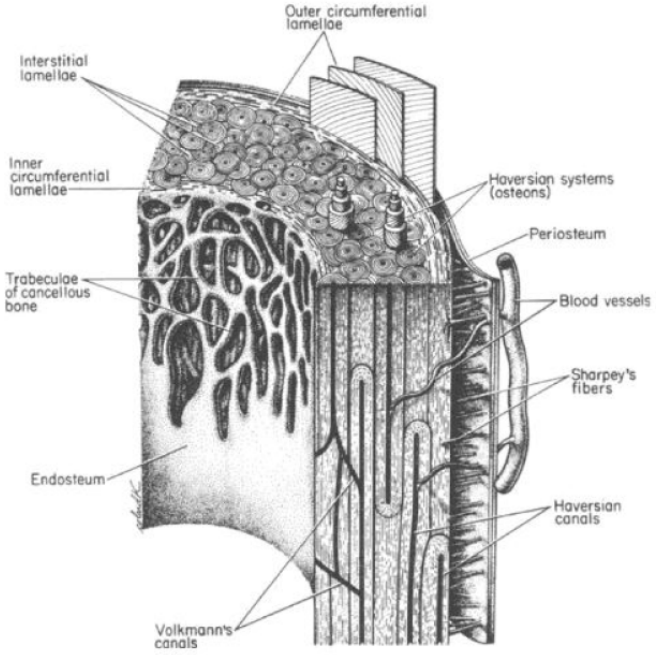

Figure 1.2: Features of Long Bone [49]

Compact bone adds strength to the exterior of long bones, encasing the marrow and trabeculae within. This often correlates to the principal direction of principal strain [49]. Oft-cited work by Reilly, Burstein, and Frankel produced the data shown in Table 1.1, which alludes to a few key ideas about the mechanical properties of compact bone [10]. It can be roughly considered to be a transversely isotropic material, and exhibits greater strength and tensile-compressive moduli in the the longitudinal axis, that is, parallel to the diaphysis of long bones [82]. While most whole bones experience combined loads, that is some simultaneous contribution from compression/tension and an applied moment, it makes sense that bones would be optimized to be strongest along the longitudinal axis in compression. In long bones like the femur, mineralized collagen fibers are nearly aligned with the diaphysis of the bone, along which it experiences most of its loading [25]. It should be noted that in the case of the human femur, the cortical tissue is weakest along the transverse direction in tension, as can

be seen in figure 1.3. Mechanical weaknesses become much more of an issue during mutli-axial loading, such as when experiencing a fall.

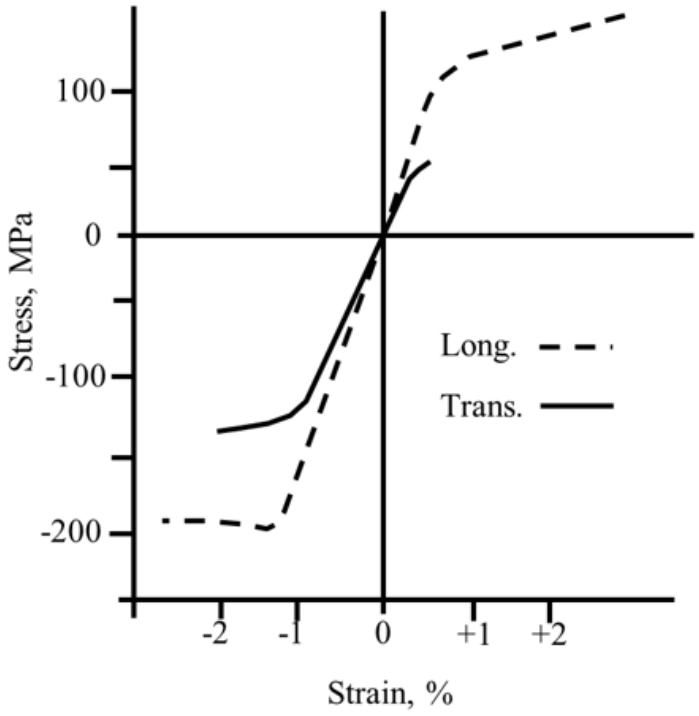

Figure 1.3: Stress-strain behavior of human compact bone [40]

Table 1.1: Anisotropic and Asymmetric Properties of Human Femoral Cortical Bone [10]

Longitudinal Direction	Elastic modulus	17,900 ± 3900 MPa
	Poisson's Ratio	0.62 ± 0.26
	Ultimate tensile stress	135 ± 15.6 MPa
	Ultimate compressive stress	205 ± 17.3 MPa
Transverse Direction	Elastic modulus	10,100 ± 2,400 MPa
	Poisson's Ratio	0.62 ± 0.26
	Ultimate Tensile stress	53 ± 10.7 MPa
	Ultimate compressive stress	131 ± 20.7 MPa
Shear	Modulus	3,300 ± 400 MPa
	Ultimate stress	65 ± 4.0 MPa

Cancellous bone, by contrast, is significantly less dense (more porous) than compact bone, and can be found in the metaphyses and epiphyses of long bones. Cancellous bone allows the body to optimize its load-capacity without adding a significant amount of weight by diverting most of the load to the much stronger compact bone. As mentioned, while it is not uncommon for the porosity of compact bone to be less than 15%, cancellous bone can range from 40% in the femoral neck and 90% in an elderly spine [56]. Cancellous bone looks distinct from compact bone, and while some cancellous bone can include secondary osteons, it is mostly composed of rod and plate-shaped forms called trabeculae which range from 50-300 μm in thickness [82]. The structure of trabeculae varies significantly from anatomic site to site, and trabeculae thin out significantly with age [56]. Figure 1.4 shows two "extremes" of the same anatomic site. Recalling Wolff's Law, some instances of cancellous bone, such as that found in the femoral neck, appears to have itself aligned with the lines of principal stress [49]. While this does seem to support the idea that bone adapts to stresses throughout an organisms lifetime, not all cancellous bone at every anatomic site arranges itself along these lines.

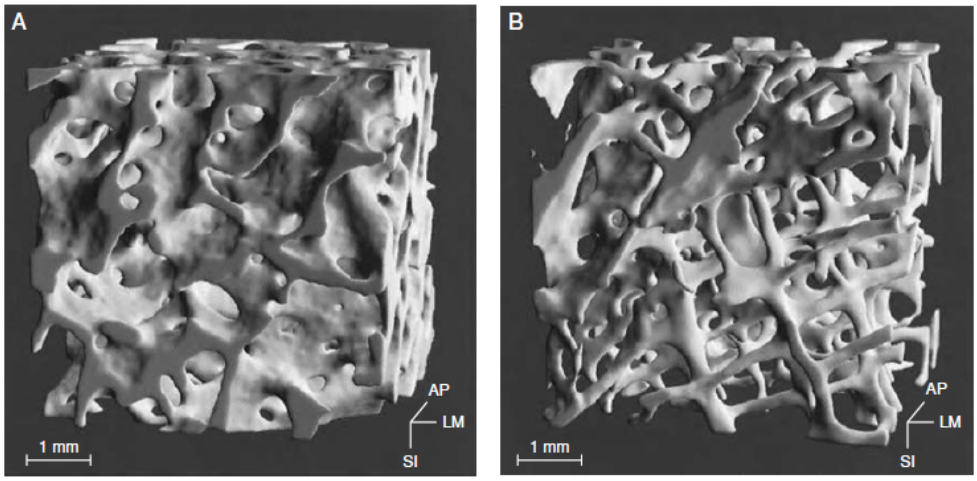

Figure 1.4: Three-dimensional structure of cancellous bone from the iliac crest in a (A) 37-year-old man with no known bone issues and (B) a 73-year-old woman suffering from osteoporosis [57].

More modern studies of cancellous bone seem to indicate that it, like compact bone, is stronger in compression than in tension given the same apparent density (a type of density that includes voids in the material when measuring volume) [39]. The apparent density of cancellous bone is 1.0-1.4 g/cm^3, markedly less than compact bone which is about 1.8-2.0 g/cm^3 [49]. As alluded to in equation 1.1, small changes in density of cancellous bone can spell dramatic results for properties such as stiffness. Generalized mechanical properties of cancellous bone are difficult to state for several reasons, most tied to its variation throughout the skeleton. Cancellous bone mechanics depend on the properties of the bone matrix, the amount of tissue present, and the structural organization of the trabeculae [21]. Table 1.2 reveals that though the elastic modulus can vary greatly among anatomical sites, cancellous bone is generally far less stiff than compact. It should be noted that while compact bone outperforms cancellous bone when considered along the axis of stress, the off-axis arrangement resembles a more isotropic material and allows greater stiffness in many directions.

Table 1.2: Cancellous mechanical properties at various anatomic sites, all tested longitudinally [56]

Anatomic Site (MPa)	Relative Density	Modulus (MPa)	Ultimate Stress (MPa)	Ultimate Strain (%)
Proximal tibia	0.16 ± 0.056	444 ± 257	5.33 ± 2.93	2.02 ± 0.43
Femur	0.28 ± 0.089	389 ± 270	7.36 ± 4.00	Not reported
Lumbar Spine	0.094 ± 0.022	291 ± 113	2.23 ± 0.95	1.45 ± 0.33

As mentioned, bone is a spectacular material not only because of all the things that it allows the body to do, but because of its ability to alter itself over a lifetime of growth and loading. It accomplishes this with the help of a few key cells via processes called Modeling and Remodeling.

1.2.2 Composition of Bone Tissue

It is appropriate to refer to bone as a fiber reinforced composite material featuring mineral and an organic matrix of water and collagen [49]. Water is often omitted in this description but plays an important role in bone tissue's viscoelasticity, and dehydrated bone samples have very different mechanical properties [82]. While there are several methods for quantifying each of these substances individually, the best predictions of mechanical behavior require architectural information. Gravimetric analysis and heating of samples to dehydrate them serve as reliable ways to establish the mineralization of the sample. While the average density of compact bone (ρ_{bone}) can be found with Archimedes's Principle, this is more difficult in cancellous bone where marrow and fat can hide more easily between the trabeculae. In this case, the apparent density ($\rho_{apparent}$) is found by machining out a cylinder or cube of the cancellous bone and dividing the weight of the sample by its overall volume [49]. The porosity (P) is the the percentage of the total sample volume that's not occupied by actual bone tissue and can be found by equation 1.2:

$$P = 1 - \frac{\rho_{apparent}}{\rho_{bone}} \tag{1.2}$$

Porosity is contrasted with bone volume fraction (BV/TV), which is simply the percentage of volume in a sample occupied by the bone matrix. The smallest spaces in bones are generally ignored as potential porosities in the calculation of BV/TV, but Haversian and Volkmann's canals are often included in the bone matrix portion of the BV/TV calculation.

The organic phase of bone mostly consists of type I collagen (98% by weight) with the remainder composed of noncollagenous proteins and cells [82]. Type I collagen is the

most abundant collagen in the body and is not exclusive to bone. Type I collagen is formed from three trimers forming a triple helix–two identical $\alpha 1$ chains and one $\alpha 2$ chain [73]. This triple-helical strand is formally known as tropocollagen, and many tropocollagen fibers together form a collagen fiber. Collagen features much posttranslational modification including crosslinking, hydroxylation, and glycosylation, some of which helps determine its triple-helix arrangement as shown in figure 1.5. As a matter of fact, collagen trimers are bound together by crosslinking [73].

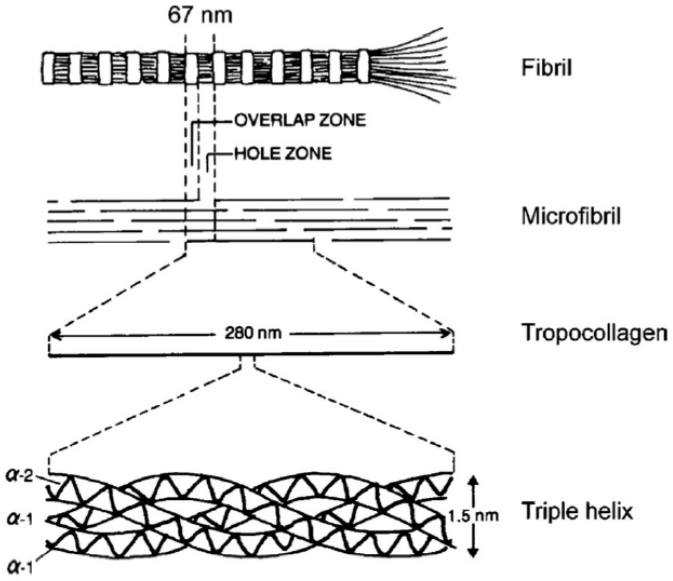

Figure 1.5: Hierarchical Diagram of Type I Collagen in Bone [21].

The mechanical properties of hierarchically arranged microscopic substances such as collagen are difficult to determine, but some mechanical testing has been performed on type I collagen fibrils found in rat tails, concluding a Young's modulus between 3.7 and 11.5 GPa [81]. The authors credit the wide range of results yielded by Atomic Force Microscopy (AFM) nano-indentation to the natural variation in mechanical fibril properties, the calibration of the cantilever, and the varying degrees to which their samples were hydrated [75]. Assumptions were also made about the material being homogeneous, isotropic and linear, which are not entirely accurate [6].

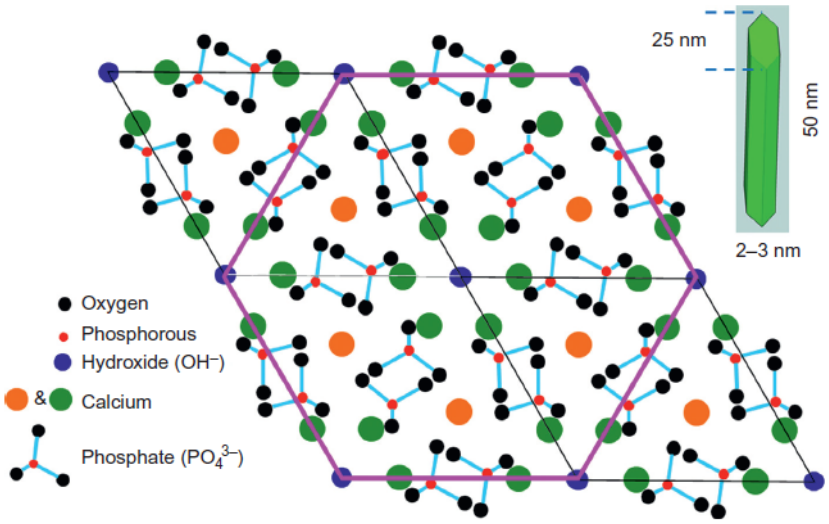

Figure 1.6: Diagram of Hydroxyapatite Lattice [77]

Aside from water, the other major component of bone is its mineral phase, which is almost entirely composed of a poorly crystalline apatite material known as hydroxyapatite, or $(Ca_{10}PO_4)_6(OH)_2$. Hydroxyapatite is found on, around and between collagen fibers within fibrils in a topology further discussed in Section 1.4. Hydroxyapatite, while brittle on its own, has a Young's Modulus of roughly 80 GPa and adds much needed stiffness to bone tissue [17]. Figure 1.6 shows the chemical structure of hydroxyapatite.

Hydroxyapatite is heavily substituted, meaning that many ions other than Ca^{2+} can be ionically bonded, such as strontium, lead, carbonate and fluoride [49]. The degree to which the mineral is substituted is affected by the surrounding tissue and can vary greatly with time, having an effect on mechanical properties such as hardness and brittleness [49]. Mineral has also been found to become more crystalline as it matures, and in tandem with changing ionic substitutions can change such things as lattice solubility, diminish crystal growth, and alter fragility [55].

To achieve the important mechanical properties of bone, especially the strain-rate dependent ones discussed in Section 1.5, the partitioning of organic and mineral phase must be carefully meted. Bone that has too much collagen and not enough mineral will likely resist fracture (to a point) but deform too much to properly transmit loads and provide structure, whereas bone that is too heavily mineralized has greater stiffness but is very vulnerable to the propagation of cracks, which lead to fracture. The sometimes great disparity of this portioning that can be seen within the skeleton, such as in the ear bones vs the femur, often speaks to the various selective pressures that shaped these bones. While collagen is universal among many other phyla, the mineral utilized is much more commonly silica or calcium carbonate, and only chordates (and a few crustaceans) seem to deposit calcium phosphate mineral directly on and around the osteoid (which is essentially bone without hydroxyapatite mineral) to adjust mechanical properties[19]. Most other phyla employ the mineral phase as isolated spicules [19]. Interestingly, arthropods use chitin fibrils bound together by highly crosslinked proteins to produce a composite that performs extremely well for its weight, but cannot be resorbed and must instead be periodically molted [19].

1.3 Bone Remodeling

Much has been said so far of bone's ability to self-regulate by cells responding to mechanical stimulus. Bone undergoes changes via two major processes: modeling and remodeling, though it should be noted that many orthopedists and bone scientists categorize both processes as simply "remodeling" [49]. While both processes involve osteoclasts (cells specialized to resorb bone) and osteoblasts (cells specialized to create bone), modeling is distinct from remodeling and is more associated with bone growth [49]. Modeling mostly happens during adolescence and is associated with the the change of a bone's size and/or shape. As bones grow in length and diameter,

osteoblasts must lay down new bone, but bone must also be removed by osteoclasts to properly shape the bones as they increase in size. For long bones, this resorption is typically applied to the periosteal surface of the metaphysis to size it correctly. For bones such as the skull, resorption occurs on the inner surface just as formation occurs on the outer surface [48]

Bone remodeling, on the other hand, occurs for much longer in the lifetime of an individual and is responsible for the repair of microscopic damage and prevents potential fatigue fracture by clearing out fatigue damage [31]. It does this by periodically removing portions of old bone while simultaneously replacing it with a brand new Haversian system. Remodeling is only accomplished by the coupled behavior of osteoblasts and osteoclasts together, which execute their respective duties as part of a Basic Multicellular Unit or BMU, which can be seen in figure 1.7 [29]. The fact that bone remodeling has been discovered to occur in a discrete, measurable packet has led Parfitt to describe this process as the "quantum concept of bone remodeling" [59]. Remodeling can create what appears to be a ditch on the surface of a bone, which is typical in cancellous bone, or it can bore straight through a bone as is typical in compact bone. BMUs bore out a tunnel of about 200 μm in diameter that can ultimately produce a secondary osteon about 3-9 mm in length over the course of about 4-6 months [49].

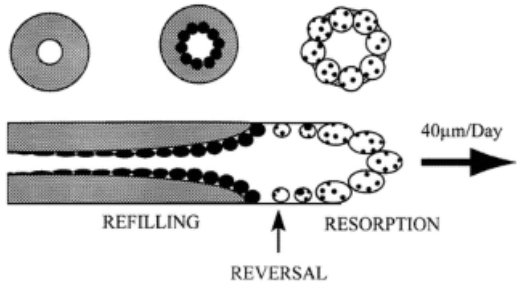

Figure 1.7: Diagram of Osteonal BMU. Darker cells are osteoblasts and spotted cells are osteoclasts [49].

The life of a BMU is described by the A-R-F sequence, which stands for activation, resorption, and formation [49]. Activation takes approximately 3-5 days to occur and involves the fusion of monocytes to create osteoclasts. This is triggered by either a biochemical signal or mechanotransduction. Resorption describes the activity of the osteoclasts breaking down composite bone at a rate of about 40 μ/day, forming a leading "resorption edge" that will eventually be trailed by formation. After a period of time, osteoblasts appear around the rim of the bored-out bone and begin laying down osteoid. This is done in a circular pattern from the outside in resulting in the lamellar patterns seen in Haversian systems [48]. A 40-50 μm diameter channel is left open in the middle called a Haversian canal, which allows for blood vessels, which facilitate nourishment and the transport of essential minerals. After about 10 days, mineralization of the collagen matrix occurs, placing hydroxyapatite within the hole-zones and around the fibers themselves, the entire process taking more than a year.

If there is more bone resorption than formation in a bone it can lead to osteoporosis, and it is possible for BMUs to create a volume of bone that is temporarily osteoporotic considering the lag time between resorption and formation [29]. It should also be noted that it takes significant time to mineralize all the osteoid in a new Haversian system, and that unmineralized bone has its own distinct mechanical properties [49].

1.4 D-Spacing

There are two types of spaces in collagen fibrils which serve to be filled with hydroxyapatite. The first run longitudinally, parallel to the collagen fibers are known as "pores" and are roughly 35 nm in diameter [49]. As shown in figure 1.5, the individual collagen fibers are "quarter-staggered", leaving gap zones known as hole-zones,

which were first discovered in 1942 [25]. As discussed in section 1.2.2, the degree of mineralization can have a serious effect on the mechanical properties of bone–in fact Currey declares that while many factors lead to the mechanical properties of bone, "the main determinant of mechanical properties is mineral content" [17]. Tissue mineralization is thought to positively correlate with stiffness, but negatively correlates with toughness [17].

Just as collagen fibers are arranged into fibrils, collagen fibrils can be considered to be arranged into bundles. Fang et al. considered fibrils to be in the same bundle if they were adjacent and roughly going in the same direction [26]. While the mean D-Spacing in type I collagen fibrils across a tissue is roughly 67 $\pm$ 10 nm in humans, the standard deviation within a bundle itself has been found to be consistently less than 1 nm through AFM investigation [26]. Observing where D-Spacing is similar within a tissue allows researchers to ponder the exact mechanism of fibrillogenesis–the process by which collagen fibers are made. Some factors thought to affect the variation in D-Spacing are substitutions of one amino acid to another, as well as both enzymatic and non-enzymatic crosslinking [26]. As an example of the former, Osteogenesis Imperfecta is mostly caused by a mutation substituting glycine with cysteine within the collagen α helices, causing steric hindrance and preventing proper folding [79]. Crosslinking and other effects are applied after fibrillogenesis however, and it is still a subject of debate exactly when and how D-Spacing is initially determined, but considering the findings of Fang et al. it seems likely that collagen bundles are assembled together, likely by adjacent cells with the same instructions under the same conditions [26]. Another potential question raised by larger D-Spacing between bundles is that perhaps different bundles are meant to serve slightly different mechanical purposes, because bone is after all an anisotropic material that is responsible for frequent, complex loading states [78].

Osteoporosis is a disease that affects millions of people and is characterized hormonally by the abrupt absence of estrogen. When Wallace et al. conducted AFM testing on sheep that had undergone randomized ovariectomy to measure a mean D-Spacing as compared to the control, the difference was noted as "striking" [78]. This finding alludes to the possibility of using AFM analysis from a biopsy to aid in the early detection of osteoporosis, with the assumption that Bone Mineral Density analysis can be incorrect and may exhibit changes before those observed in collagen D-Spacing. This study also establishes a reliable mean and standard deviation in D-Spacing for sham vs. OVX ovine bone tissue for the sake of building our Complex Model, as will be discussed in Section 2.1 [78]. Comparative distributions of D-Spacing measured between the two experimental groups can be seen in figure 1.8.

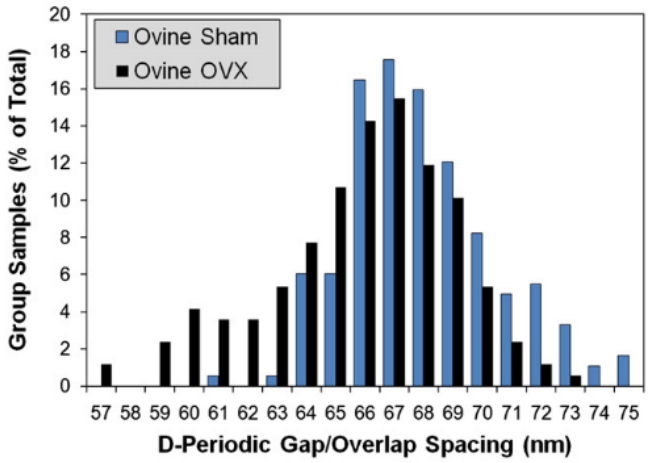

Figure 1.8: Comparative ovine bone D-Spacing distribution between OVX and Sham, indicating a lower mean for OVX samples [78]

Fang et al. also found OVX D-Spacing within bone exhibited a narrower distribution, raising several questions about the mechanism for that narrowing [25]. It is difficult to determine whether the hormonal change brought on by the ovariectomy had selected for a subset of osteoblasts that all produced fibrils of similar D-Spacing, or if instead

the biochemical signal that osteoblasts were getting post ovariectomy had caused some kind of conformity across every kind of osteoblast.

1.5 Viscoelasticity

1.5.1 Creep and Relaxation

Idealized, linearly elastic materials deform to a fixed extent when a constant stress is applied to them. Viscoelastic materials treated to the same constant stress undergo a slow and continuous deformation, a behavior known formally as creep [28]. Creep can occur from any form of stress (axial, shear, combined, etc) and is noticeable in many occurrences, including the fact that human beings are typically slightly taller in the morning than they are just before resting, the slow softening of some seat cushions, or a less obvious example: why the Roman Empire removed chariot wheels when not in use [33].

Strain in this case needs to be decomposed into two parts: the instantaneous elastic strain ϵ^e, which is constant and the creep strain ϵ^c which is variable [28]. The sum of these two parts is known as total strain, and is time dependent. An important parameter in these measurements is the strain rate or $\dot{\epsilon}$, which is easily found by differentiating the creep strain with time. The strain rate is also sometimes called the creep rate. Creep causes total strain to manifest in three phases as shown in figure 1.9: the first phase is a slow straining with a decreasing creep rate, the second a relatively linear strain region, and the third an accelerated rate due to necking. As mentioned, viscoelastic properties including the strain rate can be greatly affected by temperature.

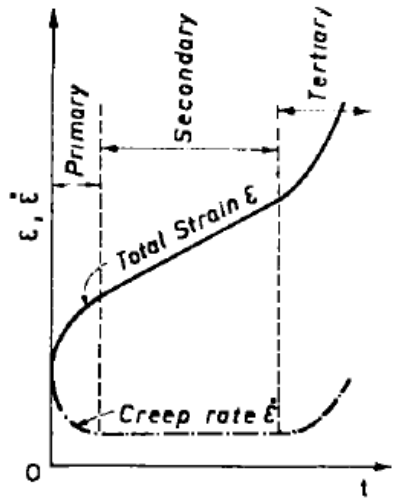

Figure 1.9: The Three Typical Phases of Viscoelastic Creep for the Strain and Strain Rate [28]

When linearly elastic materials are deformed to a constant strain, the stress too stays constant, as in a cold steel cable strained well within its elastic limits. Viscoelastic materials however exhibit something called relaxation–their stress reduces over time in a constant strain, as seen in figure 1.10 [28]. An extreme example of this could be stretching putty or pizza dough. Guitar strings also provide a good example, especially those in classical guitars made of nylon. The frequency produced by strumming a guitar string is a function of the length and the tensile stress in the string, and the latter relaxes in spite of being strung at a constant strain, requiring the strings to be re-tuned over time.

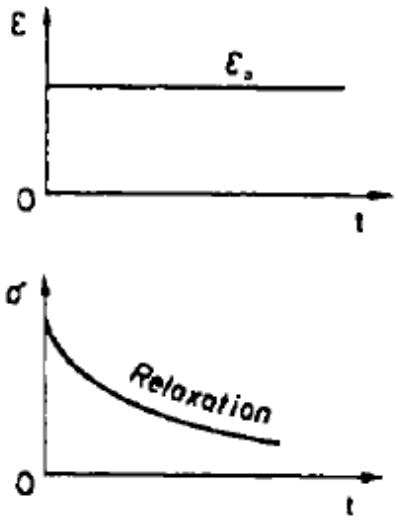

Figure 1.10: Stress Relaxation: The gradual decrease of stress under constant strain [28]

It should be noted that creep and relaxation can happen independently or simultaneously, the latter likely being the case for biological materials such as collagen. The combination of these concepts leads to interesting experimental responses and complex formulas that attempt to capture them.

1.5.2 Measuring Viscoelasticity

While static bending and torsion tests can determine familiar elastic properties such as the elastic and shear modulus of many materials, other means must be employed to test complex materials that exhibit creep and relaxation. Creep and relaxation can be observed for certain materials over the course of many seconds, hours days or even years, but experiments that involve measuring stress responses over much shorter times need to be explored via oscillating loading [28]. One popular experimental method for assessing viscoelasticity is known as Dynamic Mechanical Analysis (DMA), which allows oscillating loading across a range of frequencies. DMA is com-

monly used for plastics and metals, but has also been shown repeatedly to be a valuable tool for testing compact bone as well [83] [44].

Three important results of DMA experiments are known as the storage modulus E_1, the lesser used loss modulus E_2, and the loss factor tanδ. They are described and derived below and paraphrased from Findley et al [28].

If a constant amplitude (F_o) oscillatory force is applied to a material at a constant angular frequency (ω) it can be described by Eq 1.3.

$$F = F_o \cos \omega t \tag{1.3}$$

If the material is being vibrated continuously and only displaces in one direction via a single mode of movement, a variation in stress at a given point in the material can be described by Eq 1.4, where σ_o is the stress amplitude and ω is the frequency of the vibration.

$$\sigma = \sigma_o \cos \omega t \tag{1.4}$$

If the material is truly viscoelastic, the strain response will also be sinusoidal, share the same frequency as the stress but lag behind it by a phase angle δ, shown by the strain response in Eq 1.5.

$$\epsilon = \epsilon_o \cos(\omega t - \delta) \tag{1.5}$$

Applying knowledge of ordinary differential equations and Euler's formula yields a definition of a complex modulus of the material, E^*. E^* can be decomposed into real and imaginary components E_1 and E_2 respectively.

$$E^* = E_1 + iE_2 \tag{1.6}$$

E_1 is known as the storage modulus, typically thought of as a measure of a material's ability to store energy during oscillatory loading. It can be thought of as an analogue to the elastic modulus of non-oscillatory tests and can be expressed as follows [44]:

$$E_1 = \frac{\sigma_o}{\epsilon_o} \cos \delta \tag{1.7}$$

E_2 on the other hand is known as the loss modulus and indicates a materials ability to dissipate energy, usually in the form of heat. It is expressed as follows:

$$E_2 = \frac{\sigma_o}{\epsilon_o} \sin \delta \tag{1.8}$$

Rearranging these equations gives another expression of the loss factor, which is thought of as a general measurement of a materials ability to dampen oscillatory stress [44]:

$$\tan \delta = \frac{E_2}{E_1} \tag{1.9}$$

Tanδ, also known as the loss tangent, is associated with the internal friction of the material and is simply the tangent of the phase angle of the lag between stress and strain as shown in figure 1.11.

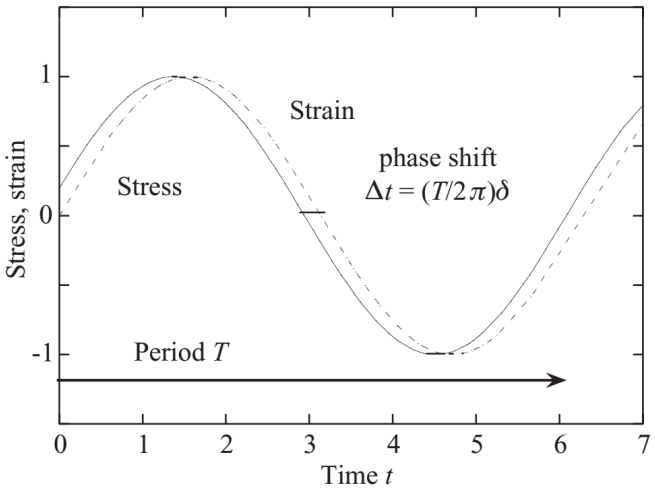

Figure 1.11: Tangent Delta: When applying a sinusoidal load to a viscoelastic material the strain response lags behind the applied stress to an extent described by phase angle δ [43]

A very useful form of Eqs 1.4 and 1.5 that will be used in experimental post-processing scripts include frequency f and are as follows:

$$\sigma = \sigma_o \sin(2\pi f t) \tag{1.10}$$

$$\epsilon = \epsilon_o \sin(2\pi f t - \delta) \tag{1.11}$$

1.5.3 Viscoelasticity and Bone

Bone, like tendon, ligament and cartilage is considered a viscoelastic material because its mechanical properties depend on the rate at which it is loaded [52]. This behavior

is not limited to wet, biological materials however–plastics, wood, concrete and even metals (if the temperature is elevated) exhibit these time dependent properties as well [28]. Carter and Hayes asserted that bone's mechanical properties do not depend on strain rate nearly as strongly as something like apparent density [14], but an increased strain rate does stiffen and strengthen bone to deal with the heightened stresses of physical activity and superphysiological or "traumatic" loading [21]. Courtney et al even found that the strain rate dependence of bone can account for up to 20% of the strength of a human femur when loaded at high frequencies [15]. This concept can also be seen clearly in figure 1.12, which is from a classic experiment on the viscoelastic properties of compact bone [52]. As the strain rate becomes faster, the bone becomes more stiff, but more brittle as well. Martin et al. notes that the energy absorbed to failure is greatest in the range of 0.01-1.0 per second and ponders its implications on function [49].

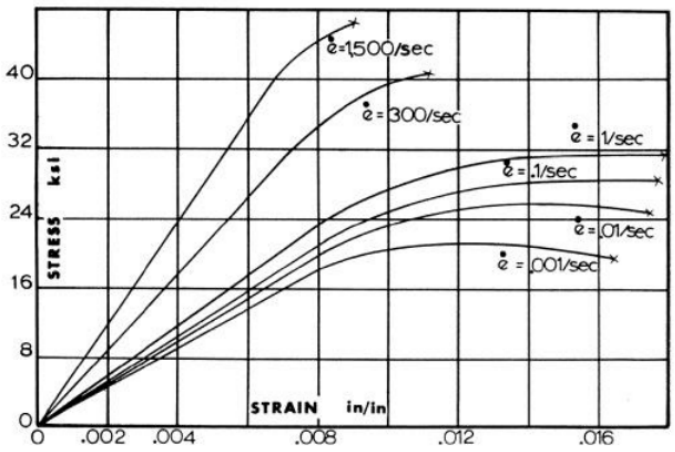

Figure 1.12: The Strain Rate Dependence of Bone: Human cortical bone was loaded at different rates parallel to the osteons [52]

Accounting for viscoelasticity can quickly become complicated–no mathematical formula is absolutely perfect at capturing the stress-strain behavior exhibited by these materials. For example, equation 1.12 is the most basic version of Hooke's Law, very familiar to Physics and Engineering students, and only somewhat capable of captur-

ing the linear segment of a stress-strain behavior in certain materials. It models the material being stressed as though it were a simple spring with a "spring constant" E, also known as Young's Modulus. It has no accounting for strain rate at all, and doesn't hold true for anything other than a linear elastic solid [28]. This equation represents a spring that exhibits instantaneous elasticity and instantaneous recovery, meaning that the strain response to applied stress is immediate and that the strain response to the stress removal is also immediate [28].

$$\sigma = E\epsilon \tag{1.12}$$

Many materials cannot be modeled in this way simply because their strain behavior is time dependent. To help capture the effects of time dependence, a linear dashpot model is employed. A dashpot can be roughly thought of as a medical syringe–steadily and slowly applying force to the injector requires much less effort than applying it quickly. To be more exact, when a dashpot is strained at a constant rate as a result of a constant stress, but when it is subjected to an instantly applied constant strain, the stress will spike up, and then gradually subside to zero as shown in part (e) of figure 1.13 [28]. Equation 1.13 shows a simple formula for a linear dashpot, stating that the stress is proportionate to the strain rate times η, also known as the coefficient of viscosity.

$$\sigma = \eta\frac{d\epsilon}{dt} = \eta\dot{\epsilon} \tag{1.13}$$

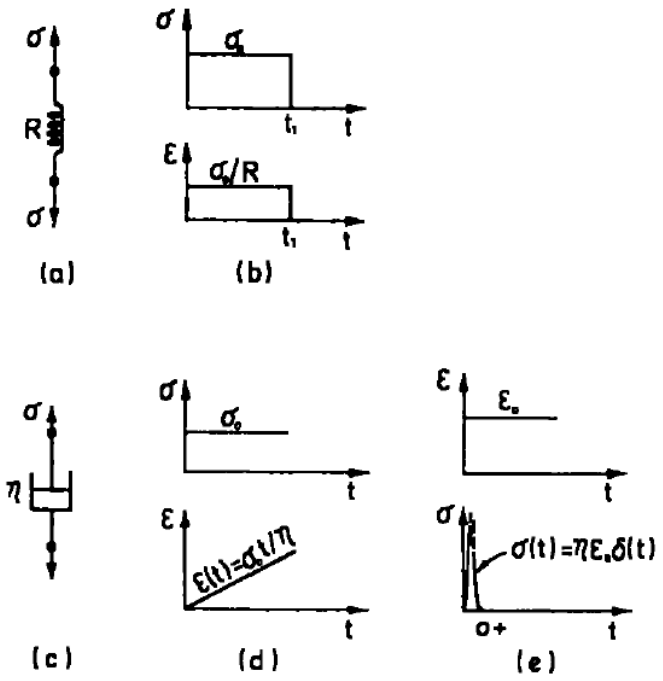

Figure 1.13: The Linear Spring and Dashpot: (a) shows a linear spring model represented by Hooke's law, with (b) showing its instantaneous elasticity and recovery. (c) shows a linear viscous dashpot model, with (d) and (e) showing the constant strain rate resulting from a constant stress, and stress relaxation resulting from a constant strain [28]

Separately, springs and dashpots are insufficient to capture the behavior of real life biological materials like collagen and cartilage but can be combined to form somewhat accurate rheological models for such materials. Springs and dashpots can be added in parallel or end to end in a series, each producing different responses. For collagen, this particular study has found that the Kelvin-Voigt version of the Standard Linear Solid offers the least amount of complexity to sufficiently capture the behavior of the materials in question, including creep and stress relaxation.

1.5.4 A Variable Dashpot

The three previous theses have all speculated on the potential need to investigate what they called a "variable dashpot" [35][16][74]. Discussed more in the section 2.4.3.2, the viscoelastic behavior exhibited by bone can be characterized by rheological models that relate stress and strain. In general, these models feature purely elastic "springs" and strain-rate dependent viscous "dashpots" in various quantities and configurations [28]. Several dashpot values have been explored, and while a few of these values are clear winners for most of the frequencies investigated, values of an entirely different order of magnitude perform far better at lower frequencies.

Thompson ran a very extensive number of tests on the Complex Model just adjusting the rheological parameters used by the Complex Model to find not only the best performing values, but to also monitor the effects of each tweak on Tangent Delta [74]. Thompson found no clear, linear relationship when adjusting tropocollagen's elastic moduli values on tangent delta, and therefore no clear relationship to either the loss or storage modulus (if the elastic moduli varied with just one and not the other, tangent delta would be steerable).

The previous is true because the rheological model for the viscoelastic behavior of tropocollagen used in these theses involves a Standard Linear Solid, which features a lone spring element, as well as a spring element in parallel with a dashpot element to help model the strain rate dependent behavior of collagen. This configuration ensures that the spring and dashpot in parallel experience the same strain, but different stresses. It may be possible to determine a relationship between the viscosity and tangent delta by finding the best performing dashpots at each frequency and establishing a relationship between frequency and the most suitable dashpot value.

It is known that collagen and bone exhibit viscoelastic behaviors but it is not known, to a great degree of certainty, which features in biology one ought to attribute this behavior to, let alone which biological feature corresponds to which specific rheological parameter. It is another aim of this research to characterize the performance of these dashpots and how they differ between specimen to provide clues about how different biology might correlate, whether it be by geometry (D-Spacing), the presence of hormone (or lack thereof), the degree of mineralization, the presence of crosslinks, etc. Several of the aforementioned differences have been quantified between specimen, and some others could at least be surmised [44].

1.6 Crosslinking

Much has been said about large factors in bone's mechanical behavior like mineral density and porosity, but studies are still exploring the contribution from collagen at the hierarchy of the fibrils themselves, which can have large effects on bone's post yield properties [69]. To explore further, the way that collagen fibrils are connected to each other must be considered. In addition to being "cemented" together with hydroxyapatite, collagen fibrils are often bound together via biochemical posttranslational crosslinks. Crosslinks are simply peptide residues that, given the correct biochemical and spatial conditions, connect with each other and affect the mechanical properties of the bodies they are attached to.

While subtle differences in the amount and placement of crosslinking is of interest to this thesis, it should be noted that some serious disorders can trace their effects to issues with crosslinks as well. Life-altering heritable disorders are often caused by incorrect posttranslational collagen modification that prevents normal occurrences of crosslinking. Ehlers-Danlos syndrome type VI is characterized by the absence of

lysyl hydroxylase, an enzyme that mediates the creation of crosslinks, producing severe symptoms such as osteoporosis and arterial rupture [42]. Overhydroxylation of collagen, on the other hand, can severely affect bone strength and is typical of osteogenesis imperfecta [42]. Incorrect crosslinking is also implicated in cases of Marfan's Syndrome and Cutis Laxa [22]. These are examples of how even though crosslinking occurs extracellularly, that is after the collagen peptides created from ribosomes and ejected from the cell, modifications to particular residues such as lysine just after translation can dictate the type and amount of crosslinks ultimately formed [42].

1.6.1 Types of Crosslinks in Bone Collagen

Crosslinks in bone can be categorized at two different hierarchies: generally enzymatic and non-enzymatic, and within enzymatic immature and mature [23].

Enzymatic crosslinks are ubiquitous in bone tissue and take various forms depending on their level of maturity. Figure 1.14 shows the general progression of crosslinks maturation in collagen. Two enzymes play a prominent role in crosslink formation: lysyl oxidase and lysyl hydroxylase. Lysyl hydroxylase (LH) acts within the cell before the collagen peptides are secreted into the intercellular matrix to hydroxylate lysine residues [67]. One major determination of mechanical properties between genetically identical collagen fibers is the extent to which its lysine residues are converted by LH into hydroxylysine, because that distinction forks into different crosslinking paths [67].

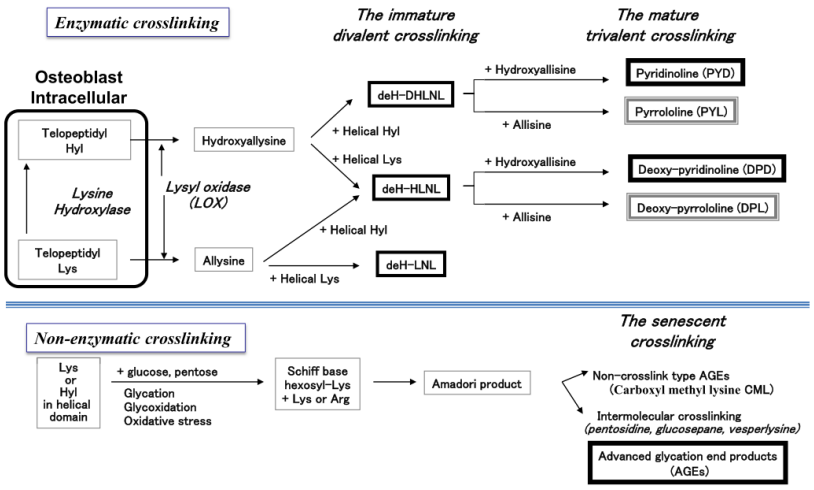

Figure 1.14: The Crosslink Maturation Scheme (DeH-DHLNL: dehydro-dihydroxylysinoleucine, deH-HLNL: dehydro-hydroxylysinonorleucine, deh-LNL: dehydro-lysinorleucine) [67]

It is said that while LH's actions control tissue specific crosslinking, Lysyl oxidase (LOX) controls the overall amount of enzymatic crosslinking [67]. LOX aggregates collagen molecules into collagen fibers by converting certain lysine and hydroxylysine residues within the telopeptide domains of said collagen molecules after they leave the intracellular space. More specifically LOX catalyzes the oxidative deamination of the ϵ-amino group of lysine and hydroxylysine [11]. Vitamin B6 and tyrosyl-lysine quinone serve as cofactors to LOX, and a lack of vitamin B6 has been shown to reduce the number of immature LOX mediated crosslinks. LOX activity was also shown to decrease in mice by 75% 3 days after an ovariectomy because estrogen is a known positive regulator, and this effect was fully reversed by the application of estradiol.[67]. LOX also plays an important wound in tissue healing because it is responsible for forming collagen fibers at injury sites, and can be seen to increase at said sites [11].

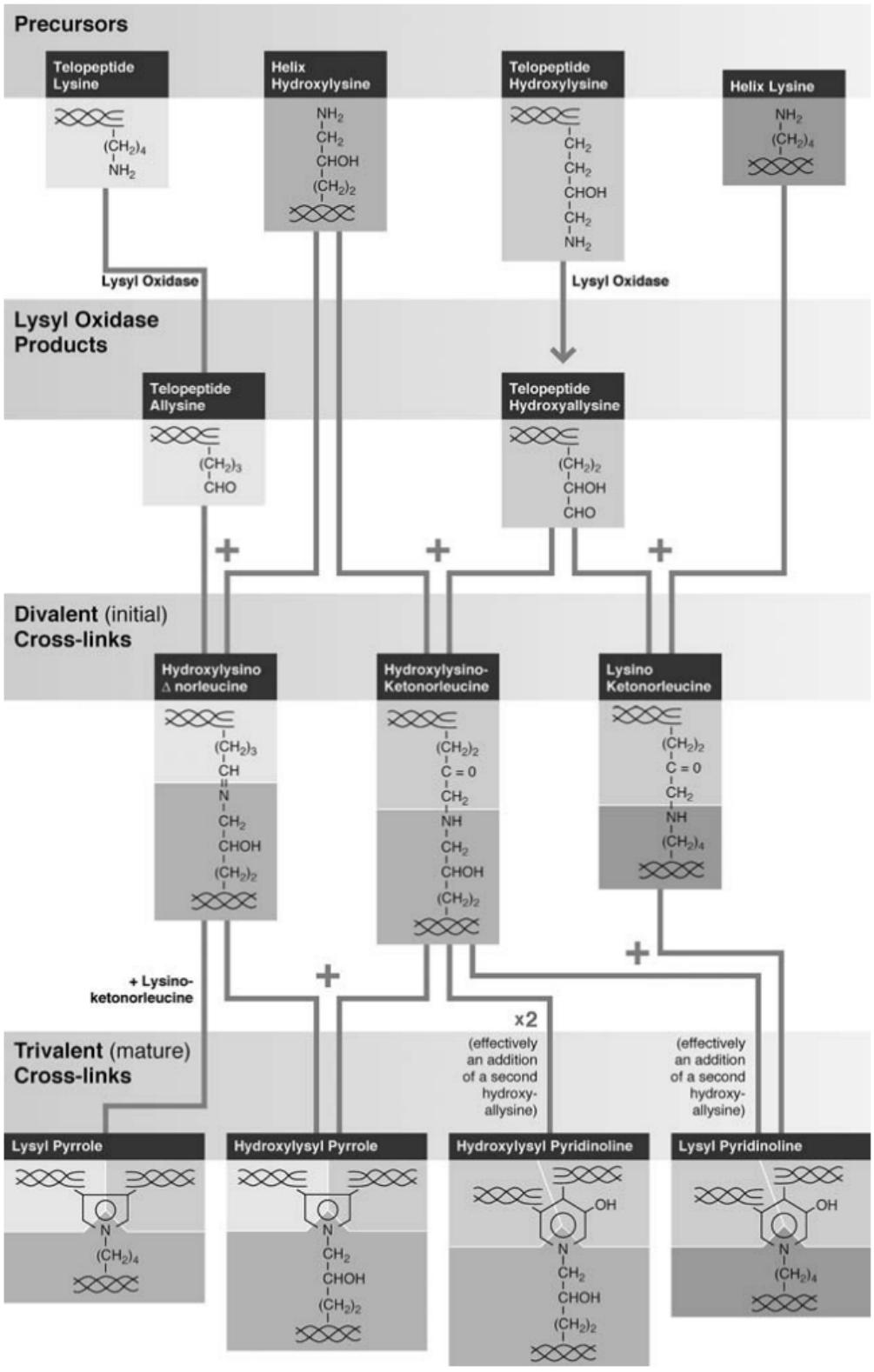

Figure 1.15: The Lysyl Oxidase Pathway for common enzymatic crosslinks in Type I collagen [24]

After the lysine and hydroxylysine residues have been deaminated, the residues now feature relatively unstable aldehyde functional groups, with lysine becoming allysine and hydroxylysine becoming hydroxyallysine as seen in figures 1.14 and 1.15. It should be noted that in tissues that bear large mechanical loads like bone the hydroxyallysine route is far more commonly expressed [22] These aldehyde groups will then react either through Schiff's base formation or aldol condensation to form immature crosslinks with neighboring lysine or other aldehydes, which are shown in the middle column of 1.14 [11]. These pairings are known as immature divalent crosslinks, taking the form of deH-DHLNL, deH-HLNL, or deH-LNL, which have been shown to decrease in number in bone as humans age [67].

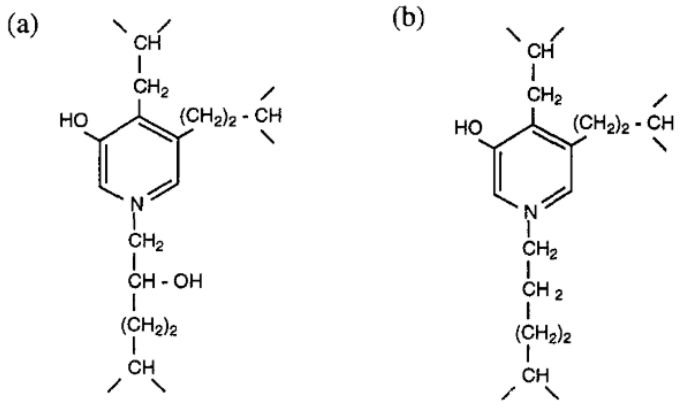

Figure 1.16: Pyridinoline Crosslinks in Bone. (a) Hydroxylysyl Pyridinoline (HL-Pyr, also called PYD), (b) Lysyl Pyridinoline (L-Pyr, also called DPD) [42]

Some portion of these immature crosslinks will undergo further reaction and mature into several possible trivalent crosslinks as seen in figures 1.14 and 1.15. Pyridinoline (PYD), Pyrrololine (PYL), Deoxy-pyridinoline (DPD) and Deoxy-pyrrololine (DPL) are are possible mature crosslinks that appear as a result of various reactive combinations involving existing immature crosslinks and sometimes also adjacent lysine and hydroxylysine residues [67]. These mature trivalent crosslinks generally accumulate in human bone until 10-15 years and remain about constant throughout a person's

life [69]. A common guess as to why this is the case is that mineralization immobilizes collagen molecules and prevents them from undergoing the necessary motion for rapid crosslinking seen in osteoid [42]. The pyridinoline crosslinks, as seen in Fig 1.16, are much more abundant than the pyrroles in bone, and an increased PYD/DPD ratio has been associated with compressive strength and stiffness in bone [5]. DPD is typically on the order of 5 times more abundant in human beings than PYD [5].

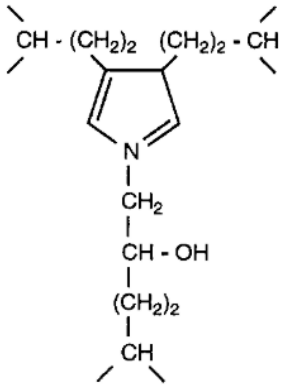

Figure 1.17: Pyrrole crosslink structure in bone [42]

While the pyrrole crosslinks (figure 1.17) are less abundant in mature bone, it has been argued that they are of more mechanical importance, and that an increase in lysyl hydroxylation correlates with a decrease in pyrrole content and bone strength [42].

Crosslinks can also be formed with glucose via the process of glycation, and these are generally referred to as non-enzymatic crosslinks [5]. Bailey describes the glycation of collagen as a more random occurrence than the aforementioned chemical pathways in enzymatic crosslinks [5]. The specific reaction of glycation is the reaction of aldehydes on the open form of glucose reacting with lysine, which happens to be the Maillard reaction, the same reaction that occurs when bread forms a crust. This forms glucosyl-lysine. Bailey notes that while there are many arginine and lysine residues in proteins to provide nucleophilic attack on any glucose aldehydes that

come by, most residues are not glycated, causing one to ponder the site specificity of glycation [5]. While Bailey does field some guesses as to what the criteria may be, Siegmund found the glycation placement to be random enough that they spread glycation derived crosslinks randomly throughout the length of their computational model of collagen [69].

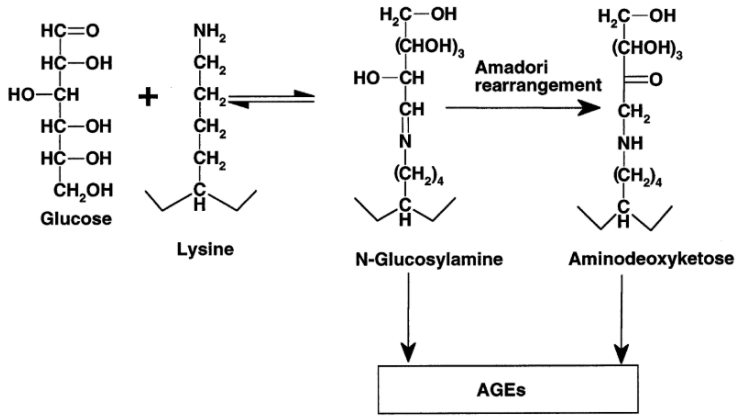

Figure 1.18: AGE Pathway via Glycation [5]

After glucosyl-lysine or N-Glucosylamine is formed, the residue may undergo spontaneous Amadori rearrangement into aminodeoxyketose [5] as seen in Fig 1.18. From there either the glocosyl-lysine or aminodeoxyketose may undergo any number of subsequent reactions to form Advanced Glycation End-Products, or AGEs, which is a catchall phrase for many different final end products that started from glycation, which can include 3-deoxyglucosone, pentosidine, and carboxymethyllysine, among others [5].

Fibrous collagen is subject to the formation of intermolecular glycation crosslinks over time, which makes it less flexible and more resistant to enzyme activity [5]. Because collagen is such a long lived protein, it tends to accumulate AGEs over time and while this has serious effects on the function of collagen in the eyes, kidneys and heart, its effects on the properties in bone may be more complicated [5]. AGEs are inversely

correlated with bone toughness, creep rate and strain to failure in general, with more specific AGEs having showing other specific correlations [71]. Because the appearance of AGEs depends on the presence of glucose, AGEs can accumulate more quickly in diabetic tissues, but also due to increased oxidative stress [67].

1.6.2 Crosslink Geometry

While the exact specificity of crosslinking is an active area of research, some researchers have educated guesses for the likely placement patterns of each type of crosslink [22]. It is this information that ultimately informs the placement of crosslinks in computational models such as our Complex Model, as well as those that Siegmund applied to borrowed models [69] [38].

While many enzymatic crosslinks are located at the collagen termini, most agree that there are an additional two universal sites about 90 residues inward from the telopeptide region, and while some call these sites "helical", it should be noted that collagen molecules are often around 1400 peptides long, so these sites are still relatively terminal, though not so close to the edge that they should appear in Siegmund's model, which only shows the terminal crosslink because by rough calculation the helical crosslink is about 19 nm from the end, which is out of bounds for Siegmund's model [22] [69]. As shown in figure 1.19, "helical" sites are hydroxylysine residues and when collagen is packed into fibrils, these sites align with neighboring collagen molecules in a staggered fashion, producing the roughly 67 nm D Spacing that was introduced in Section 1.4 [22]. The telopeptide sites are aldehydes.

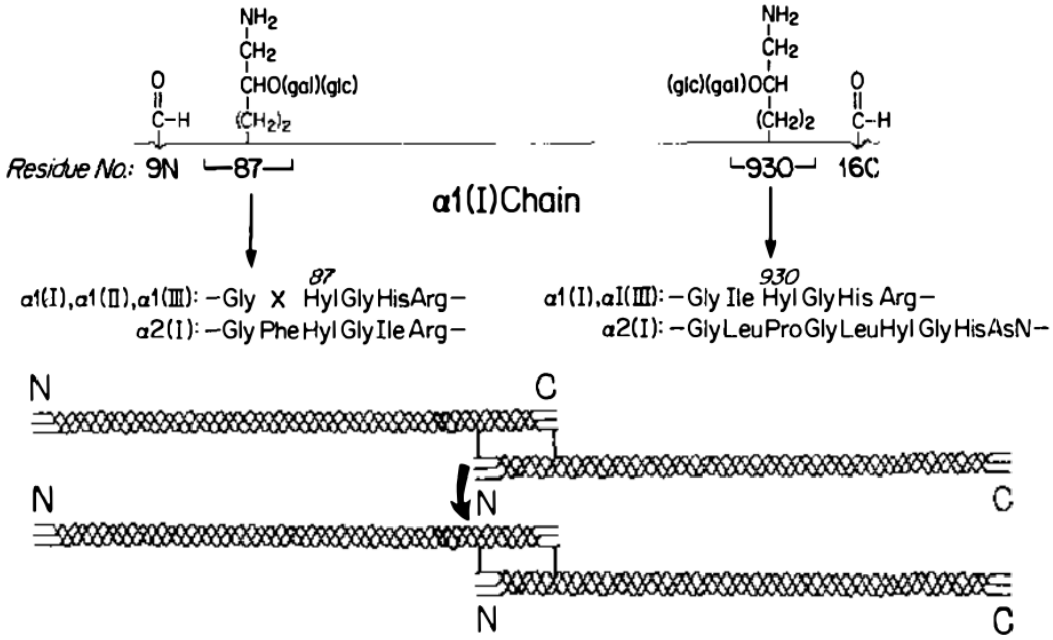

Figure 1.19: The Four Sites: Four common enzymatic loci are universal across collagen molecules of type I, II, and III. The upper portion of the figure denotes their location along the residue and the exact conserved peptide sequences, and the bottom portion illustrates how terminal and helical crosslinks can pair, producing 67 nm D-Spacing [22]

Figure 1.20 shows a common diagram of two adjacent collagen fibers being connected by mature trivalent crosslinks located near the ends of the individual collagen fibers, with non-enzymatic crosslinks randomly scattered throughout [5].

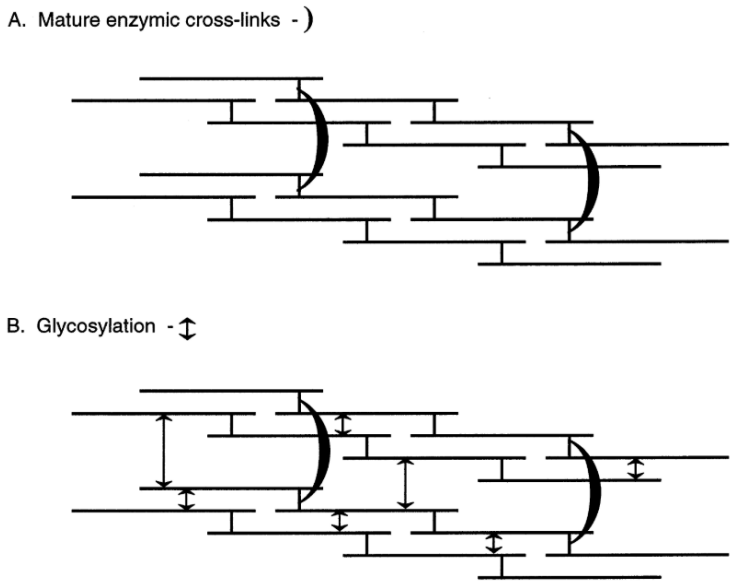

Figure 1.20: Crosslink Spacial Arrangement: A very simplified diagram of the differing arrangements of crosslinks. Immature Enzymatic |; Mature Enzymatic); and Non-Enzymatic ↕ [5]

1.7 Objective

The overarching objective of this research is to develop a version of The Complex Model that better fits experimental data. The Complex Model is an established FEA that models an arrangement of tropocollagen and hydroxyapatite at the nanoscale with loading and boundary conditions that mimic DMA.

This will be a two pronged approach. The first scheme, called the Inter Molecular Forces scheme, will focus on achieving greater fidelity through a more robust accounting of biological interface between tropocollagen molecules and mineral. This will include one fixed position enzymatic crosslink and a variable number of non-enzymatic crosslinks placed at random locations. The second scheme, called the variable dashpot theme, will seek to test a new suite of rheological dashpot parameter values, assess their fitness across eight frequencies, and establish a best dashpot value as a

function of frequency. This relationship will then be injected into a modified version of Richter's subroutine, which will enable superior model performance.

Chapter 2

METHODS

2.1 Model Basis

"All models are wrong" in some way because if they were not, they would be the thing itself. Enough simplifications need to made to allow the model to be tractable, but leaving enough complexity to still bear meaningful and useful resemblance to the subject being modeled. The desired parameters are: a typical arrangement (quarter staggered) of collagen molecules, variable size within reasonable statistical boundaries, and viscoelastic material properties for hydroxyapatite and collagen, the two materials in play. The Complex Model seeks to be a somewhat scalable, modifiable and realistic representation of parallel collagen molecules within cortical bone.

2.1.1 The Petruska and Hodge Model

The advantages of developing such a model have been well known for some time and attempts are not new. In 1964 researchers at Caltech successfully developed a sub unit model for tropocollagen [60]. This model would shed light on the conserved composition within the collagen molecule subunits, revealing that α_1 and α_2 are composed of subunits σ_1 and σ_2, respectively, and that these sub units came together in a ratio of 7:5 to allow the tropocollogen macromolecule to terminate in unison as seen in figure 2.1 [60].

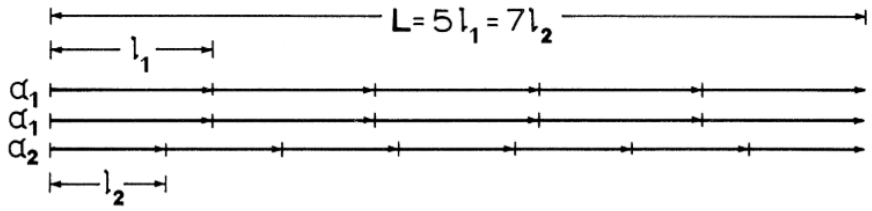

Figure 2.1: Tropocollagen Subunit Ratios: Petruska and Hodge illustrate how two sub units of distinct length combine in a ratio of 7:5 to allow the different alpha chains to have a common length [60]

Petruska and Hodge attempted to relate the length of each subunit to the D-Spacing within the fibril, which they assume as 690 Å, or 69.0 nm, but the most notable contribution of the model is the conclusion that the use of polypeptide subunits ensures macromolecules of a common length with minimal genetic programming requirements [60].

2.1.2 The Jager and Fratzl Model

Jager and Fratzl then created a mechanical model by adopting some of Hodge and Petruska's findings, specifically the staggering of collagen. They conceived of a model that encompassed mineral and collagen in order to study the positive correlation of stiffness and fracture stress with the degree of mineralization in the collagen matrix [38].

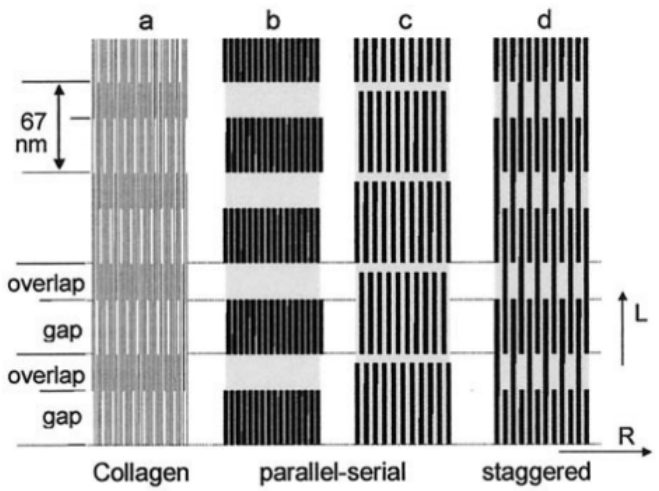

Figure 2.2: Arrangements of Mineral Platelets: Jager and Fratzl demonstrate the various arrangements of mineral platelets allowed by 67 nm D-Spacing. [38]

Jager and Fratzl speculated that the gap regions provide the initial nucleation points for nascent mineral platelets, which are very narrow (about 2-4 nm wide) but potentially as long as 100 nm [38]. In their model (figure 2.2), the stiffness of hydroxyapaptite is assumed to be infinite, an assumption that the Complex Model does not share. Jager and Fratzl concluded that their model was an improvement on existing estimations of mechanical behavior because it accounts for staggering, and that the use of FEA could improve accuracy [38].

2.1.3 The Siegmund Model

Like Jager and Fratzl's model, the Siegmund model attempted to capture the typical two-dimensional arrangements of mineral and collagen at the fibril scale, but took things further by accounting for crosslinking and leveraging FEA [69]. This model is the basis of this study and its various predecessors.

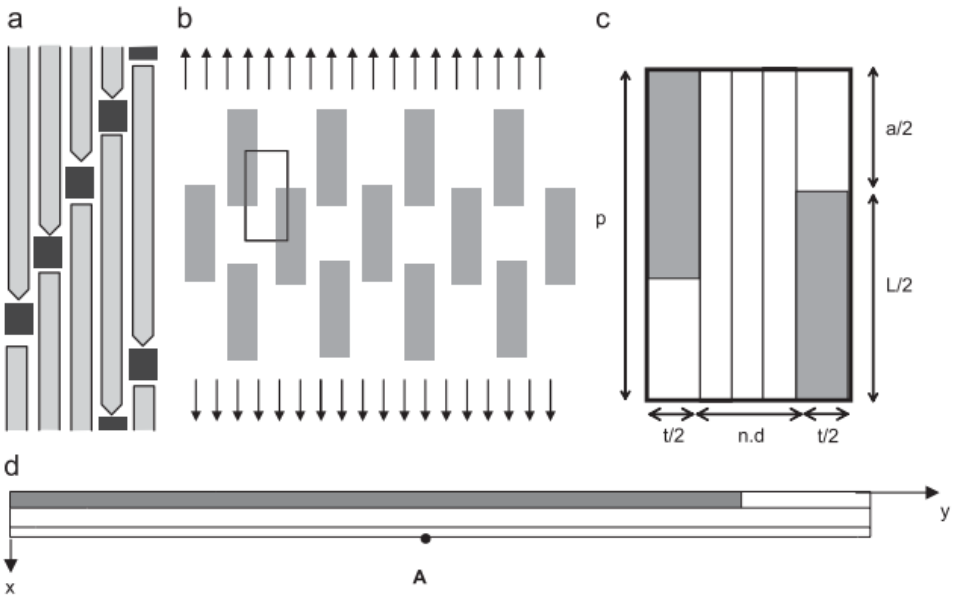

Figure 2.3: Siegmund Computational Model: (a) staggered array model of collagen molecules interspersed with mineral (b) overall view of the matrix in tension (c) a "unit-cell" of period 67 nm and (d) a half-unit cell actually used in computation [69]

This model takes 67 nm as the D-Spacing and uses it to define the period p. Other important geometric aspects can be seen in figure 2.3 and include collagen helix diameter d, number of collagen helices n, the mineral platelet length L, thickness t, distance between between platelets along the fibril a and across the fibril b.

One of several advantages that this model offers is the ability to easily calculate the mineral volume fraction due to the symmetry and consistent length of the unit cell. The mineral volume fraction V_V^m is easily calculated as

$$V_V^m = \frac{Lt}{(L+a)(3d+t)} \tag{2.1}$$

Given the other dimensions are held constant throughout the model, the mineral volume fraction is 0.30 and agrees with assumptions in other work [17] [19]. The spe-

cific remaining geometric parameters used in the Siegmund model were summarized by Luke Thompson in Table 2.1, many of which were adopted by iterations of The Complex Model [74].

Table 2.1: Geometric Parameters of Siegmund Model [69][74]

Dimensions	Variable	Value
Dimensions of Half Unit Cell Model	N	1 x 1
Periodicity	p	67 nm
Number of Collagen Helices	n	3
Collagen Thickness	d	1.5 nm
Short Collagen Length	a	20.1 nm
Fraction of Mineralization	V_V^m	0.3
Mineral Thickness	t	2.5 nm
Mineral Length	L	113.9 nm

As mentioned in Section 1.6.2, Siegmund's model is essentially a study about the effects that different crosslinking has on the mechanical properties of bone, which is why it is highly informative for this research. Before even considering the effects of crosslinking, Siegmund's model defines the cohesive forces at each kind of interface in terms of cohesion energy [69]. Cohesion of the mineral-collagen interface is stated as being mostly due to structural water. Siegmund reasons that given the assumed diameter of collagen, the bond energy of a hydrogen bond, and the number of hydrogen bonds per length of collagen molecule that the cohesive energy between mineral and collagen, here named ϕ_0^{m-c}, is roughly 1.5×10^{-7} J/μm^2 [69].

Siegmund also accounted for the binding forces of crosslinks between adjacent collagen molecules and grouped them as either enzymatic or non-enzymatic, with the former being placed deliberately at the collagen overlap position and the latter being arranged randomly in accordance with the insights of researchers such as Eyre [69] [22]. Crosslinks are modeled as strong local bonds between collagen molecules consisting mostly of C-N and C-C bonds within a single molecule [42]. Based on research

about the mechanical strength of the aforementioned covalent bonds, Siegmund takes the force of rupture to be $F_u = 1.5$ nN and after dividing over the appropriate model area settles on an ultimate stress of 3000 MPa [69][7].

While this model is extremely informative for our own, Siegmund's model is asking different questions and produces a distinct output. While the Complex Model measures viscoelastic parameters such as tangent delta in hopes of correlating with experimental DMA testing, Siegmund's model is interested in localized displacement and possible fracture of a half-unit cell (just half of figure 2.3c). Research goals aside, iterators of the Complex Model like Luke Thompson have noted that such a model would have increased fidelity by adding many more unit cells [74]. Lastly, Siegmund's methods include executing the model with perfectly bonded interfaces only, much like those so far utilized in the Complex Model, and notes a largely linear mechanical response [69]. Figure 2.4 shows a strained mineralized half cell with randomly placed crosslinks, which provided extremely strong bonds in the Siegmund model [69].

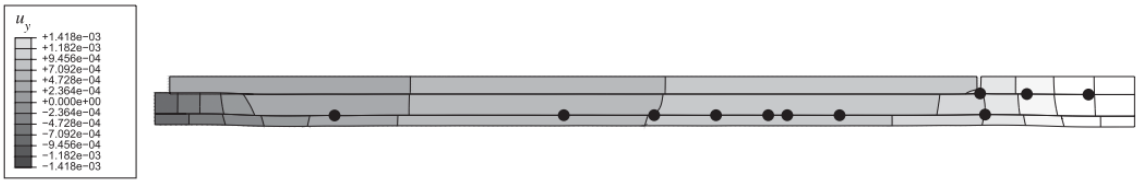

Figure 2.4: Siegmund Model Results: Plot of displacement in the load direction u_y within a strained mineralized half cell with a high level of enzymatic crosslinks (black circles) [69]

2.2 The Complex Model Evolution

Taking insight and inspiration from the aforementioned models, the Hazelwood Research Lab set out to develop the Complex Model beginning with Miguel Mendoza's thesis published in 2013 [54]. The general trajectory of the graduate research projects would follow an effort to more closely match experimental values by adjusting the ge-

ometric size of the model, adding a statistically informed Gaussian distribution to D-Spacing, and adjusting the rheological values and/or formulas of hydroxyapatite and collagen [54][35][16][74].

2.2.1 The Mendoza Model

Mendoza was the first to embark on building a model in Abaqus inspired by Siegmund's in order to analyze the effects of D-Spacing on viscoelastic properties. A base model was chosen to resemble a half-cell with 67 nm D-Spacing as seen in figure 2.5, with additional models created to test alternative D-Spaces of 73 nm and 61 nm based on the distribution observed from the ovine samples [46]. The geometric arrangement of collagen and hydroxyapatite was identical to the "staggered arrangement" identified by Jager and Fratzl in figure 2.2 [38]. Mendoza ultimately employed seven model variants each with different goals, some adjusting periodic unit length, others maintaining mineral dimension, and others maintaining collagen dimension, giving rise to variable mineral volume fraction.

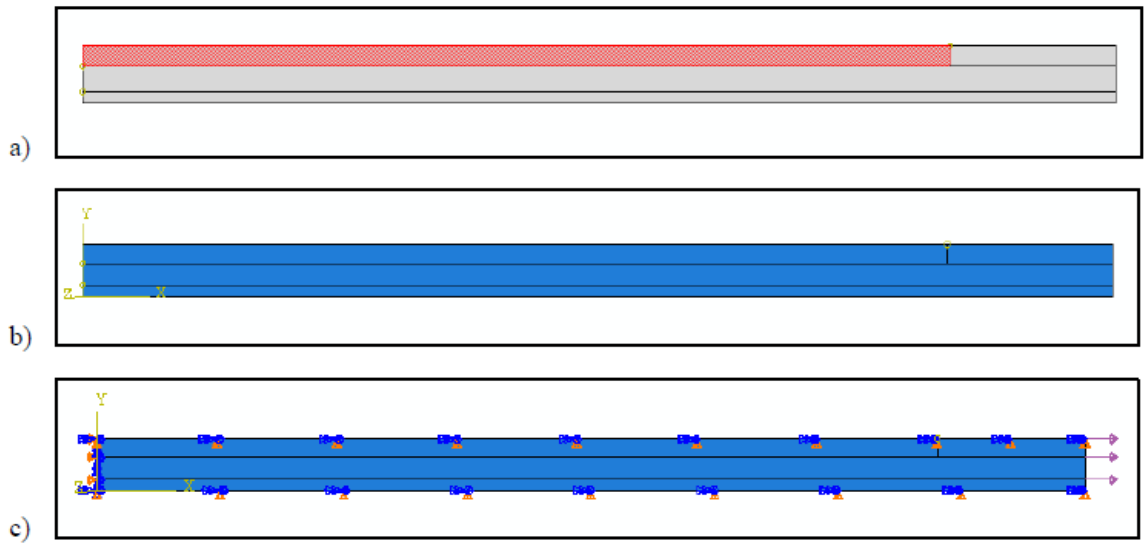

Figure 2.5: Mendoza Model: (a) "Normal" D-Spacing model where the red cell is mineral and the grey cells are collagen. (b) Coordinate axis shown for testing. (c) Sinusoidal tensile loads and symmetric boundary conditions [54]

While Mendoza elected to treat hydroxyapatite as elastic, his most important first contribution was accounting for the fact that collagen is viscoelastic [54]. In order to capture this behavior within the Complex Model, Mendoza adopted research from Frank Richter to write user defined material (UMAT) code for Abaqus that employs the stress and relaxation behavior of a Standard Linear Solid as mentioned in Section 1.5.3 and is pictured in figure 2.6 [65].

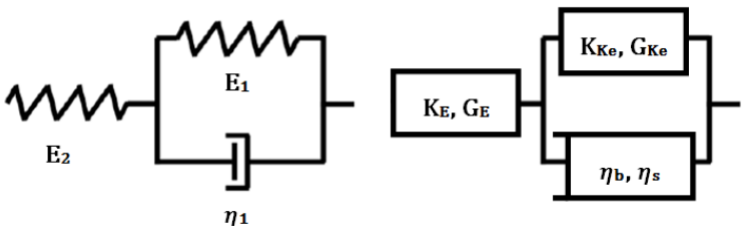

Figure 2.6: Kelvin-Voigt Form Standard Linear Solid: On the left a one-dimensional version of two springs and a dash pot, on the right a three dimensional version that includes shear and bulk moduli [54]

Though the actual DMA testing was conducted from 1 to 20 Hz in 0.2 Hz increments, Mendoza tested and correlated 1, 3, 9 and 15 Hz, which would become standard in subsequent models [54]. Mendoza opted to change a single dimension at a time to identify the Complex Model parameters that agreed with experimental values the most [74][54]. It should be noted that this model and each subsequent model assumed that the interfaces were bonded perfectly together for the sake of simplicity. It was reasoned that the approximation was sensible because DMA testing caused strains that were below yield [54].

Mendoza concluded that D-Spacing did appear to affect the tangent δ of bone, but the experiments highlighted the large effect that small changes in mineral volume fraction could have as well which could be affected by D-Spacing. Mendoza called for improving model accuracy in his Discussion, and suggested the consideration of a 3D model [54].

2.2.2 The Cummings & Ha Model

Austin Cummings and Christopher Ha realized that Mendoza's half-cell model was derived directly from Siegmund's and that the simplicity in geometry made for convenient experimentation, but that a more accurate representation should test the cells interacting in parallel and in series [35][16]. Cummings notes that the reason they thought it made sense to make the model longer is because research shows the periodicity patterns extending for a full 40 microns[54]. Cummings also chose to study cranial specimens, arguing that "there is a precedent for research on this section in tension" [16][35]. Cummings and Ha thus set out to expand the model to fifty interconnected half cells, with two long rows and a hundred columns [16]. To accomplish this they wrote a python script to construct a model directly into Abaqus. Referencing AFM data for the cranial and caudal sections of bone measured by the University

of Michigan, Ann Arbor, Ha wrote the script to vary D-Spacing in accordance with realistic means and standard deviations [35][16]. Utilizing this script produced a total of eight models and were loaded and bound as Mendoza had before, mimicking DMA testing. Collagen material properties utilized Richter's UMATs and hydroxyapatite was modeled as an elastic isotropic solid (E=100 GPa and ν=0.28) [65].

Table 2.2: Cummings and Ha Model Lengths: This table shows the length of each row in all eight of the models. Length discrepancies between rows within a model were remedied by the inclusion of a collagen spacer to make the ends even. [35][16]

Model (Sector Version)	Row Lengths (μm)	
	Top Row, L_1	Bottom Row, L_2
Cranial 1	6.85573	6.86152
Cranial 2	6.86835	6.85995
Cranial 3	6.84309	6.84149
Cranial 4	6.81504	6.85025
Caudal 1	6.63177	6.63619
Caudal 2	6.64287	6.63401
Caudal 3	6.63001	6.28433
Caudal 4	6.63087	6.63515

Because the D-Spacing was variable, the top and bottom rows ultimately terminated in different lengths. To remedy this, their script would automatically add a collagen spacer to ensure that the rows were even within a model for the sake of symmetric loading. Table 2.2 shows the row lengths of each model in microns.

At a glance, the Cummings and Ha model achieved greater agreement with experimental data at lower frequencies, as can be seen in figure 2.7. Because this new model included replicates it allowed the quantification of experimental error. The authors conducted multiple two-way ANOVA testing. They also tested for any confounding interaction between model type and frequency and found that they were statistically insignificant and so their contributions to tangent delta could be assumed indepen-

dent [35][16]. They also cite a larger correlation coefficient when compared with experimental tangent deltas.

Cummings and Ha noted limitations on the study that were largely similar to Mendoza's, but also alluded to the idea that more accurate modeling of the material properties of hydroxyapatite may achieve even greater agreement with experimental results, but lamented that the literature exhibits a great variation in measured elastic properties of the mineral [35][16]. They also suggested a fine tuning of the rheological properties in general.

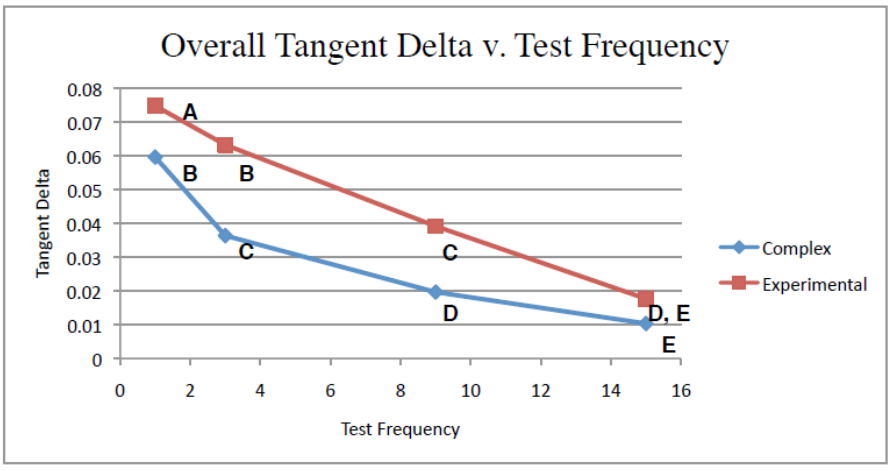

Figure 2.7: Cummings Ha Tangent Delta: Plot showing the interaction between model type and test frequency [35][16]

2.2.3 The Thompson Model

Looking for ways to further refine the model's accuracy, Luke Thompson decided to maintain the new and expanded Complex Model but to more closely scrutinize the individual parameters of the rheological model [74]. Thompson decided that the common literature value of collagen's elastic modulus of 2 GPa should by represented by the effective modulus ($E_{effective}$) as defined in the following equation [21]:

$$E_{effective} = \frac{E_1 E_2}{E_1 + E_2} \tag{2.2}$$

This meant that he assigned values of E_1 and E_2 that would yield an effective modulus of 2 GPa. Thompson ultimately settled on 3 GPa and 6 GPa respectively, noting both that Siegmund's linear elastic model used 5 GPa and that correlating biological significance to individual pieces in a Kelvin-Voigt body was somewhat ambiguous [74][69].

Thompson also ran experiments with an effective modulus of 5 GPa, but the increase reduced coincidence with experimental tangent delta. For future work Thompson suggested exploring lower effective moduli such as 1 GPa with spring element values of 1.5 and 3, noting that collagen values in literature vary wildly, even in similar anatomical contexts [74]. He also speculated that it may help the Complex Model be less conservative in its estimation of Tangent Delta in general to lower the effective modulus, noting this trend in his and preceding studies. He also recommended the exploration of variable dashpot values to help model agreement near lower frequencies.

2.3 Experimental Data

2.3.1 Sample Prep

Sample prep was a combined effort between multiple schools, though it should be stated that no physical experimentation or sample prep took place at Cal Poly San Luis Obispo.

In accordance with the Institutional Animal Care and Use Committee guidance, 12 5 year-old Columbia-Rambouillet sheep were the subject of a joint study involving

the Department of Orthopaedic Surgery at Henry Ford Hospital, The College of Veterinary Medicine and Biomedical Sciences at Colorado State University Ft. Collins, and the Department of Biostatistics and Research Epidemiology, part of the Henry Ford Health System [46]. As noted previously, ewes provide a reasonable biological analogue to human beings particularly for the study of osteoporosis for several key reasons: sheep are larger than commonly used specimen such as mice which makes them more compatible with common surgical procedures, older sheep exhibit Haversian bone remodeling, they have more genetic similarity to humans than other commonly chosen species, and ewes ovulate spontaneously and have a sex hormone profile similar to that of women [85].

The last similarity is especially important in consideration of those factors discussed in Section 1.1, namely that acute, chronic estrogen loss is thought to be the largest factor behind bone density loss during and after menopause and that performing an ovariectomy on humans and sheep causes a hormonal response very similar to menopause [48].

All variables that could be controlled including diet and locale were, with the ewes kept at an altitude of 1600m and fed alfalfa and grass hay. Of the twelve, half were randomly selected for anesthesia and ovariectomy (OVX) and the remaining six were given a sham surgery (Control) [46]. To allow for the effects of hormonal differences to take place, the ewes were allowed to live for one year before they were sacrificed via intravenous barbiturate overdose. The left radius of each ewe was harvested and stored at -20°C.

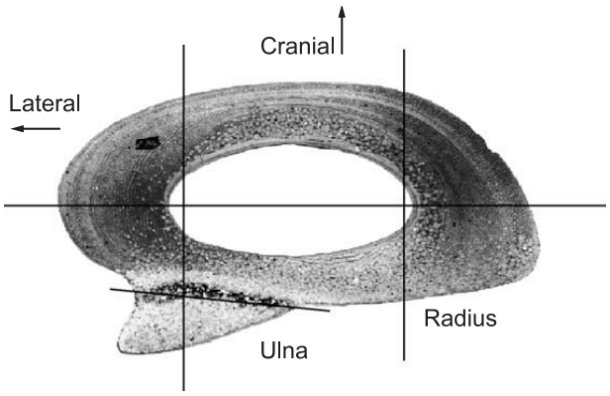

Figure 2.8: Location of Specimens: a ewe's fused left radius and ulna at mid-diaphysis. [46]

After removing the ulna, thin beams (2 x 2 x 19 mm) were taken from six different sections of the radius (craniomedial, cranial, craniolateral, caudomedial, caudal, and caudolateral as pictured in figure 2.8) and stored in saline solution [46]. This was accomplished using Exact Technologies Inc. equipment at Henry Ford Hospital.

2.3.2 Mechanical Testing

One beam from each sector was randomly selected for mechanical testing. Mechanical testing was conducted at Henry Ford Hospital in three-point bending in a 0.9% saline solution on a Perkin-Elmer DMA7a, wherein the cranial side was loaded in relative tension. A static load of 550 mN was tested as well as a dynamic load of 500 mN over a frequency scan from 1 and 20 Hz at 0.2 Hz intervals, but previous studies in the Hazelwood research group have focused on results from 1, 3, 9 and 15 Hz [46]. In vivo stresses were interpreted based on the anatomical location of the beam with cranial sections in tension and caudal section in compression. Because collagen is modeled in tension in the Complex Model, the cranial sector was of particular interest.

2.4 Model Description

In an effort to further refine the fidelity of the Complex Model, this thesis involved a sequence of two approaches. The first approach sought to model the intermolecular forces (IMFs) in a less general way than was done in previous studies and the latter approach sought to rein in on a variable dashpot value mentioned by Luke Thompson [74]. Moving forward this former approach will be called the IMF scheme and the latter the Variable Dashpot scheme. These approaches have very different methods and results, with the most immediate distinction being the respective models they are refining: The IMF scheme is a modification of the shorter, roughly 67 nm Mendoza model, and the Variable Dashpot scheme is a parameter adjustment and optimization of the longer Complex Model developed by Cummings and Ha, the same version of the model against which Thompson did his in depth statistical analysis. This section will bifurcate when distinction is necessary to avoid confusion and will give description to all components typical of a Finite Element Model, including involved material properties, loads and boundary conditions, mesh development, and model validation.

The finite element analysis software used for this research is Abaqus 6.14, both because the flexibility allowed with user written material subroutines (UMATs) and for continuity within the research group. Upwards of 160 testing runs were executed during this research on appropriately equipped computers provided by Dr. Scott Hazelwood.

2.4.1 IMF Scheme

As stated in section 2.1.3, the Siegmund model is a half unit cell model that sought to investigate the relationship between the quantity and placement of crosslinks and

mechanical failures such as delamination between layers [69]. Siegmund's model took care to model multiple intermolecular forces, including hydrogen bonding, electrostatic interactions, and both enzymatic and non-enzymatic crosslinking described in section 1.6. Citing the the same sources found in that section such as Petruska and Hodge and Eyre, Siegmund recognized that in the half unit cell only one enzymatic crosslink would appear in a specific place, and that a number of non-enzymatic crosslinks would occur along the interfaces between collagen molecules [22][60]. Siegmund ran experiments involving no crosslinks, one enzymatic crosslink, and one enzymatic crosslink plus either 1, 5, 10 or 20 non-enzymatic crosslinks. Figure 2.9 shows an example of a model created with 10 non-enzymatic crosslinks.

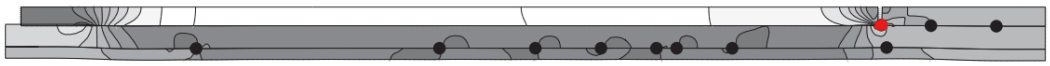

Figure 2.9: Siegmund Crosslink Sites: Black dots represent randomly arranged non-enzymatic crosslink sites, the lone red dot represents a single enzymatic crosslink site. [69]

While Siegmund explored output properties as strength and toughness, this research approach endeavoured to clarify how a more robust accounting of the intermolecular forces would affect the Complex Model's correlation to experimental data. All previous iterations of the Complex Model in this research group, regardless of length, have been treated in Abaqus as a single continuous part. This means that the different sections of the model are really only distinguished by their assigned materials (collagen or hydroxyapatite) and are fused together as one piece for the purposes of stress propagation, a configuration Siegmund refers to as "perfectly bonded" [69].

Mendoza's model very closely resembles Siegmund's but introduces some geometric variability by adjusting its total length to model a few different D-Spacing configurations including 61, 67 and 73 nm, all following the basic layout shown in figure 2.5.

For the sake of simplicity the IMF scheme utilized the 67 nm D-Spacing Mendoza model, but constructed each section out of separate parts to control interaction forces between each. This would provide a starting basis of comparison to established model performance to validate robust intermolecular force modeling as a viable approach.

In a fashion described in later sections, each part was assigned a material, the assembly was meshed and given boundary conditions and an applied sinusoidal load in the exact configuration used by Mendoza for the basis of regression testing. To prevent an FEA phenomenon known as overclosure wherein nodes from different parts erroneously cross over into each other, basic normal and tangential mechanical rules were applied to every surface in the model as seen in figure 2.10.

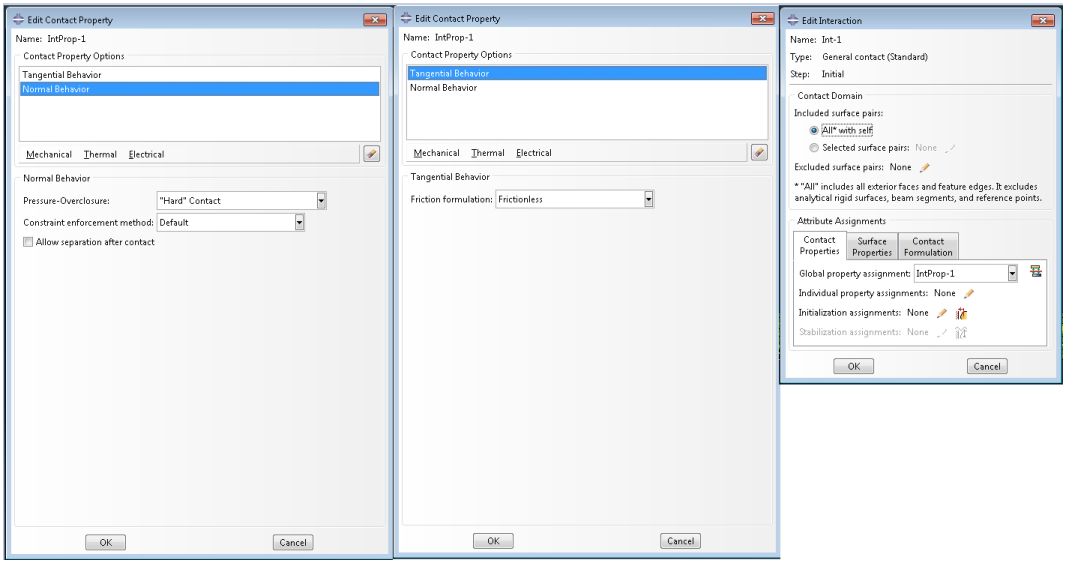

Figure 2.10: IMF Scheme Interaction Properties: General interaction properties assigned to prevent overclosure.

Inspired by Siegmund's method of randomly located non-enzymatic crosslinks, multiple surface and node sets were created in Abaqus to be attached together. While there are several ways to constrain surfaces and nodes together in Abaqus, tie constraints were chosen to represent both kinds of crosslinks, which disallow relative translation between two adjacent nodes, surfaces, or a mixture between the two. Tie constraints

were chosen both for their simplicity, and because modeling failure criteria such as delamination between parts is not an objective of the IMF scheme or the Complex Model, so accounting for things like failure criteria involved in some other constraints was not necessary. It should also be noted that crosslinks are considered very strong bonds, composed of either N-C (347 kJ/mol) or C-C (305 kJ/mol) bonds, which Siegmund estimates to feature an ultimate stress of about 3000 MPa [69].

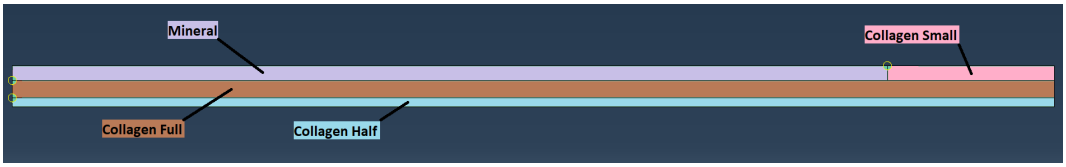

Figure 2.11: Model Part Names: The IMF scheme assembly has four parts named as they are above, which are reused in scripts to specify the location of nodes and surfaces

Parts in the IMF scheme were named as seen in figure 2.11. A surface set known as XLINKBTMSURF was created along the entire top surface of the part Collagen Half, and XLINKTOPSURF was created along the entire top surface of Collagen Full. Node sets were then created along the entire bottom surfaces of Collagen Small and Mineral, named XLINKNODETOP, and along the entire bottom surface of Collagen Full, named XLINKNODEBOTTOM, as seen in figure 2.12.

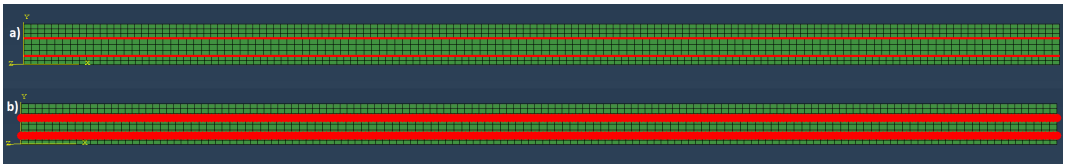

Figure 2.12: Crosslink Tie Constraint Sets: a) Surface sets to accept tie constraints from randomly selected nodes on the opposite surface. b) Node sets containing every candidate node on the surface of a part that could be randomly assigned to have a tie constraint representing a non-enzymatic crosslink.

A job was then created, and from that job an Abaqus input file was generated for the purposes of further manipulation. An input file is a text file that contains all information about an Abaqus job, to include all parts within an assembly, all surfaces, element and node sets, interaction properties, boundary conditions, mesh assignments, loads, output instructions, etc. It is necessarily a large file because it encompasses everything created in Abaqus up to this point, and is all that is required to execute a job from the command line using Abaqus and our user material subroutine.

Expressing the job entirely in terms of plain unformatted text opens up the possibility of manipulation via python script, as fully featured in Appendix E.4. A python script was used to scrape the input file regular expression matches to the names of node sets such as XLINKNODEBOTTOM, ingest the following node numbers, randomly select a set number of them (dictated by how many total non-enzymatic crosslinks were desired), and rewrite brand new randomized node sets and finally inject said sets into the correct location in the input file. The python script also safeguarded against undesired crosslink distribution edge cases (e.g. all crosslinks on top surfaces and none on bottom surfaces) by ensuring a reasonable distribution between top and bottom and partially tying the probability that a node will be selected along a particular surface to the length of that surface so that generally longer surfaces generally have more crosslinks. Figure 2.13 shows a model with 15 randomly placed non-enzymatic crosslinks along the top and bottom interfacial surfaces.

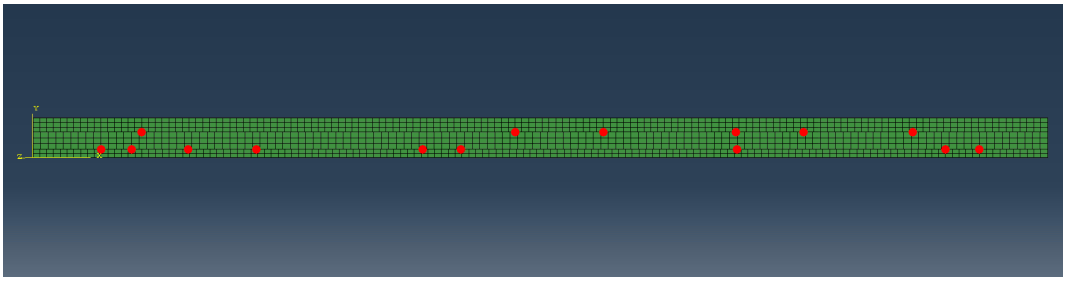

Figure 2.13: Randomized Crosslinks: Red dots represent three different node sets collectively containing a total of N = 15 randomized nodes, which can conveniently pair with adjacent surfaces for tie constraints.

After the python script adjusted the input file with new randomized node sets, Abaqus was relaunched and the model was rebuilt using the input file, now featuring the randomized nodes for the purposes of selection. Tie Constraints were made for each of the three node sets against the surfaces previously created, permanently attaching them there throughout the duration of the job.

While only one fully tie-constrained model was created and tested, for N=20, 25, 30 and 35, three models were generated and tested for each non-enzymatic crosslink count, with their tangent delta performance averaged for each frequency. This was done to prevent skewed data from a model with unusual crosslink distribution. Table 2.3 shows the naming convention used for each randomized node model. Between the tie-constrained model and the randomized crosslink models there were a total of thirteen distinct FEA models tested over four frequencies each with each frequency having been post-processed ten times, producing a total of 520 tangent delta values averaged.

Table 2.3: Randomized Crosslink Model Naming Convention

# Crosslinks	Model Names
20	AddedBC201 AddedBC202 AddedBC203
25	AddedBC251 AddedBC252 AddedBC253
30	AddedBC301 AddedBC302 AddedBC303
35	AddedBC351 AddedBC352 AddedBC353

2.4.2 Variable Dashpot Scheme

The Variable Dashpot Scheme sought to further refine the fidelity of the Complex Model generated by Christopher Ha and Austin Cummings leveraging the findings of Thompson's extensive parameter testing [74][35][16]. As such it was necessarily based on their model which was much longer than Mendoza's (generally about 0.67 microns in length instead of 6.7 nm) and twice as tall, composed of 2 x 100 half cells seen in Siegmund's research. Its geometry is also generated from a python script which, after taking in parameters such as mean and standard deviation of D-Spacing, generates all nodes and elements based on a Gaussian distribution, allowing the collagen fibril to resemble the variability in D-Spacing that it does in literature, which is shown to persist for up to 40 μm in collagen fibrils [25]. This empowers the experimenter to import aforementioned parameters from those recorded in DMA experimental results, resulting in a model that more closely mimics that geometry.

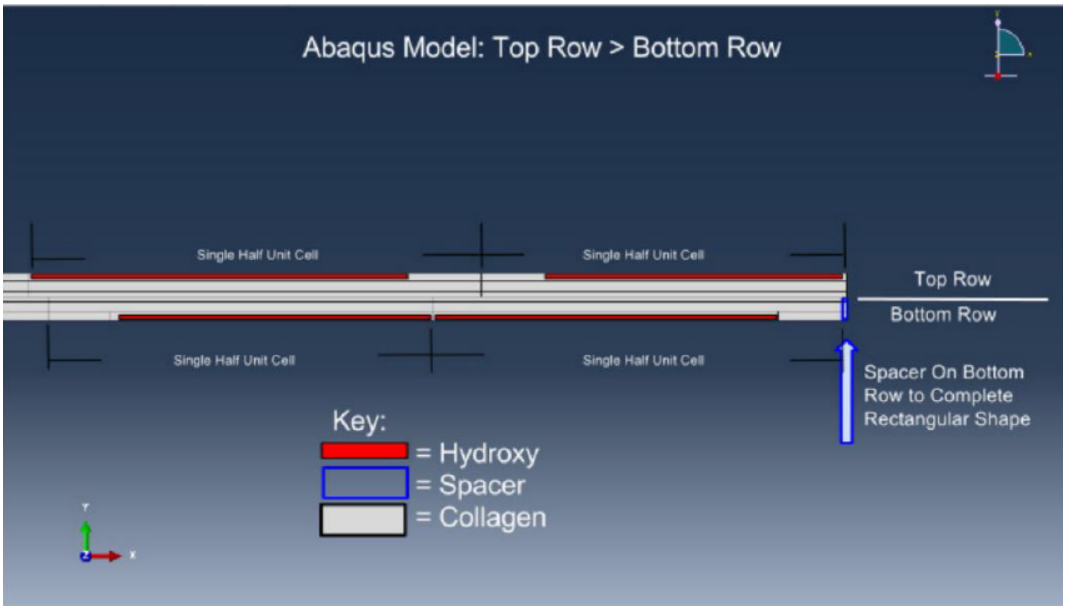

Figure 2.14: Complex Model Composition: This zoomed image of the Complex Model terminus illustrates that it is made up of 2 X 100 half unit cells arranged in two rows which are generated sequentially [74].

Because the model generation script generates the top and bottom rows of half unit cells sequentially and with semi-random dimensions, their ultimate lengths will differ. If that difference is below an acceptable threshold, the model will be deemed biologically relevant, and a tropocollagen spacer will be inserted to bring the lengths to parity, if not, the script will be run again. The spacer can be seen in figure 2.14.

In contrast with the IMF Scheme, this scheme does not require the experimenter to generate an input, manipulate it, and rebuild a model, all remaining segments of the job can be accomplished in the following order before finally generating an input.

2.4.3 Materials

Materials were identically employed for both schemes with the exception of altering dash pot values for the Variable Dashpot Scheme. These models used just two different materials: tropocollagen and hydroxyapatite (sometimes referred to as mineral), as first discussed in Section 1.2.2. Spatial assignment of tropocollagen and hydroxyapatite was deliberate and mimics biology. The materials bear significantly distinct properties, many of which were treated as dependent variables in this and each preceding thesis.

2.4.3.1 Hydroxyapatite

Hydroxyapatite in this and preceding studies is modeled as an isotropic linear elastic solid, which is a solid that obeys Hooke's law in that the stress tensor varies linearly with the strain tensor [28].

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl} \tag{2.3}$$

Shown in equation 2.3 are the stress tensor, the stiffness tensor, and the strain tensor. The stiffness tensor, sometimes called the elasticity tensor, is a 4th order tensor with 81 elements that describes the elastic moduli, which are constant for a given temperature and other environmental variables. Accounting for energy conservation and because all tensors involved are symmetric including the stiffness tensor ($C_{12} = C_{21}$), the total independent constants in the stress tensor sum to 36 for the worst case scenario but much more commonly 21 for general isotropic materials[28][49].

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ & & C_{33} & C_{34} & C_{35} & C_{36} \\ & & & C_{44} & C_{45} & C_{46} \\ & \textit{Symmetric} & & & C_{55} & C_{56} \\ & & & & & C_{66} \end{bmatrix}$$

Figure 2.15: Stiffness Tensor Symmetry: Accounting for energy conservation, off diagonal elements of the 4th order stiffness tensor are equal, significantly reducing independent constants [49].

The number of independent stiffness constants in a material can vary, but symmetry within a material can help cull their numbers further, as seen in figure 2.15. While materials such as bone and wood are anisoptropic, meaning that the relationships between the stress or strain tensor is dependent on the direction of loading, mineral here is modeled as isotropic. Modeling the mineral as isotropic reduces the constants down to just two constants known as Lamé constants, λ and G (the elasticity in shear), which are sometimes used in lieu of more familiar constants such as Poisson's ratio because of they are more tractable in tensor math calculations.

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{2.4}$$

$$G = \frac{E}{2(1+\nu)} \tag{2.5}$$

Examining the previous two equations reveals that to know Poisson's ratio ν and Young's modulus E, is to know the Lamé constants as well. For isotropic materials the stress-strain equation becomes:

$$\sigma_{ij} = \frac{E\nu}{(1+\nu)(1-2\nu)}\epsilon_{kk}\delta_{ij} + \frac{E}{1+\nu}\epsilon_{ij} \tag{2.6}$$

It should be noted that the model is limited to plane strain (strain in only two dimensions), which provides yet more opportunity for simplification in these equations, as seen in figure 2.16.

$$\begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{zz}\\ \sigma_{yz}\\ \sigma_{zx}\\ \sigma_{xy}\end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix}1-\nu & \nu & \nu & 0 & 0 & 0\\ \nu & 1-\nu & \nu & 0 & 0 & 0\\ \nu & \nu & 1-\nu & 0 & 0 & 0\\ 0 & 0 & 0 & 1-2\nu & 0 & 0\\ 0 & 0 & 0 & 0 & 1-2\nu & 0\\ 0 & 0 & 0 & 0 & 0 & 1-2\nu\end{bmatrix}\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ 0\\ 0\\ 0\\ \varepsilon_{xy}\end{bmatrix} \Rightarrow \begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{xy}\end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix}1-\nu & \nu & 0\\ \nu & 1-\nu & 0\\ 0 & 0 & 1-2\nu\end{bmatrix}\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{bmatrix}$$

Figure 2.16: Plane Strain Simplification: Accounting for strain in only two dimensions means that any entries involving the z dimension in the strain tensor are zero, facilitating the removal of the corresponding columns (3, 4, and 5) in the stiffness matrix [50].

These significant reductions eliminate the choice for anything beyond Poisson's ratio and Young's Modulus. While it is difficult to find a biologically relevant (large enough) sample of hydroxyapatite to submit to testing, clues of its material properties in vivo can be gleaned from specimen of highly mineralized bone in the animal kingdom. *Monodon monoceuos*, a whale species, is of particular interest because in spite of having a generally low density skeleton, the rostrum is, by contrast, very dense with mineral. It has been found that the rostrum has an elastic modulus of 31 GPa when

86% mineral and 46 GPa when 96% mineral [86]. These insights provided previous theses in the group with an order of magnitude for selecting the elastic modulus.

The most recent effort identified 36 and 100 GPa as promising values, and after significant testing concluded that while holding the other test parameters that produced the best fit constant, the choice of 36 GPa yielded the lowest Root Mean Square Error (RMSE) value and highest correlation coefficient when compared to other plausible values of mineral elastic modulus as seen in figure 2.17 [74].

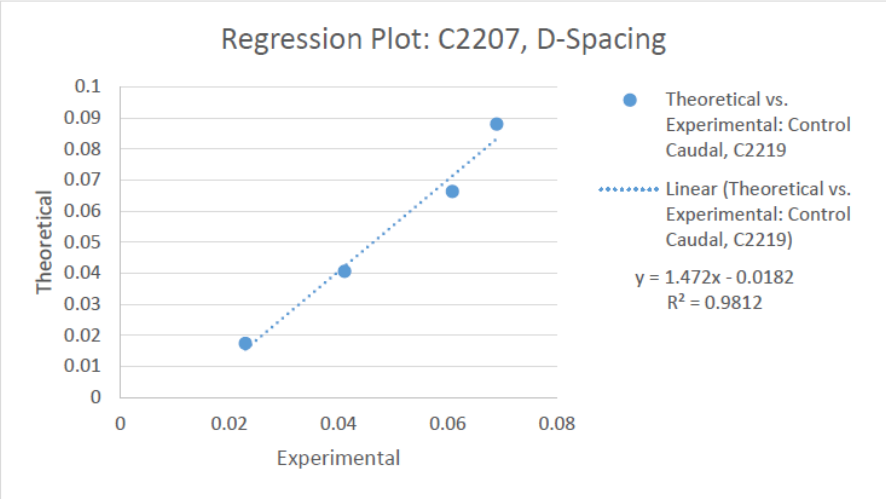

Figure 2.17: Thompson C2207 36 GPa results: A simple linear regression of the Complex Model's tangent delta value while using best fit dashpot values and 36 GPa for mineral elastic modulus (mislabeled as C2219) [74]

2.4.3.2 Tropocollagen

Recall that the main feature of Mendoza's work was to bring a more sophisticated material definition of tropocollagen to the Siegmund model, and that basic rheological framing has been sustained through each subsequent thesis [54]. Mendoza accomplished this by adopting a user-defined material subroutine written by Richter [65].

As discussed in section 1.5, bone exhibits the earmarks of viscoelasticity, such as a rate dependent strain response. Since hydroxyapatite is generally modeled as a linear elastic solid, this viscoelasticity is likely derived from the tropocollagen. There are many choices for modeling creep and relaxation [28]. The rheological configuration utilized here is the Kelvin-Voigt form of the Standard Linear Solid, as pictured in figure 2.6. While it's possible to utilize many more elements, overfitting is an ever present danger and comes at the cost of additional computation time, and a difficulty in attributing biological significance to each additional parameter.

Richter's formulation was three dimensional, but the form of his UMAT that was adopted was one dimensional. Recall that a constitutive equation for a specific material is one that, if all other variables are held relatively constant, relate the stress to the strain [41]. Consider the following three dimensional constitutive equation that relates to figure 2.6:

$$\left(1+\frac{G_{Ke}}{G_E}\right)\sigma_{ij}+\left(\frac{K_{Ke}}{K_E}+\frac{G_{Ke}}{G_E}\right)\frac{\sigma_{kk}}{3}\delta_{ij}+\frac{\eta_s}{G_e}\dot{\sigma}_{ij}+\left(\frac{\eta_b}{K_E}+\frac{\eta_s}{G_e}\right)\frac{\dot{\sigma}_{kk}}{3}\delta_{ij}$$
$$=2G_{Ke}\epsilon_{ij}+(3K_{Ke}-2G_{Ke})\frac{\epsilon_{kk}}{3}\delta_{ij}+2\eta_s\dot{\epsilon}_{ij}+(3\eta_b-2\eta_s)\frac{\dot{\epsilon}_{kk}}{3}\delta_{ij} \quad (2.7)$$

Terms in equation 2.7: σ_{ij} and ϵ_{ij} are the stress and strain tensors. A dot over either term indicates that it's a time derivative of the term (e.g. $\dot{\sigma}_{ij}$ is the stress rate). δ_{ij} is known as the *kronecker delta* and is equivalent to $\mathbf{I}$ or the identity matrix found in Linear Algebra as seen in figure 2.18, because when $i=j$ the kronecker delta equals one, but when $i\neq j$ it equals zero, producing a matrix with ones along the diagonal and zeroes elsewhere.

$$\mathbf{I} = \mathbf{1} = (\delta_{ij}) = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Figure 2.18: Kronecker Delta: The Kronecker delta is equivalent to the Identity Matrix [41]

The subscripts i and j in indicial notation represent the index of a tensor, in this case either the stress or strain tensor. The subscripts kk represent the trace of a tensor. Using the trace of the stress tensor as an example:

$$\sigma_{kk} = \sigma_{xx} + \sigma_{yy} + \sigma_{zz} \tag{2.8}$$

or alternately using numbered indices:

$$\sigma_{kk} = \sigma_{11} + \sigma_{22} + \sigma_{33} \tag{2.9}$$

Again referencing figure 2.6, K and G represent the bulk modulus and shear modulus in the 3D formulation. The subscripts Ke and E indicate the spring element within the Kelvin Voigt body (the parallel configuration in figure 2.6) or the lone spring element. The shear modulus was already mathematically defined in terms of Young's modulus and Poisson's ratio in equation 2.5, the bulk modulus can be similarly formulated:

$$K = \frac{E}{3(1-3\nu)} \tag{2.10}$$

That covers the elastic components, but in the 3D formulation there are also viscous components η_b and η_s, or the bulk and shear viscosity. Here they are formulated in terms of Poisson's ratio and what will be called from this point forward as the dashpot parameter or simply "dashpot" η_1:

$$\eta_b = \frac{\eta_1}{3(1 - 3\nu_{\eta_1})} \tag{2.11}$$

$$\eta_s = \frac{\eta_1}{2(1 + \nu_{\eta_1})} \tag{2.12}$$

Mendoza notes that if the Poisson's ratio for each of the elements and perpendicular components of the stress and strain tensors are set to zero, the 3D formulation simplifies down to one dimension as follows [54]:

$$\sigma + \frac{\eta_1}{E_1 + E_2}\dot{\sigma} = \frac{\eta_1}{1 + \frac{E_1}{E_2}}\dot{\epsilon} + \frac{1}{\frac{1}{E_1} + \frac{1}{E_2}}\epsilon \tag{2.13}$$

Equation 2.13 is completely in terms of the coefficients found on the left side of figure 2.6, coefficients which are a bit more intuitive than the 3D formulation. There are a few more refinements made to this governing equation before it's a functional UMAT recognized by Abaqus, and the code can be seen in Appendix E.2.

It is a task of this research group to choose sensible values for the remaining parameters (E_1, E_2, η_1 and their respective Poisson's ratios): namely those values that provide the greatest correlation with experimental DMA results without running afoul of literature.

The choice for Poisson's ratio for all three Poisson's ratios in the UMAT was set for simplicity as 0.2, the approximate value of hydrated collagen tissue [69].

Values for Young's Moduli both within the body are slightly more complicated, especially because they do not have easily correlated natural features. There are numerous starting points to guess at a good value for in vivo tropocollagen, but how is a single elastic modulus value to be made compatible with two parameters in the Kelvin-Voigt Standard Linear Solid rheological model (Appendix D) [49][6][46]? Mendoza notes that creep and relaxation are both viscoelastic behaviors that are modeled as exponential functions, and when the loading conditions are applied for a sufficiently long time the resultant elastic modulus, if measured at that time, is a combination of both moduli (E_1 and E_2) known as the equilibrium or effective modulus E_{eq}.

$$E_{eq} = \frac{E_1 E_2}{E_1 + E_2} \tag{2.14}$$

This equation allows for multiple configurations of both springs for a single E_{eq}, so for example if literature suggested a modulus of 2 GPa, the choice of 3 GPa for E_1 and 6 GPa for E_2 would equate to the effective modulus and vice versa, but it should be noted that because the springs occupy different locations in the rheological model, the ordering of these values absolutely matters, as preceding experiments confirmed [74]. Mendoza utilized an effective modulus of 2 GPa, and though Cummings and Ha subsequently suggested the exploration of values lower than this, Thompson concluded that a combination of 3 GPa and 6 GPa produced the greatest correlation with experimental results in C2207 and C1809, the sham and OVX cranial samples tested in this thesis [54][16][35][74]. Luke Thompson's results for this treatment can be seen in figure 2.19. For this reason these moduli were chosen for this thesis in both the IMF and Variable Dashpot schemes.

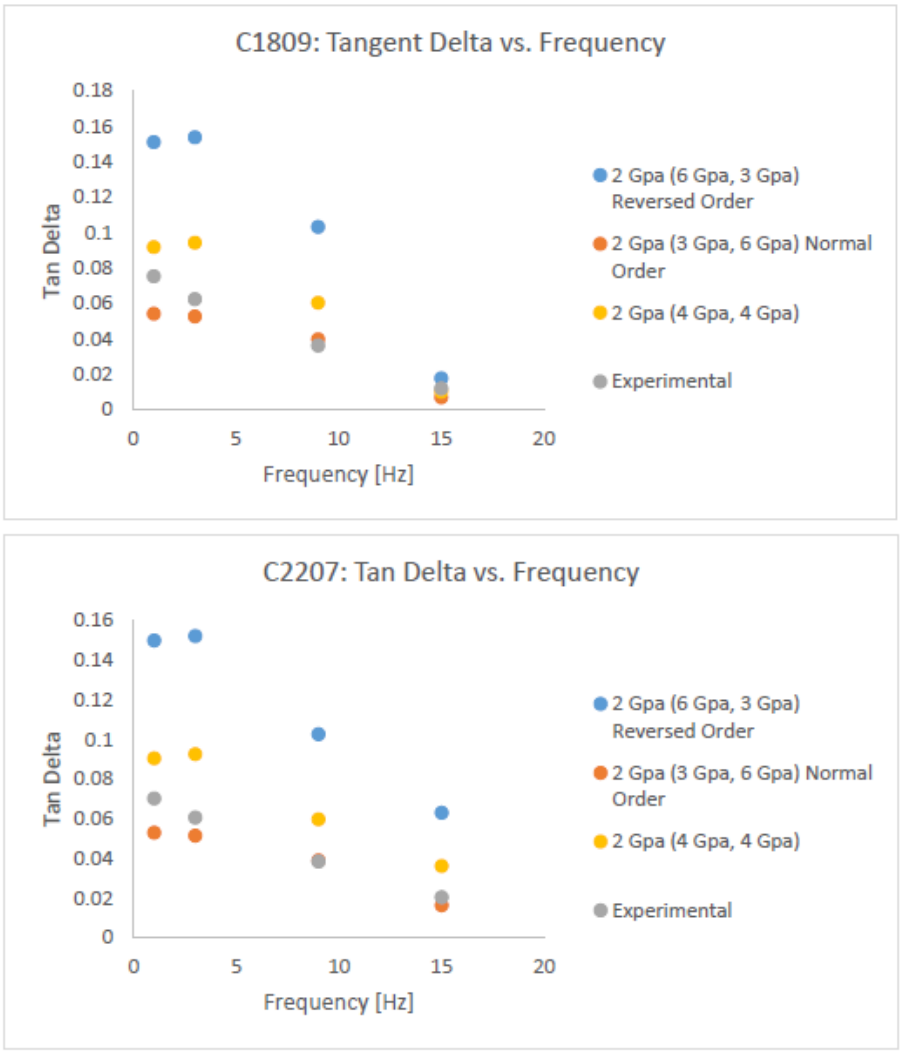

Figure 2.19: Thompson Effective Modulus Results: Thompson found that while using the best fitting and most experimentally sensible mineral elastic modulus of 36 GPa, an E_{eq} equal to 2 GPa with the ordering of 3 GPa and 6 GPa produced the best results for C1809 and C2207 [74]

This leaves just one final parameter in tropocollagen to inspect for the chosen rheological model: the dashpot value η_1, perhaps most difficult to find a literature value for. Mendoza selected his dashpot last, holding other parameters constant and executing the graphical approach shown in figure 2.20 [54]. Knowing the mean tangent delta for control animals, he adjusted his model's dashpot until his tangent delta matched that of the control's DMA performance at 15 Hz, ultimately selecting a dashpot of 1.25

GPa-s [54]. 1.25 GPa-s turned out to be an effective choice for subsequent research, and is the value of choice for the IMF scheme for the sake of simplicity.

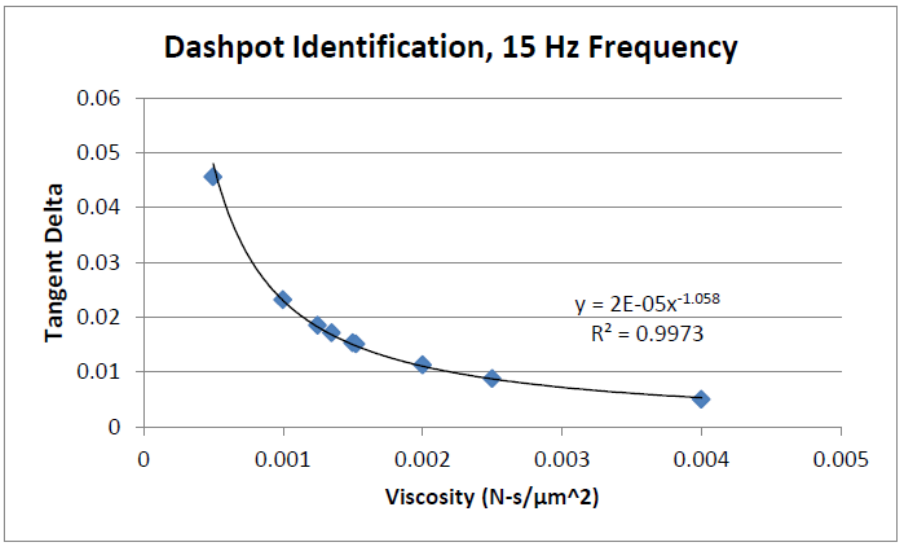

Figure 2.20: Mendoza Dashpot Determination: Graphical approach used by Mendoza to select a η_1 value of 1.25 GPa-s. [54]

Subsequent research found however that a careful consideration of the dashpot value could yield significant results and that altering it changed tangent delta values significantly. Thompson elected to study 0.0125, 0.125, and 1.25 GPa-s as values going off of what Mendoza had chosen to discover what changing the order of magnitude of the dashpot might reveal, and additionally used 0.3125 GPa-s for its respectable performance in previous studies [74]. Thompson found that in the upper frequencies 0.3125 GPa-s seemed to produce the greatest fit, but at 1 Hz several other values produced better fit: when 36 GPa was used for mineral elastic modulus, 1.25 GPa-s worked best, for a mineral elastic constant of 100 GPa, 1.00 GPa-s won out [74]. Because different dashpot values best suited different frequencies, Thompson suggested the exploration of additional dashpot values and in general a study of a dashpot viscosity that is driven by the loading condition (i.e. frequency) [74].

In the Variable Dashpot scheme, samples C1809 and C2207 were tested against additional frequencies in attempt to identify spikes in fit, and additional dashpots to select even better fits. Models were run at loading conditions of 2, 5, 7, and 12 Hz in addition to the original 1, 3, 9 and 15 Hz used in preceding studies. Dashpot values spanned the gaps of values that performed well in previous reports. Dashpot values 0.0125, 0.125, 0.3125, 0.45, 0.5875, 0.725, 0.8625, and 1.25 GPa-s were studied.

After identifying a best fit dashpot for each frequency, plots were generated to establish a relationship between loading frequency and dashpot value selection. A polynomial best fit line was extracted (figures 2.21 and 2.22), and used as a conversion formula within a modified version of Richter's UMAT. When editing the input file, instead of entering a dashpot value in the usual place, the user enters the desired loading condition frequency so that the modified UMAT takes that as an independent variable and calculates a best fit η_1 value for that particular frequency.

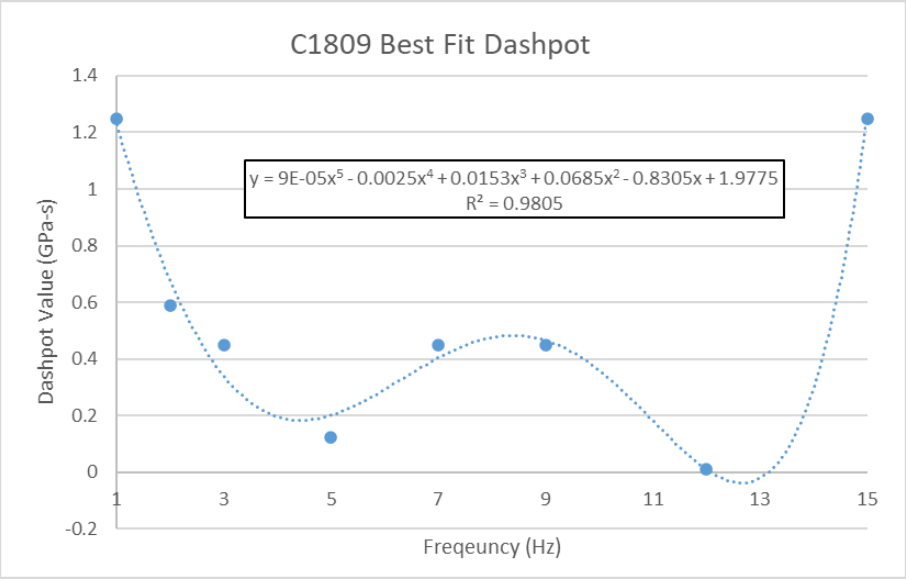

Figure 2.21: C1809 OVX Best Fit Dashpot v. Frequency: Plotting the dashpot values that performed the best at each frequency allows the fitting of a polynomial equation.

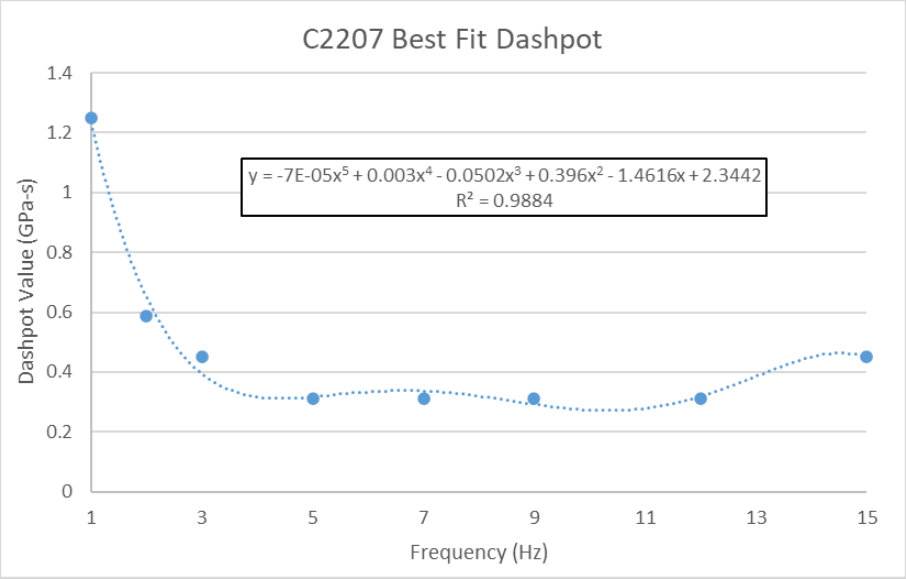

Figure 2.22: C2207 Control Best Fit Dashpot v. Frequency: Plotting the dashpot values that performed the best at each frequency allows the fitting of a polynomial equation.

If the polynomial had a perfect correlation coefficient, the resulting variable dashpot performance would look like figure 2.23, where every best dashpot for both models at each frequency are plotted alongside experimental data. Table 2.4 show the exact values outputted by the modified Richter UMAT used (section E.2.1) with the exception of the negative values: the absolute value of all values were taken to avoid nonsensical negative dashpot values. All outputs were subsequently divided by 1000 to provide Abaqus with the proper units of MPa-s. It should be noted that while these outputs were close, they were not perfectly aligned with the best performing dashpot values exhibited in figures 2.21 and 2.22.

Table 2.4: Modified UMAT Dashpot Values: Values outputted by modified Richter UMAT used by Abaqus. All dashpots are in units of GPa-s.

Frequency (Hz)	1	2	3	5	7	9	12	15
C2207 UMAT Dashpot	1.23137	0.64921	0.39383	0.31717	0.33631	0.29232	0.31813	0.44914
C1809 UMAT Dashpot	1.19864	0.61681	0.24936	0.05118	0.19391	0.19596	-0.3491	0.80019

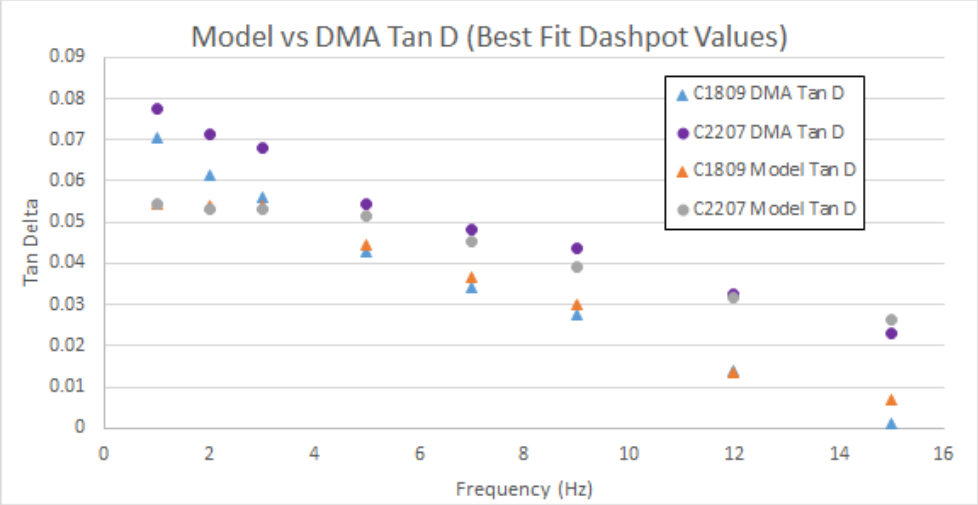

Figure 2.23: Best Fit Dashpot Compared to Experimental: Performance of a hypothetical perfect variable dashpot as compared to experimental DMA data for Sham and OVX

Two models were built (C1809 and C2207) and eight dashpot values were tested over eight frequencies with post-processing scripts run over node displacements ten times each, generating a total of 1280 tangent delta values to be averaged.

2.4.4 Boundary Conditions and Loading

Though the geometry of the IMF and Variable Dashpot schemes differ, boundary conditions and loading were applied exactly the same way between the two.

Boundary conditions were applied to mimic biology, the Siegmund experiments, and to simplify the analysis of the model [69]. The general approach was to have a long structure that experienced horizontal but not vertical displacement when loaded on an unrestricted right edge (referred to as model terminus). Regardless of exact model geometry or D-Spacing, the entire left edge was selected as one surface and a boundary condition named XSYM was applied, which restricted displacement in X or rotation about Y or Z. Restricting the left edge this way as shown in figure 2.24 allowed cells

of the model, when loaded from the right side, to experience tension the same way they would in vivo.

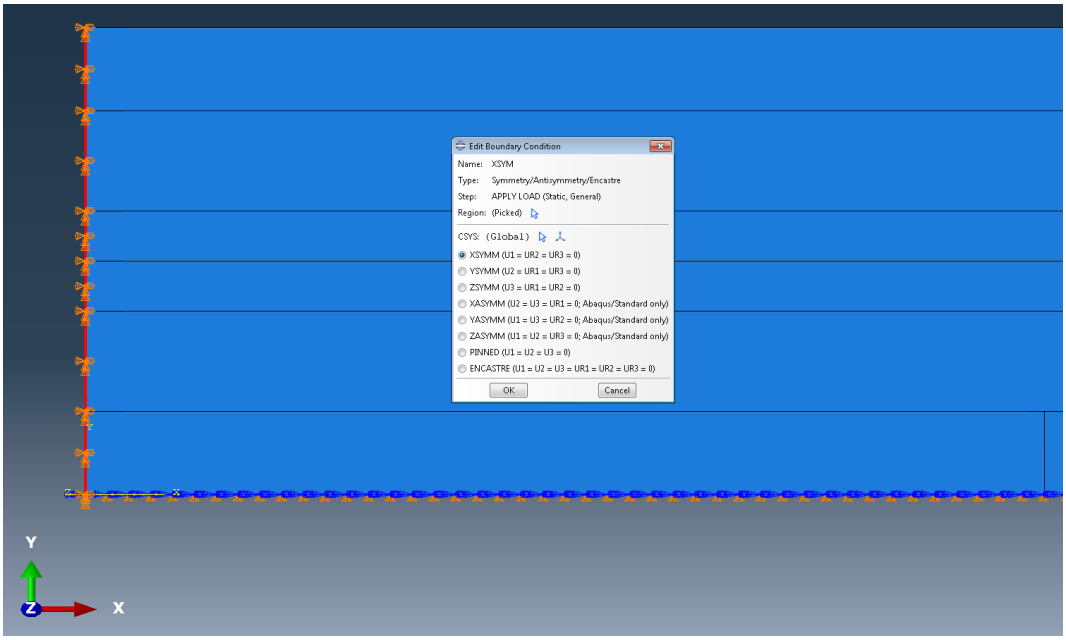

Figure 2.24: XSYM Boundary Condition: The highlighted triangles on the model left edge represent a restriction in horizontal displacement

Similarly, a boundary condition named YSYM was applied to the entire bottom surface of the model to restrict displacement in Y or rotation about X or Z as shown in figure 2.25. This allowed for meaningful measurements of terminal node displacement and made sense in the biological context of rows of tropocollagen and mineral surrounded by other rows of tropocollagen and mineral.

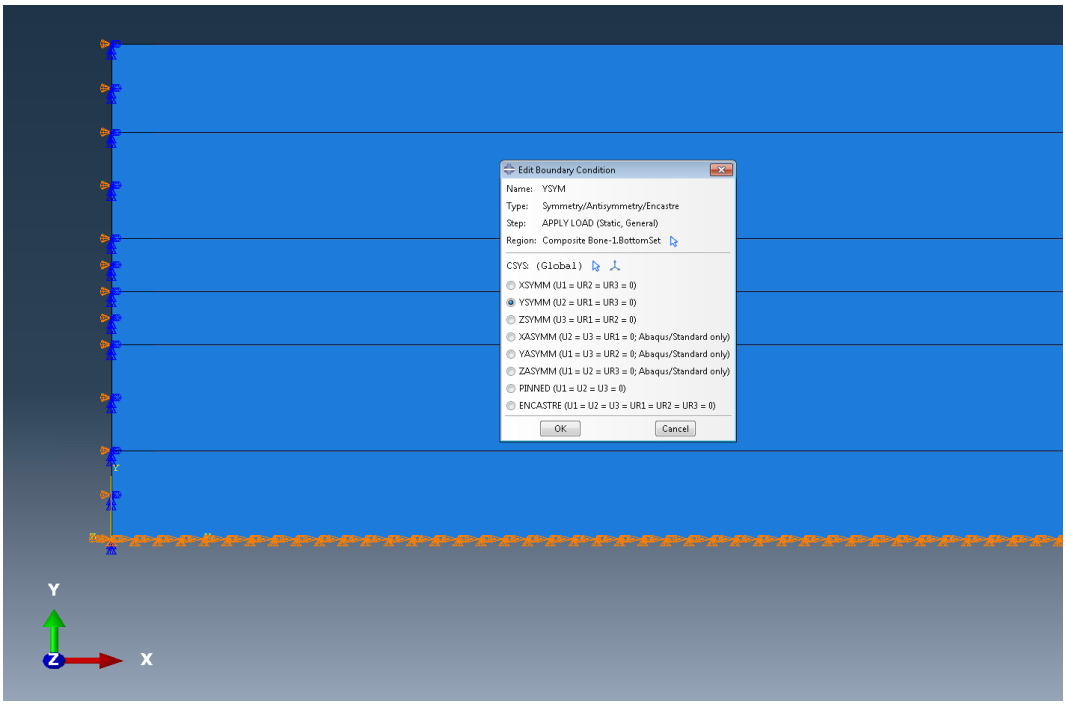

Figure 2.25: YSYM Boundary Condition: The highlighted triangles on the model left edge represent a restriction in vertical displacement, but does allow horizontal displacement in reaction to loading.

The unrestricted right edge of the models had a sinusoidal load applied to it to induce uniaxial tension reminiscent of experimental DMA. The load consisted of 20 cycles with 20 increments each, making for a total of 400 steps. These 20 cycles were applied at frequencies at 1, 2, 3, 5, 7, 9, 12 and 15 Hz, though the real DMA data spanned 1 through 20 Hz in increments of 0.2 Hz [46]. Table 2.5 shows how long each step took at each test frequency configuration run in these experiments and figure 2.26 shows the typical sinusoidal displacement pattern traced out by the terminal nodes.

Table 2.5: Step time and angular frequency for each test frequency

Test Frequency (Hz)	Step Time (s)	Angular Frequency (rad/s)
1	0.05	6.28319
2	0.025	12.56638
3	0.016666667	18.84957
5	0.01	31.41595
7	0.007142857	43.98233
9	0.005555556	56.54871
12	0.004166667	75.39828
15	0.003333333	94.24785

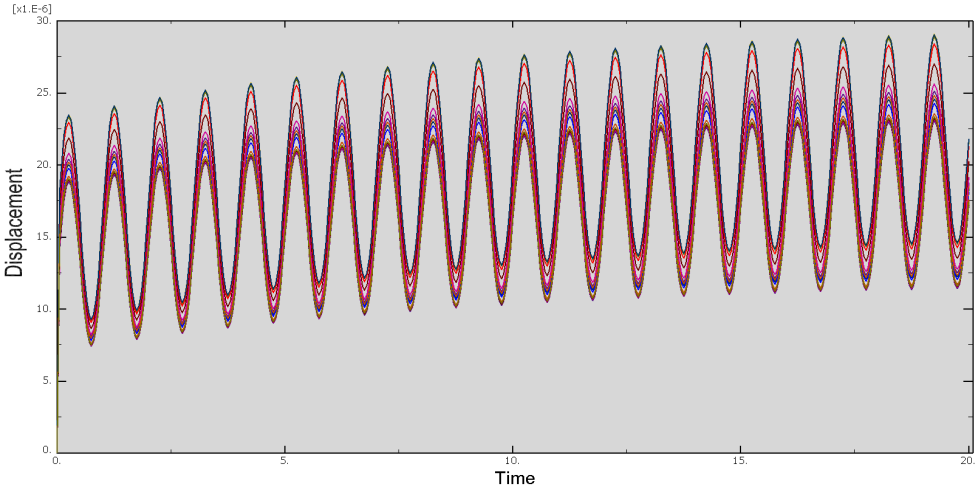

Figure 2.26: 20 Cycles at 1 Hz: Plot of all ending nodes displacement v. time for the IMF scheme model. Note that the plot ordinate is in units of microns.

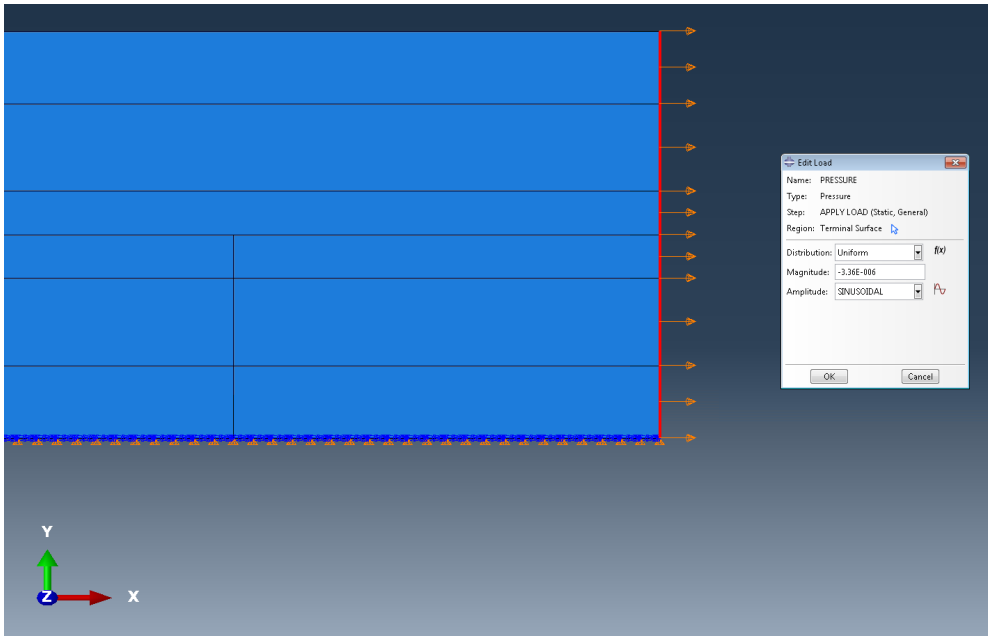

Figure 2.27: Sinusoidal Load: A 20 cycle sinusoidal load was applied to the entire right edge of the model to induce uniaxial tension.

The load amplitude was 3.36 MPa regardless of testing configuration or scheme, which was meant to mimic the approximate stress of bone samples in the real DMA testing. This was applied to the terminal end of the model as seen in figure 2.27. Three point bending dynamic mechanical testing was performed on each specimen including the two used in this thesis, which were 15mm x 1.75mm x 1.75mm using the arrangement shown in the figure 2.28.

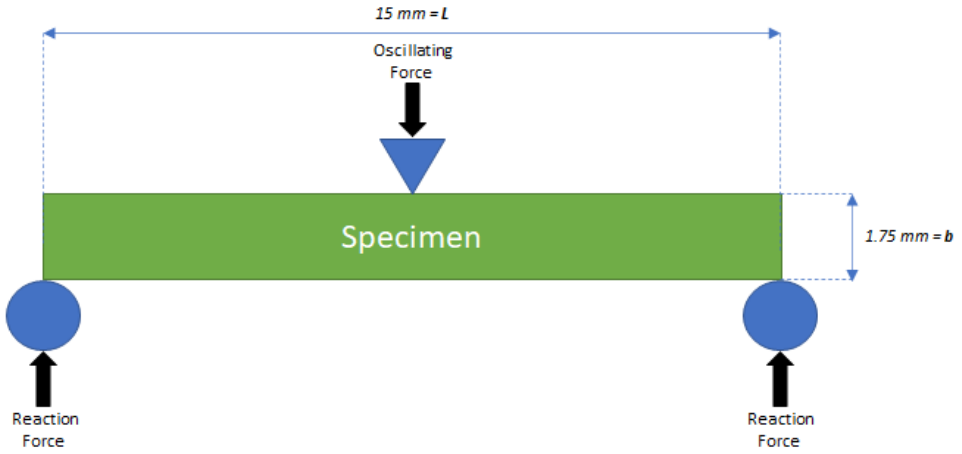

Figure 2.28: DMA Setup: Each specimen underwent 3 point bending DMA with b and h = 1.75mm and L = 15mm.

550 mN static and 500 mN dynamic loads were applied. The amplitude of the sinusoidal wave value chosen in Abaqus was derived from considering the equation for the maximum resultant stress from a bending moment:

$$\sigma_{max} = \frac{M_{max} y}{I} \tag{2.15}$$

Where M_{max} is the bending moment halfway along the length of the specimen, y is the vertical distance from the neutral axis (middle of the specimen in this case) to the specimen edge, and I is the moment of inertia. For an object of this shape, the moment of inertia equals

$$I = \frac{bh^3}{12} \tag{2.16}$$

and filling in dimensions gives

$$I = \frac{(1.75 \cdot 10^{-3}m)(1.75 \cdot 10^{-3}m)^3}{12} = 7.82 \cdot 10^{-13}m^4 \tag{2.17}$$

To determine the maximum moment, the static and dynamic forces were summed. The static force was straightforward. The dynamic loading condition was written as a function of length from halfway along the length of the specimen and the time at which the dynamic force was at a maximum (recall that it's oscillating). Measuring the dynamic load at a force maximum halfway along the length of the specimen:

$$F_{dynamic} = F(0, t_{max}) = \frac{500 \cdot 10^{-3}N}{4} \tag{2.18}$$

and combining this with the static force gave

$$\Sigma F = F_{static} + F_{dynamic} = \frac{550 \cdot 10^{-3}N}{2} + \frac{500 \cdot 10^{-3}N}{4} = 400 \cdot 10^{-3}N \tag{2.19}$$

and so the maximum bending moment was

$$M_{max} = \Sigma F \cdot d = 400 \cdot 10^{-3}N \cdot (7.5 \cdot 10^{-3}m) = 3 \cdot 10^{-3}Nm \tag{2.20}$$

and so the maximum stress experienced anywhere in the specimen at peak loading time was

$$\sigma_{max} = \frac{M_{max}y}{I} = \frac{(3 \cdot 10^{-3}Nm)(\frac{1.75 \cdot 10^{-3}m}{2})}{7.82 \cdot 10^{-13}m^4} = \mathbf{3.36 \cdot 10^6 Pa} \tag{2.21}$$

2.4.5 Mesh Development

Mesh choice in Finite Element Analysis is of utmost importance because it can not only affect accuracy, but also computation time. The latter is particularly salient in the Variable Dashpot Scheme, where models featured about 21,900 elements and required run times of about 2.5 hours for each job. At a certain element and node count in a FEA model, accuracy comes at starkly diminished returns in spite of increased computation time. Because this exact geometric format of the Complex Model has undergone convergence studies through various theses, a seed size of 0.5 nm was chosen which correlates to about 1.3 million degrees of freedom [35][16][74]. Smaller seed sizes caused Abaqus to crash or did not confer significant increases in accuracy.

The Complex Model's simple two-dimensional shape made for a straightforward choice of element type. Plane-strain element types were chosen because the model is based off of that used by Siegmund, and quadratic quadrilateral elements were chosen because they offered increased accuracy and because of their appropriateness for uniaxial tension (no bending moments) [69]. The exact configuration settings can be seen in figure 2.29.

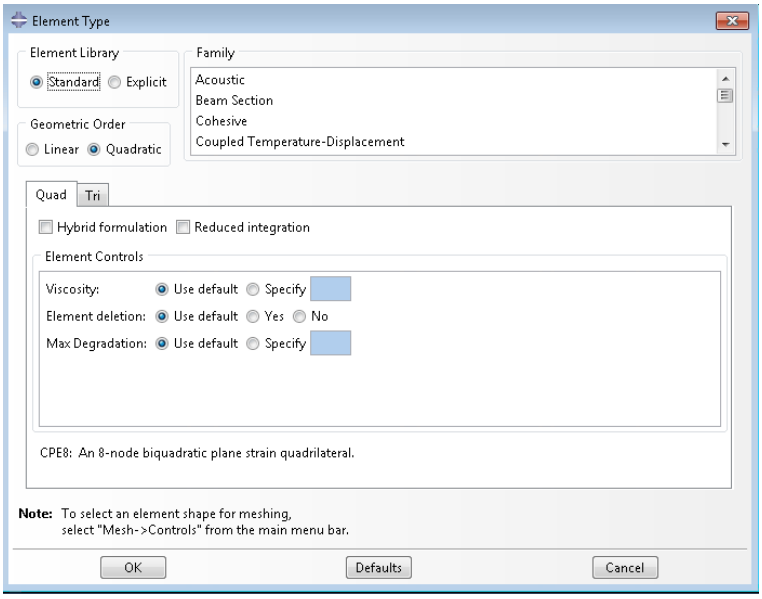

Figure 2.29: Element Selection: Quadratic Quadrilateral Plane Strain elements (Abaqus code CPE8) were chosen for every model considered

No fatal errors related to mesh or element types occurred in any of the models.

2.4.6 Model Validation

Both the IMF and Variable Dashpot schemes use models whose specific geometry have been validated both in practice and by hand calculation for several iterations. Confirmation of the viscoelastic parameters of collagen were done via hand calculation by way of creep and stress relaxation equations. These equations and their derivations can be viewed in Appendix D.

2.5 Post Processing

A completed model run results in the creation of several files, most notably the output database (.odb) file. This file contains all the data that was previously requested during job creation for every step of the computation. It can be opened within the

Abaqus GUI to view such things as replay animations, graphical representations of stress and strain, or plots of things such as node displacement like those seen in figure 2.5. In the service of more streamlined repeatable experiments, odbs are also fully accessible via python scripts by way of the odbAccess package so long as the python script is run within an Intel 64 Visual Studio command prompt window. A typical experimental run is done via batch file as seen in the following code.

```
call abaqus job=%jobname% interactive user=RichterUMATv2.f cpus=8
md %jobname%
cd %jobname%
call ..\abq_cleanup
call abaqus python postprocessing_TGM.py %jobname%.odb
erase postprocessing_TGM.py
cd ..\
```

After each job is complete the batch file moves the .odb into a specific directory so that a python node retrieval file is run against it. For the IMF scheme this file can be viewed in Appendix E.5.2 and iterates through each part in the model to grab horizontal displacements from nodes in a pre-specified set named TERMINUS. The Variable Dashpot scheme used a separate file (Appendix E.5.1) which required an input of nine equally spaced nodes on the terminus of the model. These nodes were different between sham and OVX models because each had a different geometry and therefore different node and element placements. Either python script created one text file per node and wrote a total of 400 displacement values, one per step, for later ingestion into Matlab post processing scripts.

The data was then read in by a Matlab script (Appendix E.6) where the Tangent Delta was computed. Each file was named, imported, and the contents were arranged

into arrays. The average nodal displacement for each time step was then used to fill a new 400 x 1 array, which was important because if some part of the model terminus was moving significantly more or less than the rest of the terminus, this would skew the data. This array was then divided by the total length of the model to compute the average strain, which in the IMF scheme was exactly 67 nm (input as 0.067 microns) and in the Variable Dashpot scheme something in the ballpark of 6.7 microns, depending on the geometric output of the Model Generation Script.

The proper frequency parameter was un-commented so that a 200 x 1 time array was created with the correct start, stride and termination for each step based on the chosen frequency. The Matlab optimization function fminsearch was used to minimize the function CurveFit to fit a sinusoidal curve to the plotted displacements and then produce an Error Sum of Squares between the fitted curve and the actual data.

Recall that

$$\sigma = \sigma_o sin(2\pi f) \tag{2.22}$$

and

$$\epsilon = \epsilon_o sin(2\pi f - \delta) \tag{2.23}$$

One of the outputs returned by fminsearch was the phase shift from that normal sine function, and this phase shift is tangent delta. The strain curve was computed by inputting the phase shift and the amplitude of the sine function returned by fminsearch into equation 2.23. A visual output of this can be seen in figure 2.30. The function rsquare was used to compute the correlation coefficient and root mean square error between the newly generated strain curve and the displacements from the initially imported data. The RMSE, R^2 and tangent delta values were recorded. The script was replayed 10 times and the three aforementioned parameters were averaged for plotting and comparison purposes.

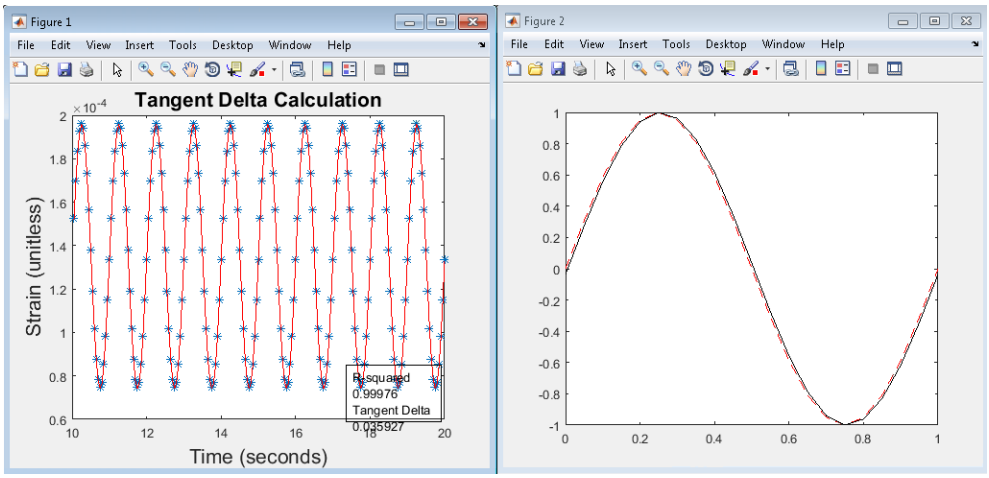

Figure 2.30: Post Processing Plots: The left plot shows the red fitted curve applied over the data (blue dots) providing a correlation coefficient and a tangent delta value. The right figure shows the stress and strain curves being offset by a value equal to tangent delta.

2.5.1 Statistical Analysis

To determine the performance of each model configuration both for comparison between experimental tan delta values and those generated by previous studies, simple linear regression was employed to provide visual distinction as well as R^2 and RMSE values.

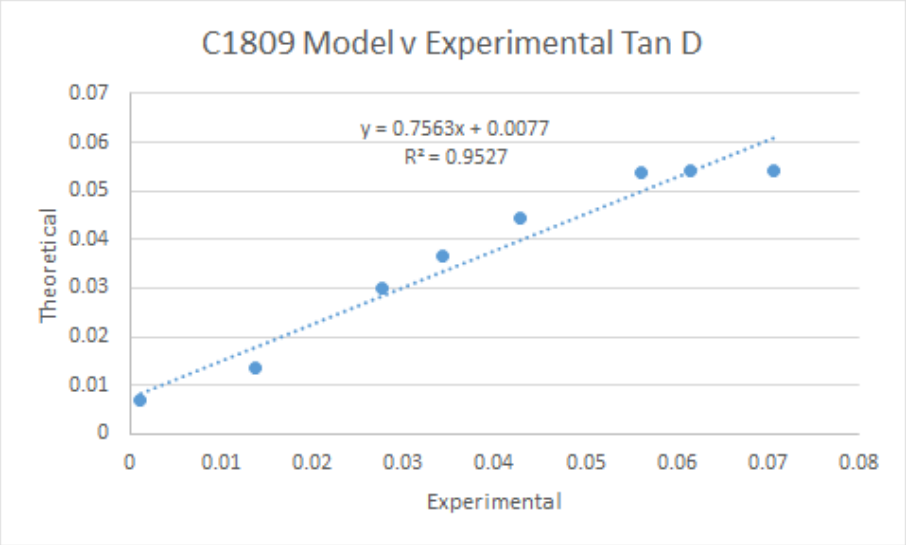

Figure 2.31: Linear Regression Example Plot: An example of a basic linear regression plot showing a fitted linear equation and an R^2 value.

R^2 or the correlation coefficient is widely used in regression and could roughly be defined as a metric of the "tightness" of a relationship in data and is displayed on equation of fit in plots generated in excel such as in figure 2.31. It is more specifically the percentage of the variability between data that is explained by the fitted line [34]. This is not to be confused with something like the F value produced by two way ANOVA, which established whether or not, beyond a reasonable doubt, there is a relationship between data, not to what extent. The correlation coefficient has an intuitive scaling from 0 to 1, with zero being no relationship whatsoever to 1 being a perfect fit between the data and the trend line, the latter being suspicious in its own right.

The RMSE on the other hand is the square root of the variance of the residuals, or alternatively worded: the standard deviation of the unexplained variance [34]. Unlike R^2, it is not a percentage or fraction but shares the same units as the response variable (in this case tangent delta). Lower RMSE means better fit, and RMSE is especially handy in regression models with many predictors because while the correlation coef-

ficient increases with each added predictor in a regression equation, the RMSE is an absolute value and will not be affected by this. For the regression performed in this research, both values are worth noting because this regression equation is simple and has only one predictor: that the experimental value will predict the theoretical value.

Chapter 3

RESULTS

Results are split into two major sections: the first being for the IMF scheme and the second for the Variable Dashpot scheme, the latter of which contains a subsection dedicated to results captured using newly modified versions of Richter's UMATs [65].

3.1 IMF Scheme Results

All IMF models were geometrically identical to Mendoza's "Normal D-Spacing" 67 nm length model, and use the parameters specified in Table 3.1 [54]. All IMF scheme models were run at 1, 3, 9 and 15 Hz for the basis of comparison with previous model approaches, and were all compared against experimental control cranial specimen C2207. Each model is presented with a Tangent Delta v. Frequency plot alongside corresponding experimental values as well as a simple Linear Regression plot against aforementioned experimental values.

Table 3.1: Parameter values used for all IMF scheme model runs

Collagen Parameter Values	
Elastic Modulus 1	3 GPa
Elastic Modulus 2	6 GPa
Dashpot Viscosity	1.25 GPa-s
Poisson's Ratio 1	0.2
Poisson's Ratio 2	0.2
Poisson's Ratio 3	0.2
Mineral Parameter Values	
Elastic Modulus	100 GPa
Poisson's Ratio	0.28

The first model to be run was named the Tie Constraints Model, which acted as a baseline to see how a model performed that was made of the four separate parts specified in figure 2.11. Each of these parts were tied together at every node using tie constraints in Abaqus, and this was expected to behave closely to previous single-part models.

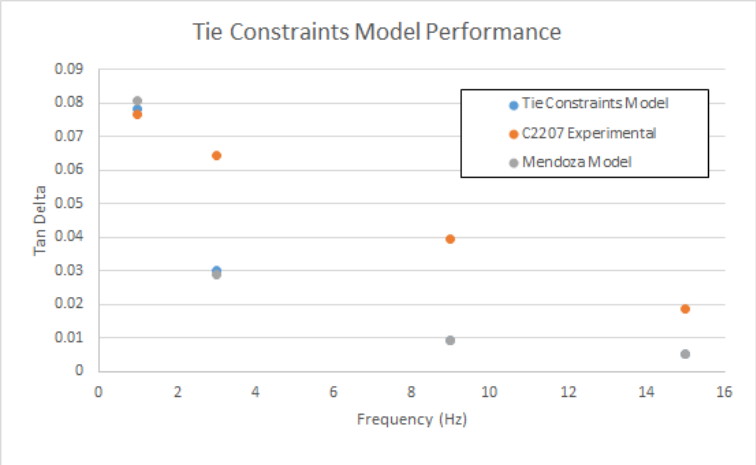

Figure 3.1: Tie Constraints Model Performance: Fully tie-constrained model tangent delta performance

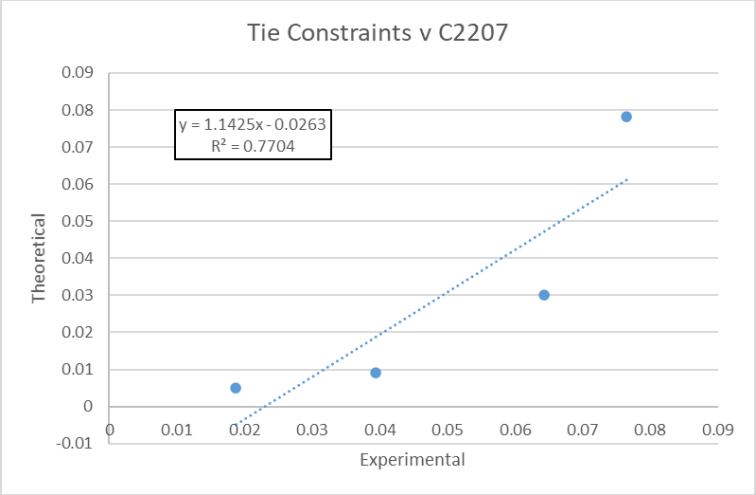

Figure 3.2: Tie Constraints Model vs C2207 Linear Regression

Figure 3.1 reveals that while the tie constraints model yields very similar tan delta values at 1 Hz, the numbers quickly diverge, almost converging again at 15 Hz. Overall

performance at mid-frequency is poor, and figure 3.2 shows an unimpressive R^2 value of 0.7704.

The next phase of experimentation involved generating a single enzymatic crosslink at the vertex of the Collagen Small, Collagen Full and Mineral parts (fig 2.11) in addition to a set number of non-enzymatic crosslinks randomly arranged as detailed in section 2.4.1. These model iterations included N=20, 25, 30 and 35 non-enzymatic crosslinks. Each of these four configurations included three distinct models with different random node assignments to ensure that a skewed random assignment would solely represent the configuration, and the tangent delta values for each model was averaged and plotted against C2207.

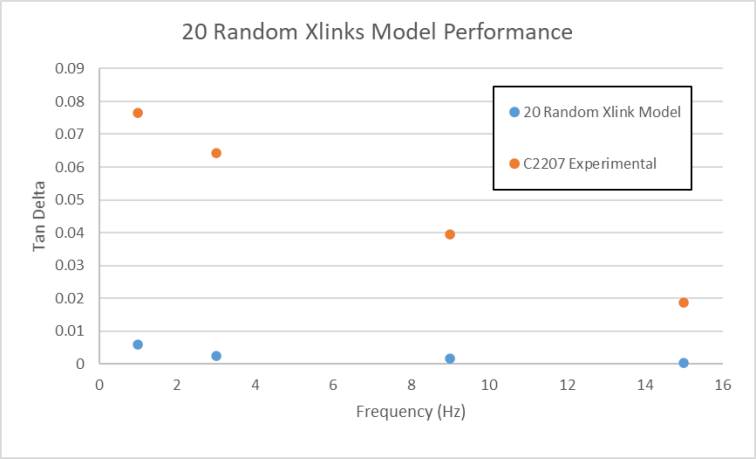

Figure 3.3: N=20 Random Crosslinks Model Performance

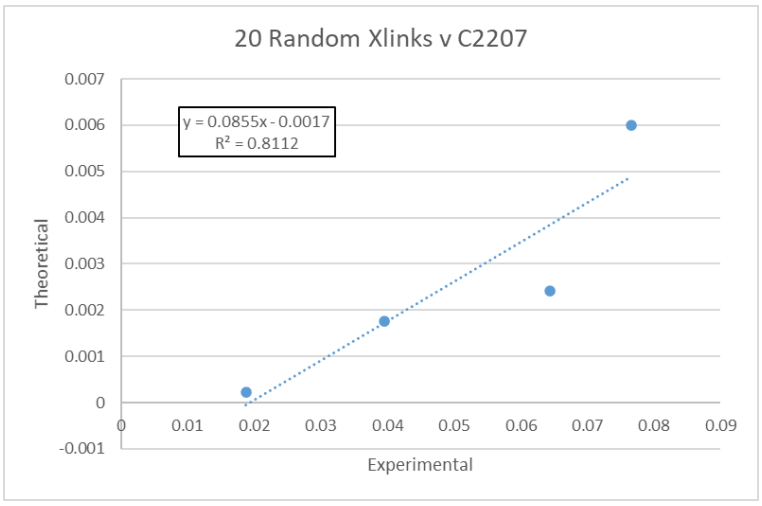

Figure 3.4: N=20 Random Crosslinks Model vs C2207 Linear Regression

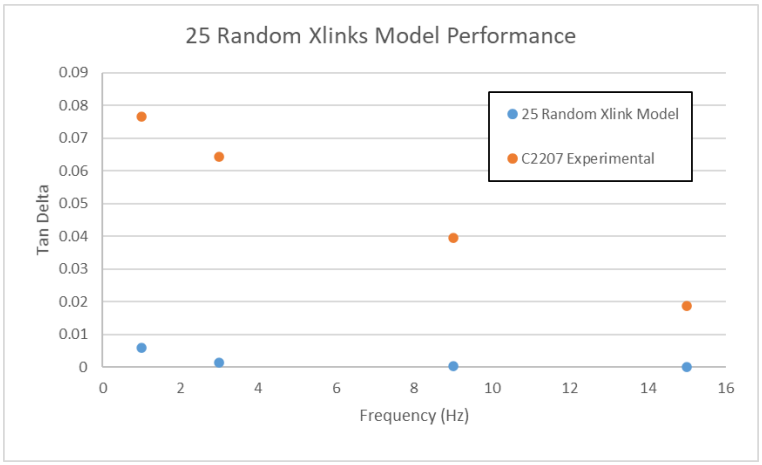

Figure 3.5: N=25 Random Crosslinks Model Performance

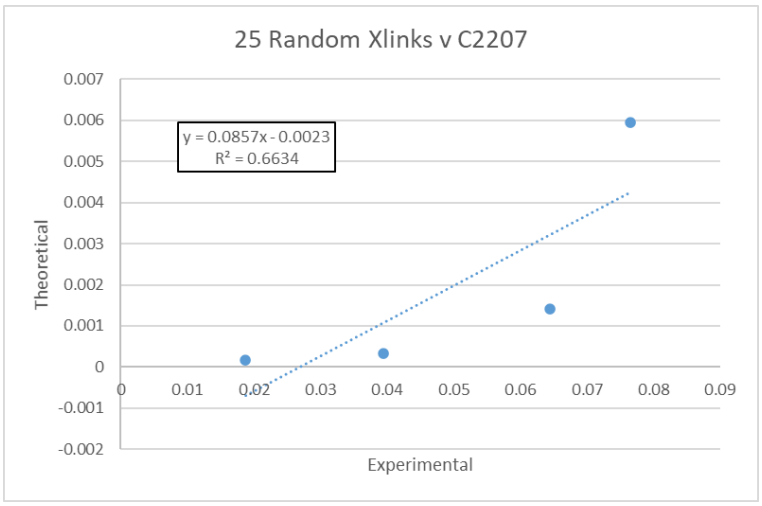

Figure 3.6: N=25 Random Crosslinks Model vs C2207 Linear Regression

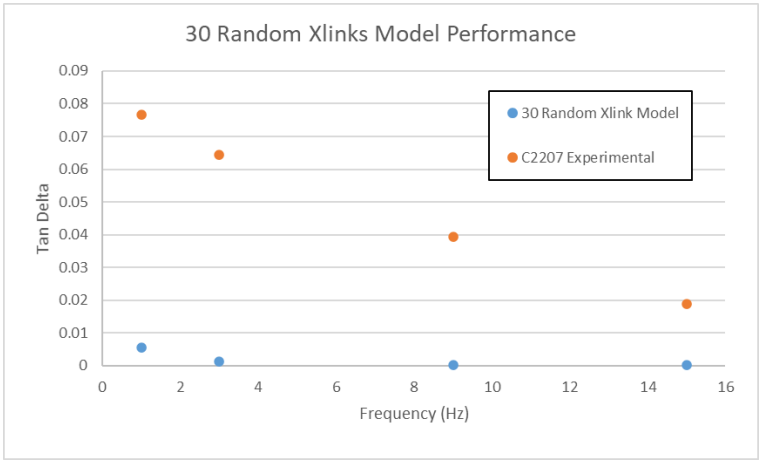

Figure 3.7: N=30 Random Crosslinks Model Performance

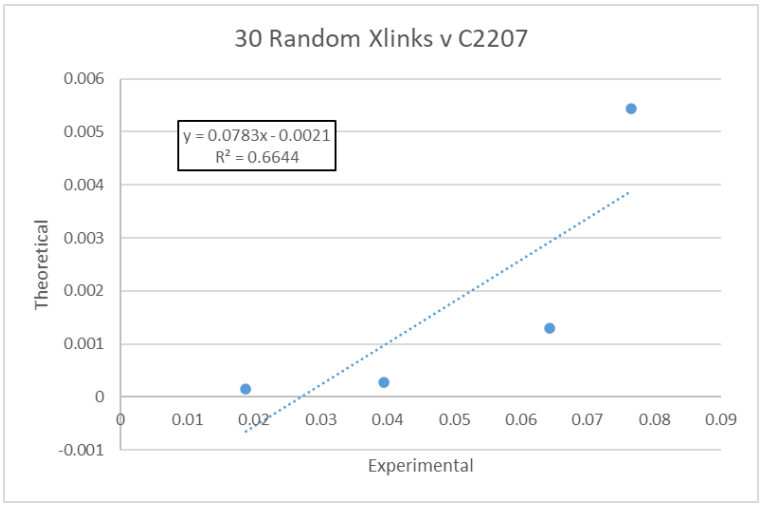

Figure 3.8: N=30 Random Crosslinks Model vs C2207 Linear Regression

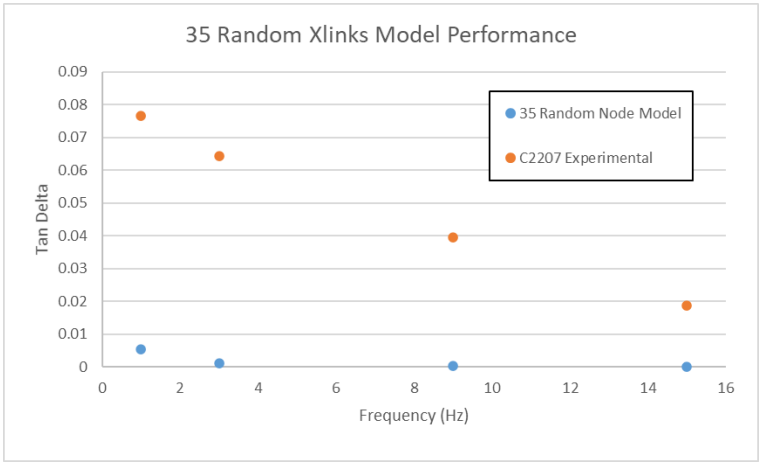

Figure 3.9: N=35 Random Crosslinks Model Performance

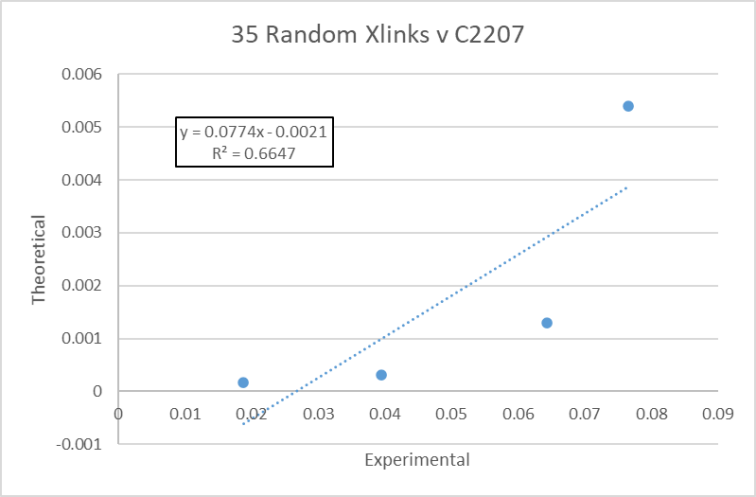

Figure 3.10: N=35 Random Crosslinks Model vs C2207 Linear Regression

Generally the randomized crosslink models performed poorly when compared to Mendoza's approach of using a single part, and when compared to the fully tie constrained model as shown in figure 3.11. Surprisingly, the best performing randomized node model according to correlation coefficient was the N=20 model (fig 3.3 and 3.4), with N=25 (fig 3.5 and 3.6) N=30 (fig 3.7 and 3.7) and N=35 (fig 3.9 and 3.10) all performing much worse, sporting correlation coefficients less than 0.67. It should again be noted that the middle frequencies on average performed noticeably worse than 1 and 15 Hz.

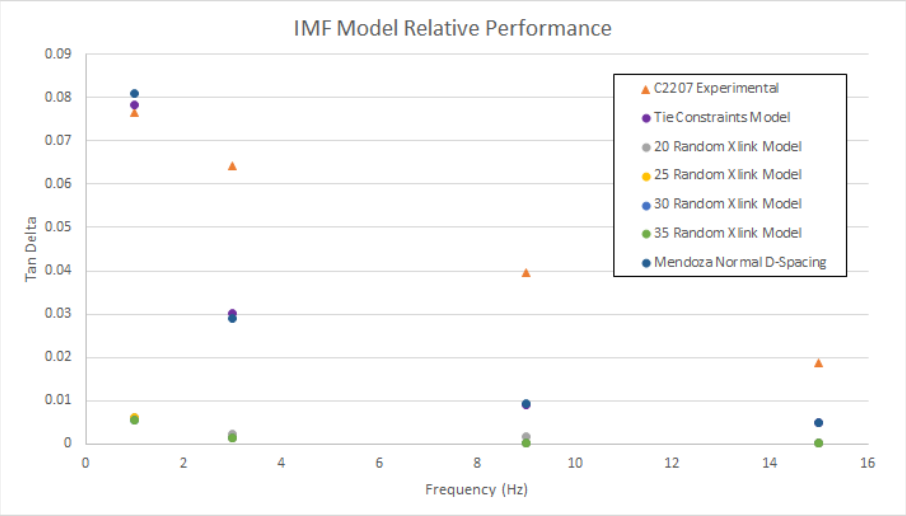

Figure 3.11: Relative Performance of IMF Models

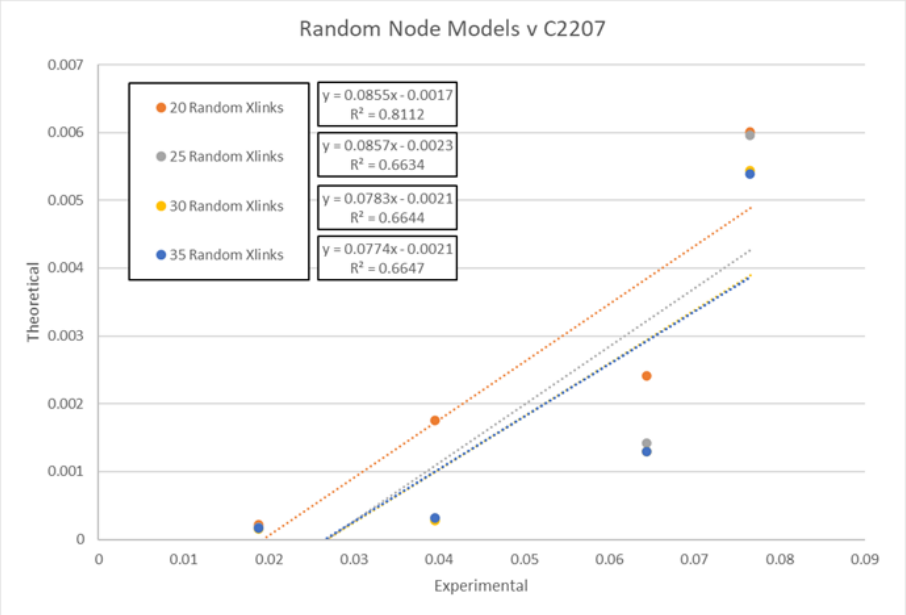

Figure 3.12: All Random Node Models vs. C2207

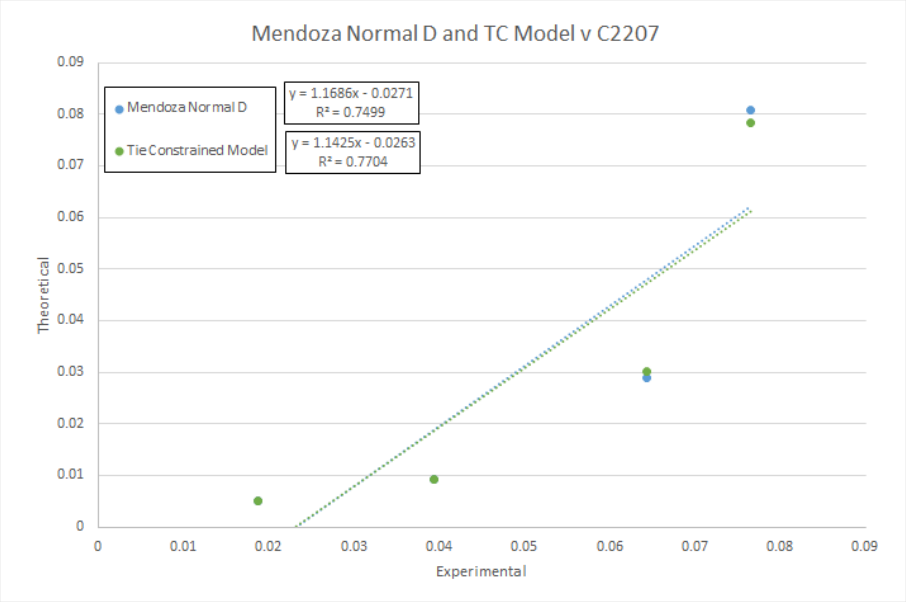

Figure 3.13: Mendoza Normal D-Spacing and Tie Constraints vs. C2207 [54]

Table 3.2: Mean Tangent Delta values of each IMF model and their RMSE vs. experimental and Mendoza Normal-D Spacing model. Includes T.C. (Fully Tie Constrained) and RX (Random Crosslink) models [54].

	IMF Model Mean Tan Deltas						
Frequency	**T.C.**	**20 RX**	**25 RX**	**30 RX**	**35 RX**	**Mendoza**	**C2207**
1 Hz	0.078221	0.006004	0.005960	0.005431	0.005393	0.080891	0.076557
3 Hz	0.030023	0.002406	0.001419	0.001296	0.001296	0.028964	0.064371
9 Hz	0.009141	0.001757	0.000322	0.000284	0.000321	0.009268	0.039474
15 Hz	0.004967	0.000222	0.000179	0.000157	0.000171	0.005051	0.018764
RMSE	0.023943	0.051438	0.052022	0.052248	0.052253	0.024356	
R^2	0.770400	0.811200	0.663400	0.664400	0.664700	0.749900	

Each IMF model variant (e.g. Tie Constrained, N=20 Crosslinks etc.) were run with three different random arrangements and their average tangent delta values are shown in Table 3.2. Note that while the correlation coefficients were somewhat similar between models, the RMSE value for the fully Tie Constrained model is much lower than other IMF models and indicates a better fit. Figure 3.13 shows a linear regression of the fully tie constrained model against C2207, as well as Mendoza's Normal D-Spacing model against C2207 [54].

3.2 Variable Dashpot Scheme Results

Variable Dashpot Models were generated in the same fashion as the most recent study conducted by Luke Thompson: with a model generation script inputted with mean and standard deviation values for D-Spacing that correspond to the specimen of interest [74]. In the case of this study, the specimen of interest were cranial control specimen C2207 and cranial OVX specimen C1809. The cranial portion of the ewe radius was suspected to be of particularly good fit for the Complex Model because it is predominately in a state of tension, and the model's boundary conditions and applied load are meant to simulate uniaxial tension. Models built against these geometries were tested with dashpot values of 0.0125, 0.125, 0.3125, 0.450, 0.5875, 0.725, 0.8625, and 1.25 GPa-s across 1, 2, 3, 5, 7, 9, 12 and 15 Hz. All parameters were held constant with the exception of dashpot values as shown in Table 3.3.

Table 3.3: Parameter values used for all VD scheme model runs

Collagen Parameter Values	
Elastic Modulus 1	3 GPa
Elastic Modulus 2	6 GPa
Dashpot Viscosity	Variable
Poisson's Ratio 1	0.2
Poisson's Ratio 2	0.2
Poisson's Ratio 3	0.2
Mineral Parameter Values	
Elastic Modulus	36 GPa
Poisson's Ratio	0.28

3.2.1 Control Cranial Specimen C2207 Results

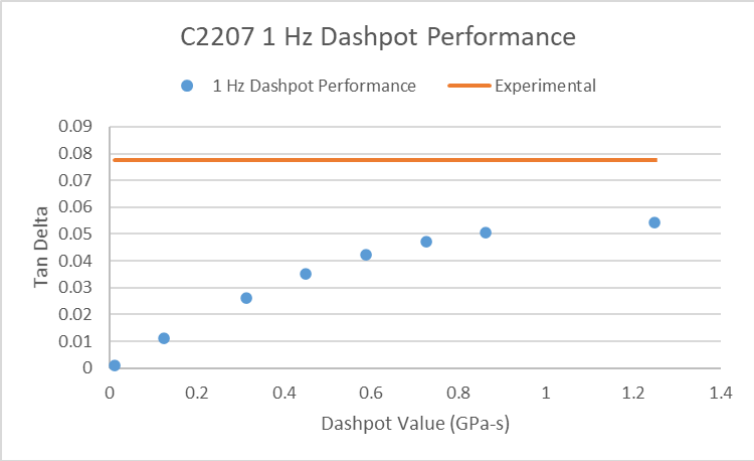

Figure 3.14: C2207 1 Hz Dashpot Performance

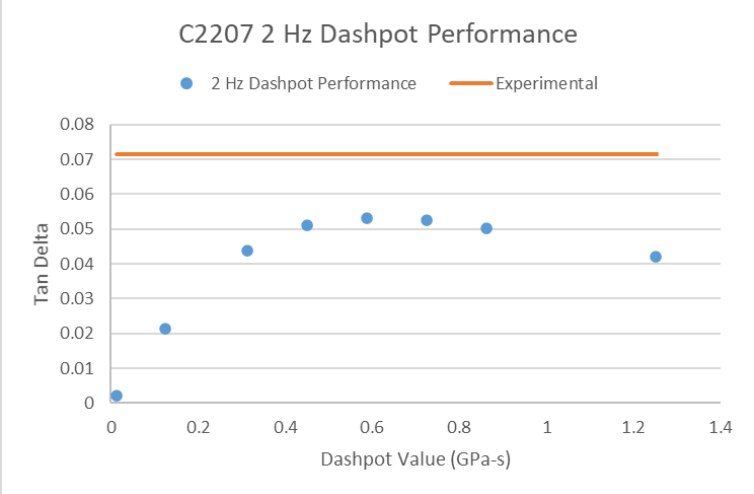

Figure 3.15: C2207 2 Hz Dashpot Performance

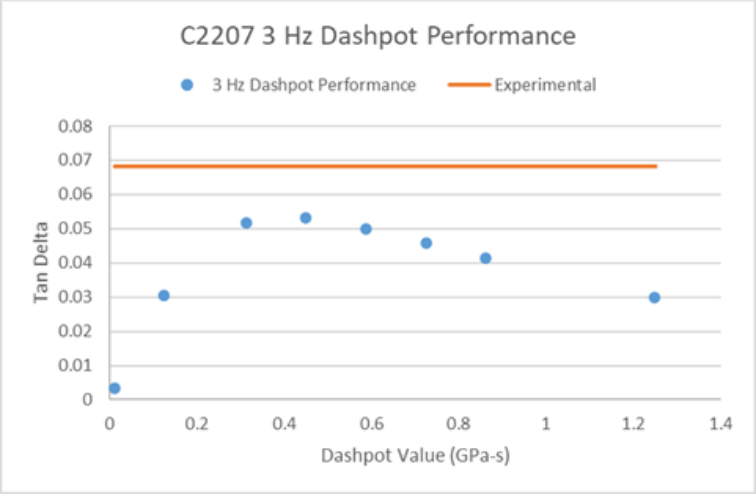

Figure 3.16: C2207 3 Hz Dashpot Performance

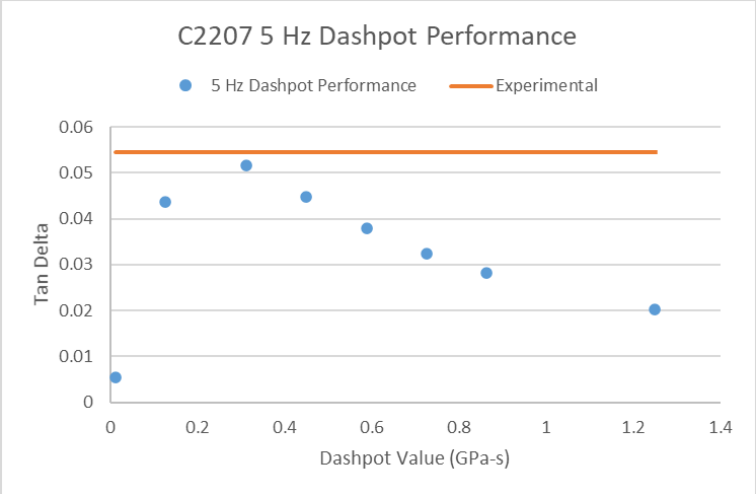

Figure 3.17: C2207 5 Hz Dashpot Performance

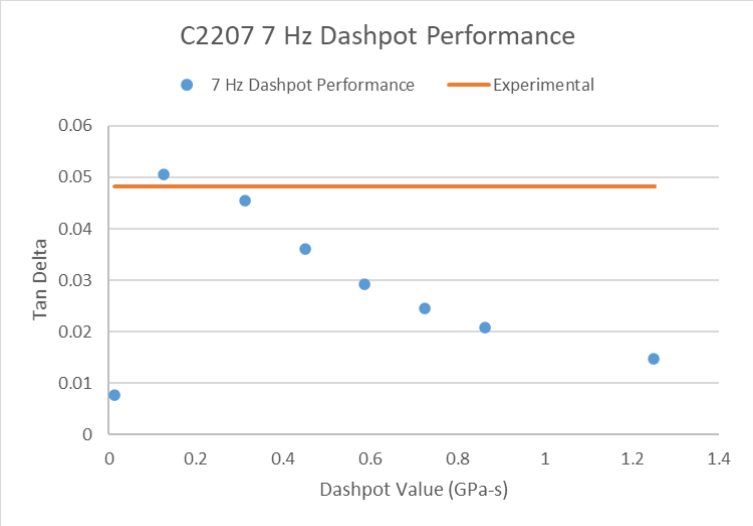

Figure 3.18: C2207 7 Hz Dashpot Performance

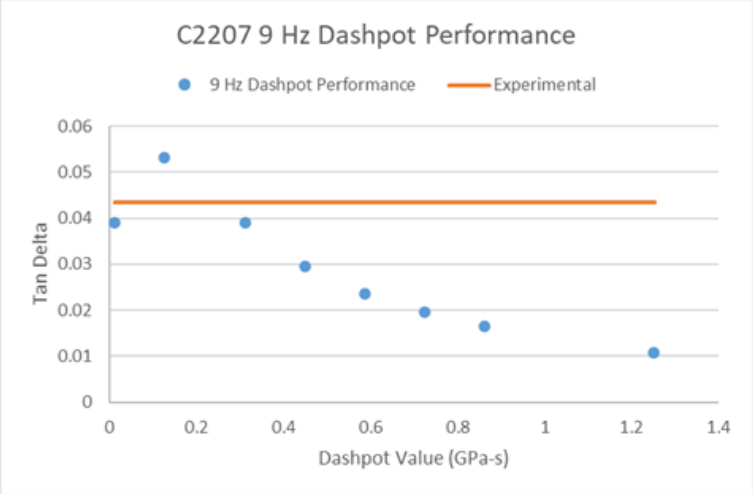

Figure 3.19: C2207 9 Hz Dashpot Performance

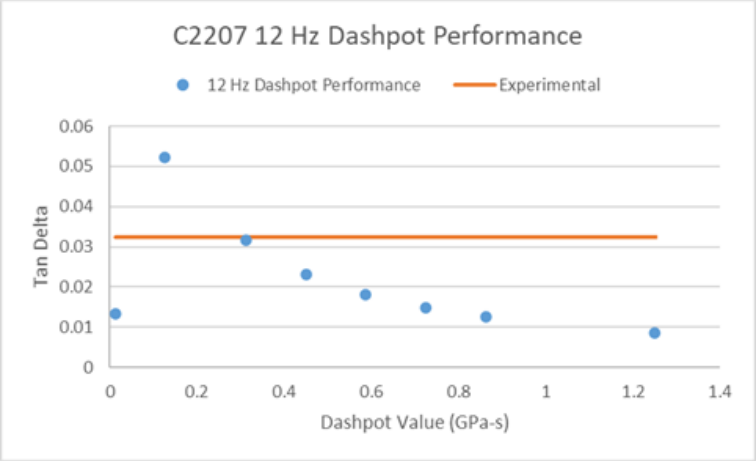

Figure 3.20: C2207 12 Hz Dashpot Performance

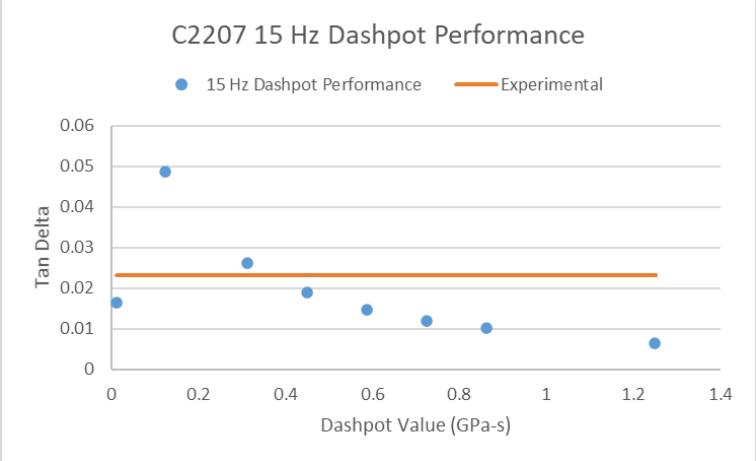

Figure 3.21: C2207 15 Hz Dashpot Performance

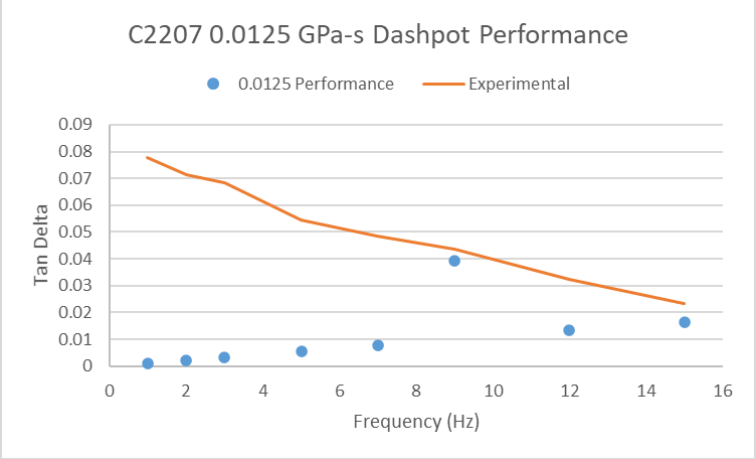

Figure 3.22: C2207 0.0125 GPa-s Dashpot Performance

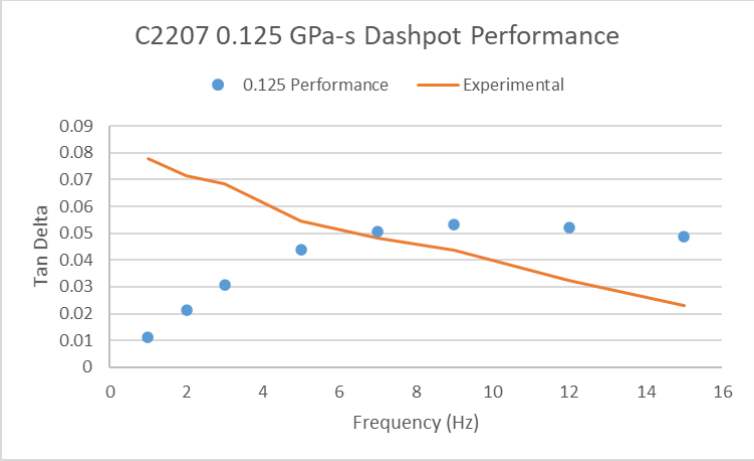

Figure 3.23: C2207 0.125 GPa-s Dashpot Performance

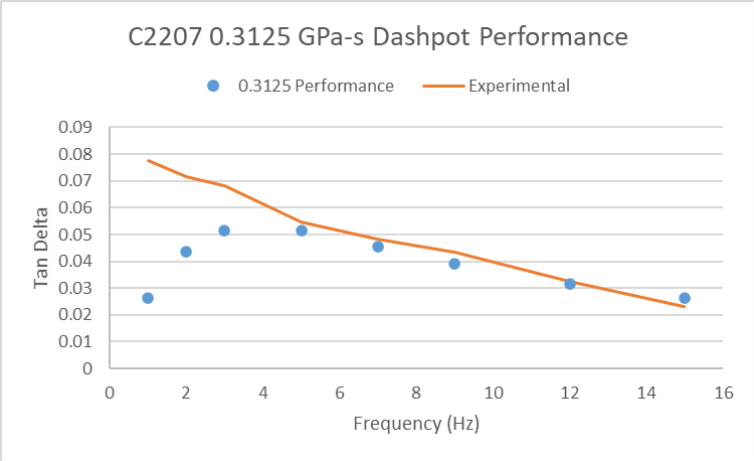

Figure 3.24: C2207 0.3125 GPa-s Dashpot Performance

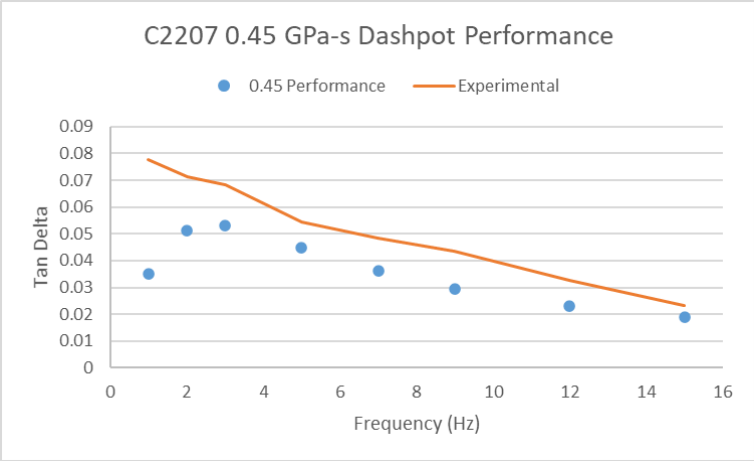

Figure 3.25: C2207 0.450 GPa-s Dashpot Performance

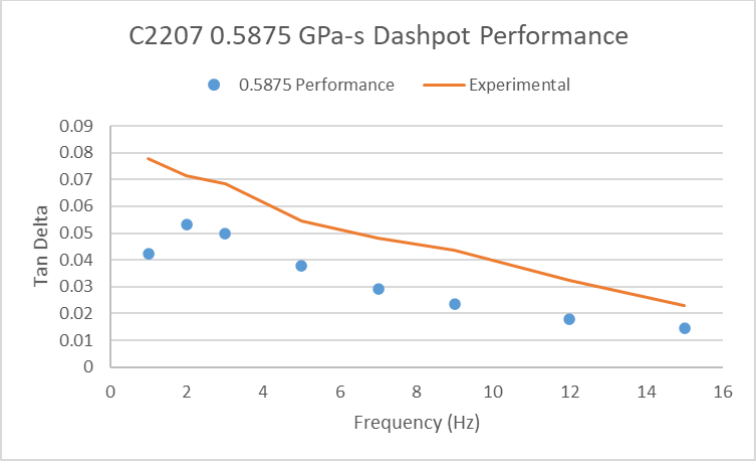

Figure 3.26: C2207 0.5875 GPa-s Dashpot Performance

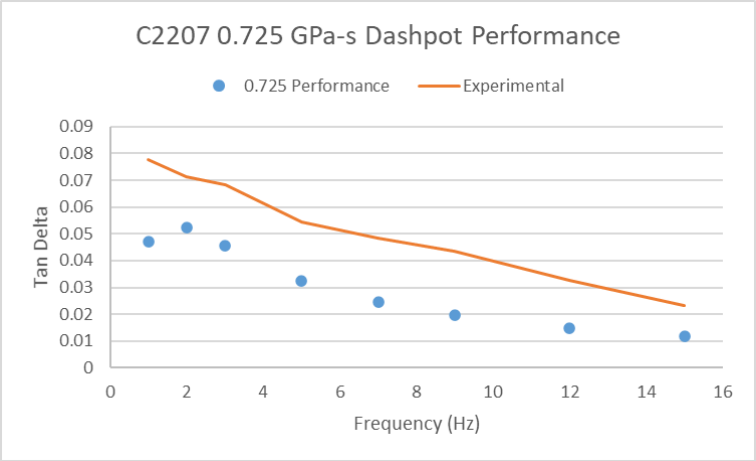

Figure 3.27: C2207 0.725 GPa-s Dashpot Performance

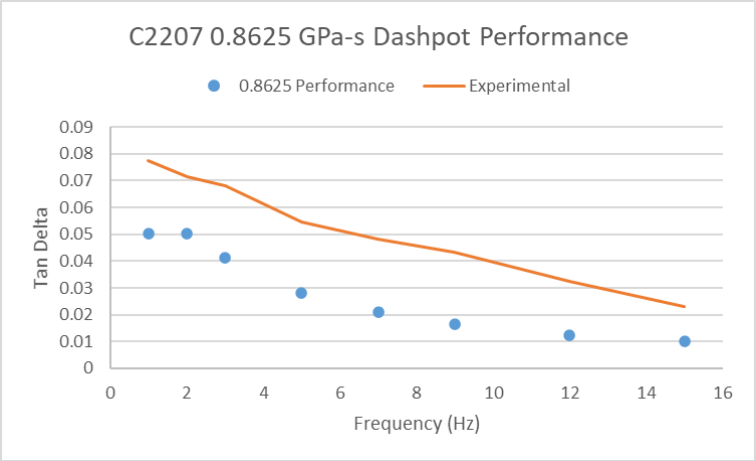

Figure 3.28: C2207 0.8625 GPa-s Dashpot Performance

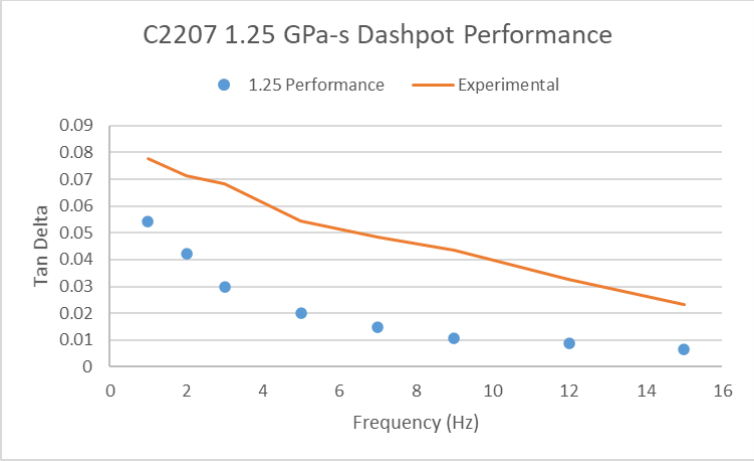

Figure 3.29: C2207 1.25 GPa-s Dashpot Performance

Table 3.4: C2207 Relative Dashpot Performance

Dashpot (GPa-s)	Tangent Delta by Frequency							
	1 Hz	2 Hz	3 Hz	5 Hz	7 Hz	9 Hz	12 Hz	15 Hz
0.0125	0.001040	0.002182	0.003401	0.005551	0.007782	0.010035	0.013228	0.016457
0.125	0.011039	0.021472	0.030560	0.043713	0.050658	0.053096	0.052128	0.048734
0.3125	0.026147	0.043713	0.051618	0.051668	0.045451	0.039143	0.031602	0.026260
0.45	0.035211	0.051073	0.053060	0.044782	0.036131	0.029616	0.023133	0.018930
0.5875	0.042167	0.053236	0.049975	0.037969	0.029317	0.023660	0.018089	0.014706
0.725	0.047154	0.052462	0.045670	0.032472	0.024463	0.019531	0.014801	0.012000
0.8625	0.050431	0.050248	0.041382	0.028170	0.020908	0.016596	0.012503	0.010127
1.25	0.054265	0.047450	0.030015	0.024785	0.018221	0.010845	0.010813	0.006540
C2207 DMA	0.077646	0.071401	0.068250	0.054588	0.048305	0.043483	0.032480	0.023132

$$\eta_{variable} = -6.61581 \cdot 10^{-5} f^5 + 2.972 \cdot 10^{-3} f^4 - 5.016075 \cdot 10^{-2} f^3 + 3.960115 \cdot 10^{-1} f^2 - 1.461599 f + 2.344206 \quad (3.1)$$

Table 3.4 illustrates the overall performance of each dashpot value for each frequency, with the best performing values highlighted in green. Figures 3.14 through 3.21 display the tangent delta of the various dashpots at each tested frequency with the C2207 Complex Model against C2207 DMA values at that same frequency. It should be emphasized that the blue marks depict performance of the model, and the orange line in each plot represents the experimental DMA tangent delta value at that partic-

ular frequency and is expressed as a horizontal line only for ease of visual comparison. Figures 3.22 through 3.29 show the performance of each dashpot value across every frequency, with the experimental tangent delta values in orange. Figure 3.30 shows the performance of the C2207 Complex Model run with the modified Richter UMAT detailed in Appendix E.2.1. The UMAT used a fifth order conversion polynomial (equation 3.1) that takes frequency input and outputs the dashpot value that was found to work best at that frequency, and was derived as described in section 2.4.3.2. It is noteworthy that the variable dashpot model produces very similar tangent delta values for the first three frequencies in spite of utilizing fairly different dashpot values via the polynomial employed: 1.23, 0.649 and 0.394 GPa-s for frequencies 1, 2, and 3 Hz respectively. Figure 3.31 shows the linear regression of the aforementioned results against the experimental DMA values. Figure 3.32 shows how the polynomial modified UMAT performed. If the polynomial achieved a perfect fit and the UMAT perfectly copied the polynomial, the R^2 would equal 1.

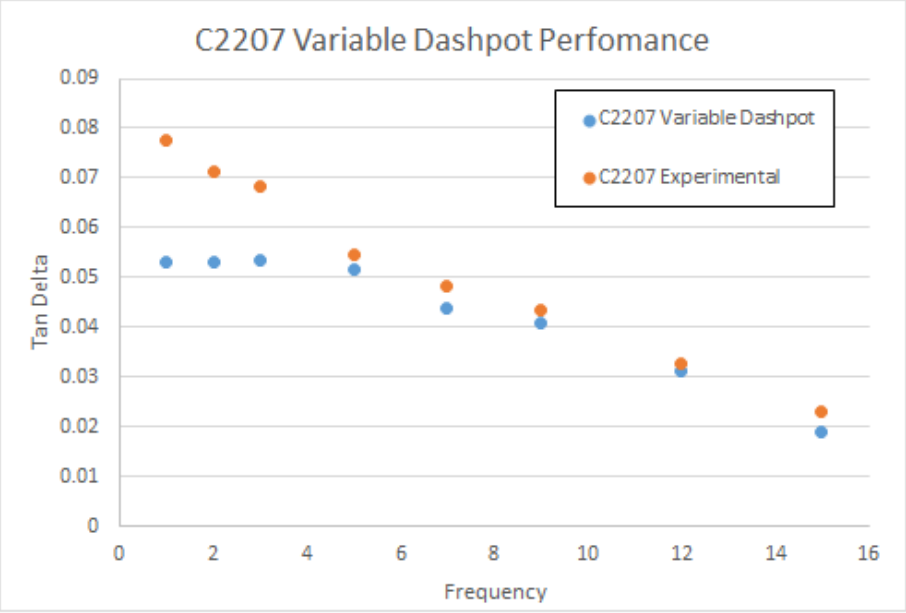

Figure 3.30: C2207 Variable Dashpot Performance

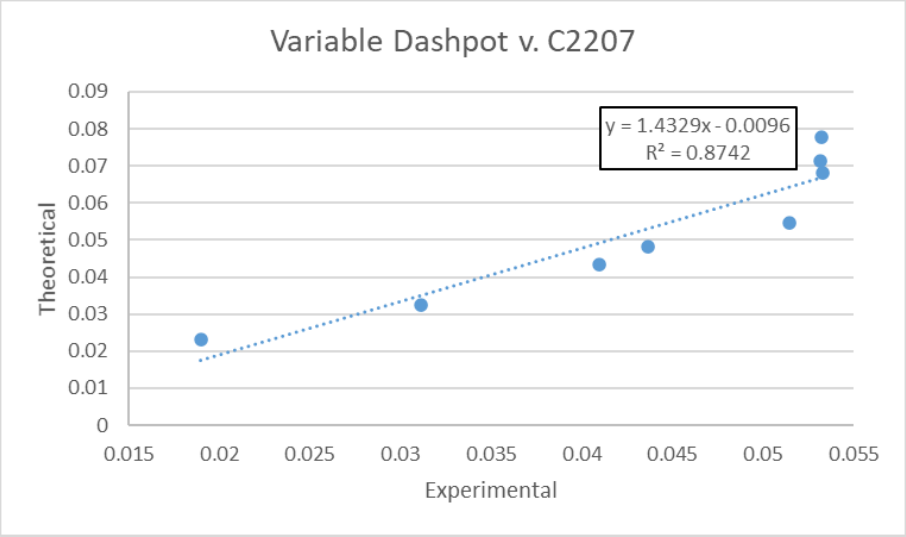

Figure 3.31: C2207 Variable Dashpot vs. Experimental Linear Regression (RMSE: 0.012)

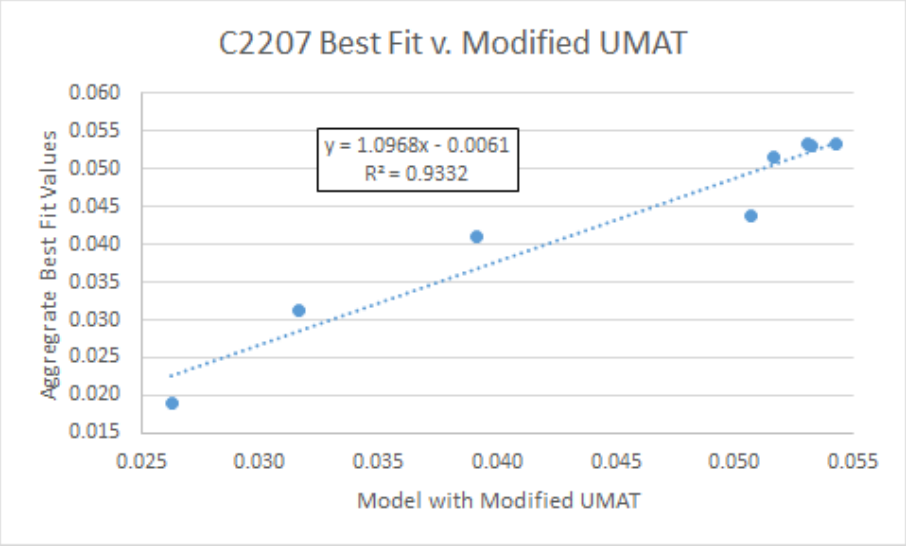

Figure 3.32: Performance of C2207 Modified UMAT: Linear Regression of Aggregate Tangent Delta values from Best Dashpots vs. Output of Model Running C2207 Modified UMAT

3.2.2 OVX Cranial Specimen C1809 Results

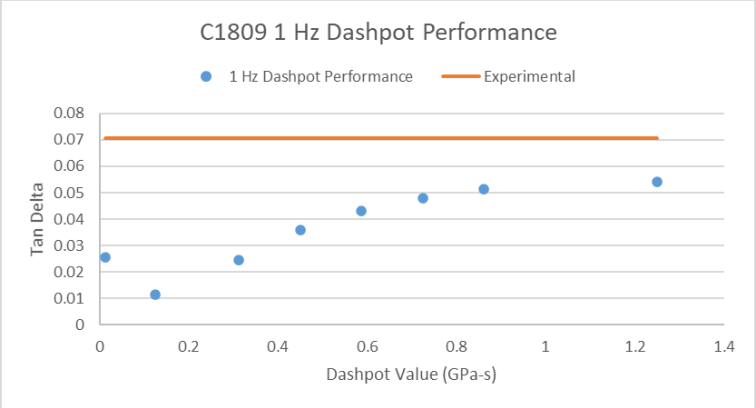

Figure 3.33: C1809 1 Hz Dashpot Performance

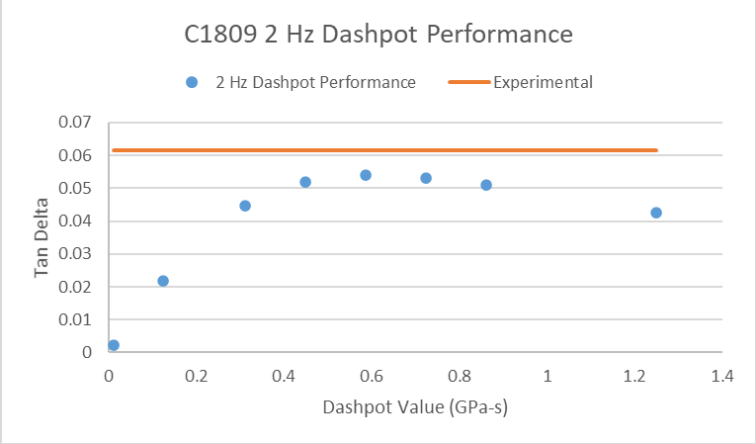

Figure 3.34: C1809 2 Hz Dashpot Performance

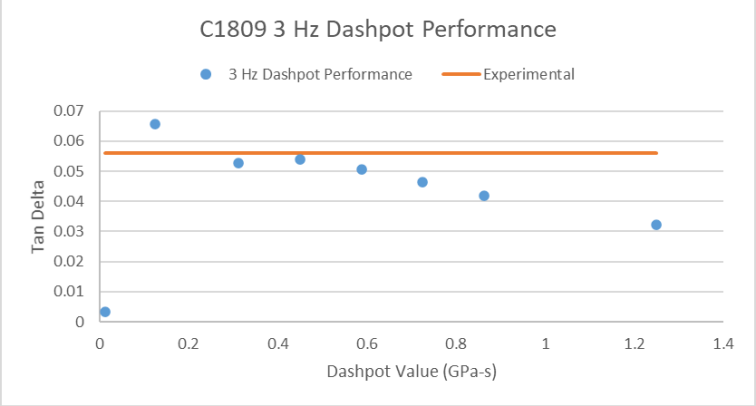

Figure 3.35: C1809 3 Hz Dashpot Performance

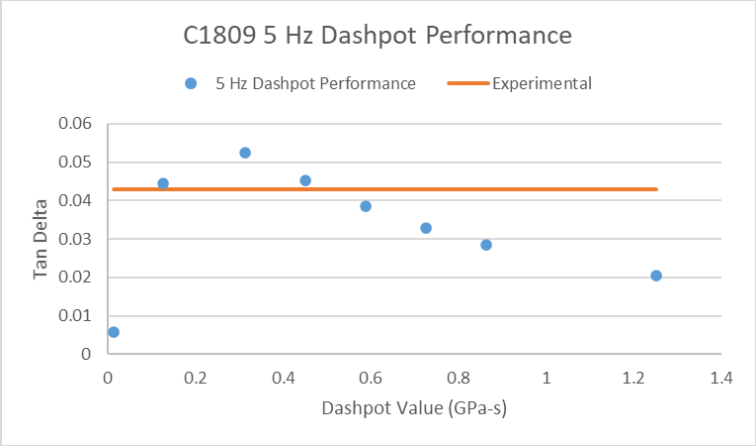

Figure 3.36: C1809 5 Hz Dashpot Performance

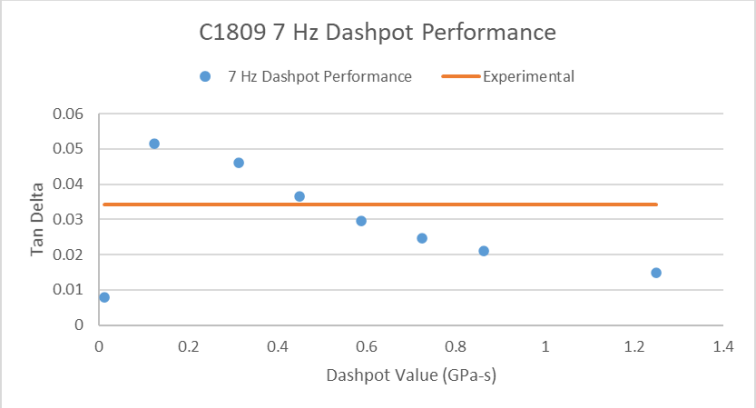

Figure 3.37: C1809 7 Hz Dashpot Performance

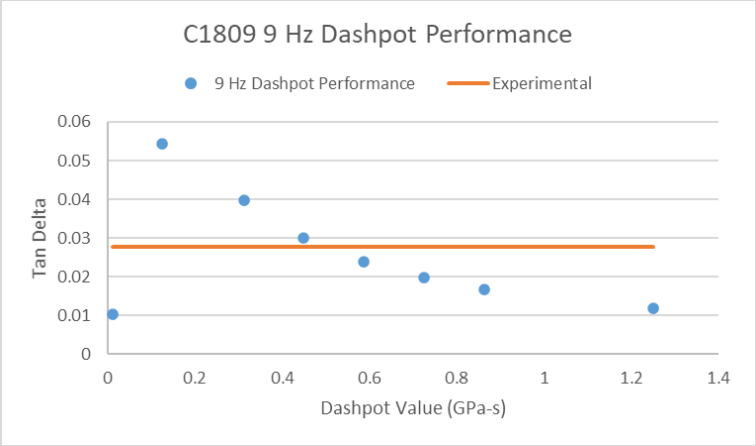

Figure 3.38: C1809 9 Hz Dashpot Performance

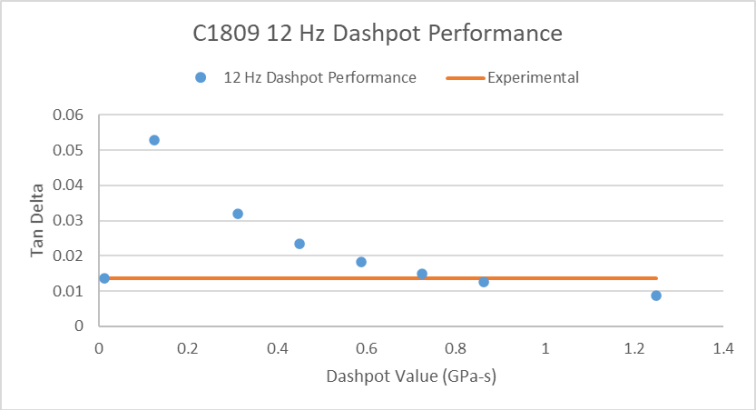

Figure 3.39: C1809 12 Hz Dashpot Performance

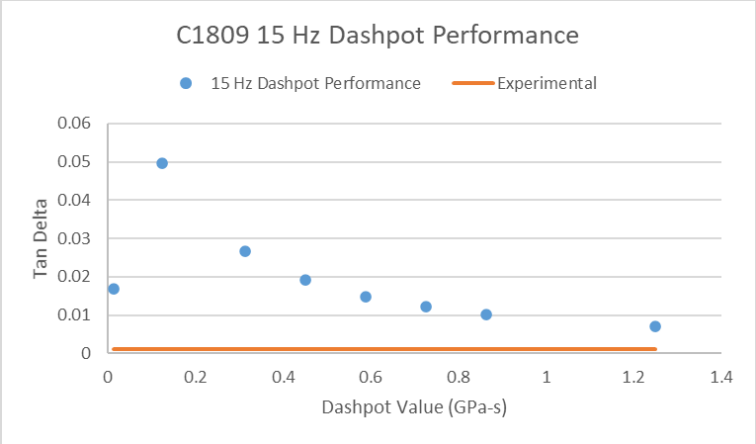

Figure 3.40: C1809 15 Hz Dashpot Performance

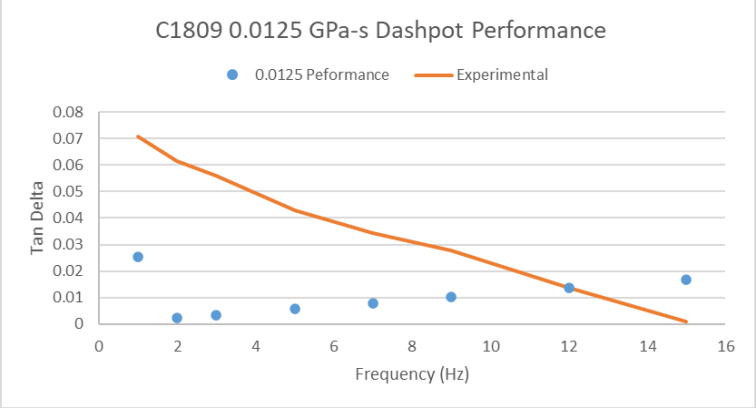

Figure 3.41: C1809 0.0125 GPa-s Dashpot Performance

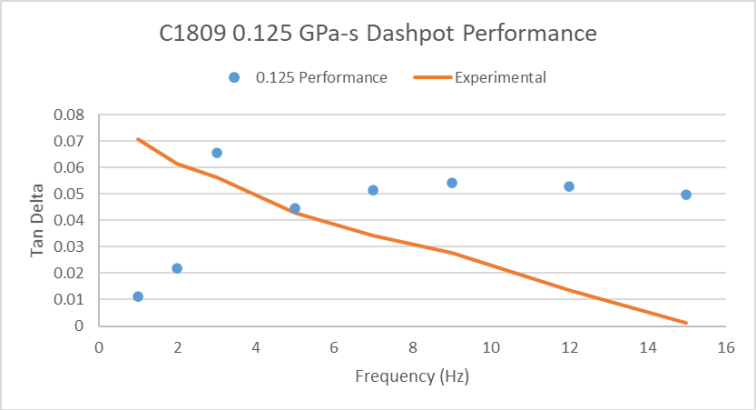

Figure 3.42: C1809 0.125 GPa-s Dashpot Performance

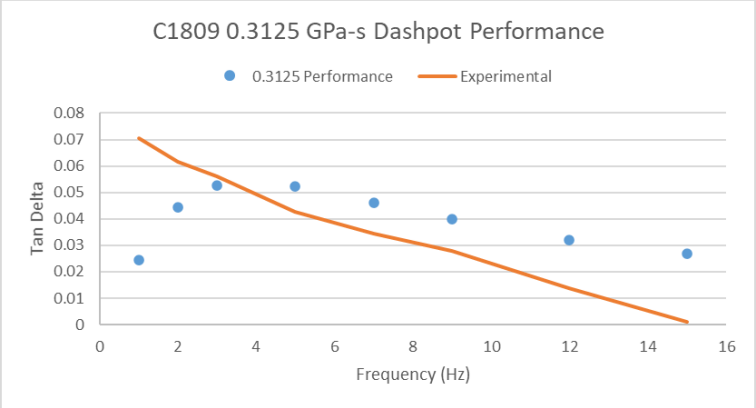

Figure 3.43: C1809 0.3125 GPa-s Dashpot Performance

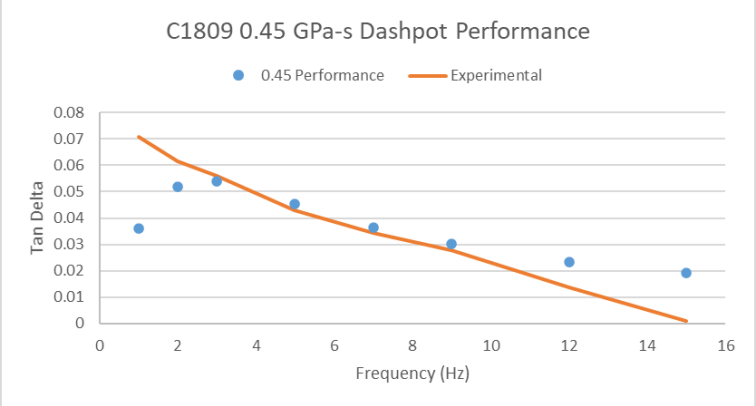

Figure 3.44: C1809 0.450 GPa-s Dashpot Performance

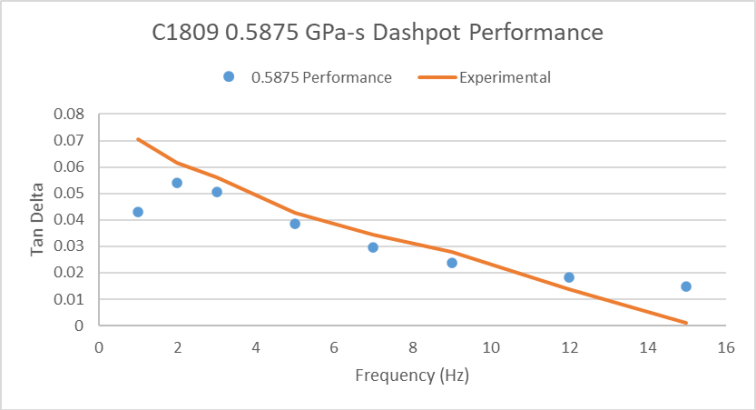

Figure 3.45: C1809 0.5875 GPa-s Dashpot Performance

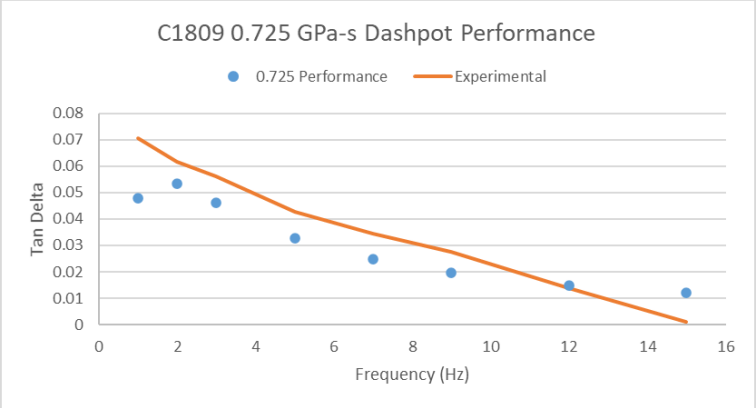

Figure 3.46: C1809 0.725 GPa-s Dashpot Performance

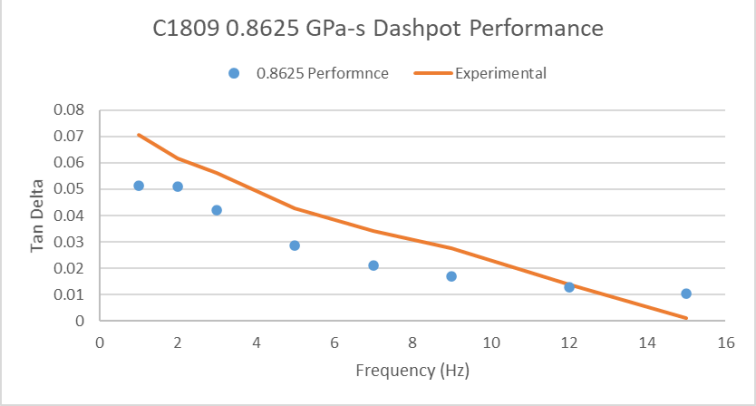

Figure 3.47: C1809 0.8625 GPa-s Dashpot Performance

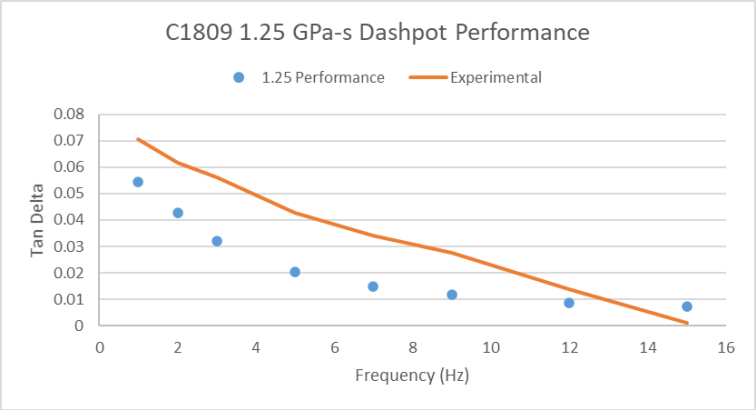

Figure 3.48: C1809 1.25 GPa-s Dashpot Performance

Table 3.5: C1809 Relative Dashpot Performance

Dashpot (GPa-s)	Tangent Delta by Frequency							
	1 Hz	2 Hz	3 Hz	5 Hz	7 Hz	9 Hz	12 Hz	15 Hz
0.0125	0.025432	0.002235	0.003490	0.005675	0.007952	0.010335	0.013518	0.016918
0.125	0.011366	0.021934	0.065657	0.044551	0.051544	0.054283	0.052884	0.049669
0.3125	0.024477	0.044553	0.052811	0.052411	0.046036	0.039828	0.031969	0.026687
0.45	0.035926	0.051957	0.053858	0.045356	0.036562	0.030108	0.023394	0.019141
0.5875	0.042985	0.054079	0.050660	0.038428	0.029656	0.023923	0.018288	0.014867
0.725	0.048023	0.053234	0.046260	0.032853	0.024737	0.019748	0.014964	0.012131
0.8625	0.051316	0.050943	0.041890	0.028492	0.021142	0.016777	0.012640	0.010235
1.25	0.054295	0.042700	0.032182	0.020476	0.014891	0.011758	0.008759	0.007114
C1809 DMA	0.070666	0.061492	0.056106	0.042801	0.034272	0.027731	0.013758	0.001091

$$\eta_{variable} = 9.0877 \cdot 10^{-5} f^5 - 2.45779 \cdot 10^{-3} f^4 + 1.53295 \cdot 10^{-2} f^3 + 6.84851 \cdot 10^{-2} f^2 - 0.83055 f + 1.97746 \quad (3.2)$$

Table 3.5 illustrates the overall performance of each dashpot value for each frequency, with the best performing values highlighted in green. Figures 3.33 through 3.40 display the tangent delta of the various dashpots at each tested frequency with the C1809 Complex Model against C1809 DMA values at that same frequency. Figures 3.41 through 3.48 show the performance of each dashpot value across every frequency, with the experimental tangent delta values in orange. Figure 3.30 shows the perfor-

mance of the C1809 Complex Model run with the modified Richter UMAT detailed in Appendix E.2.1. The UMAT used a fifth order conversion polynomial (equation 3.2) that took frequency inputs and outputted the dashpot value that was found to work best at that frequency, and was derived as described in section 2.4.3.2. Like C2207, a tangent delta plateau expands across the lowest frequencies, with the dashpot values utilized being 1.20, 0.617, and 0.24 for 1, 2, and 3 Hz respectively. Figure 3.50 shows the linear regression of the aforementioned results against the experimental DMA values. Finally, figure 3.51 shows how the polynomial modified UMAT performed. If the polynomial achieved a perfect fit and the UMAT perfectly copied the polynomial, the R^2 would equal 1.

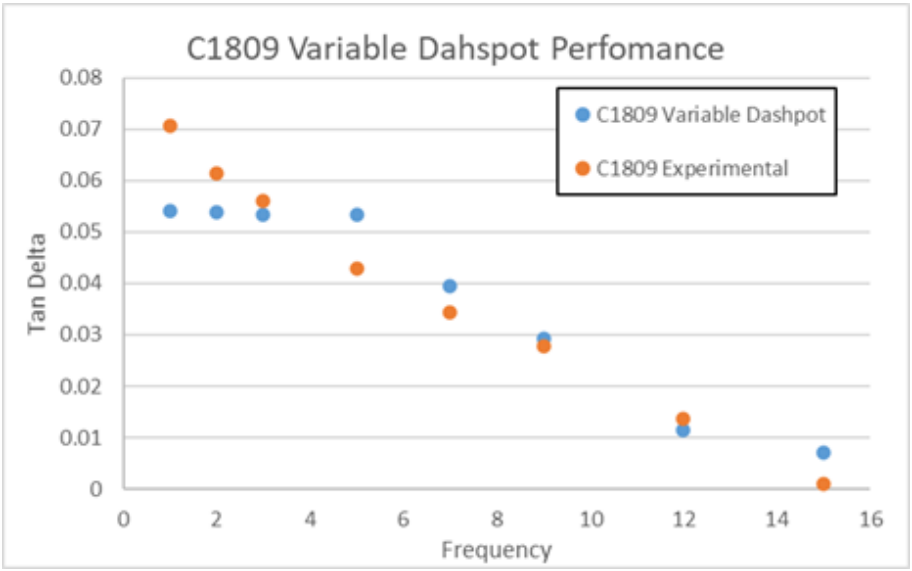

Figure 3.49: C1809 Variable Dashpot Performance

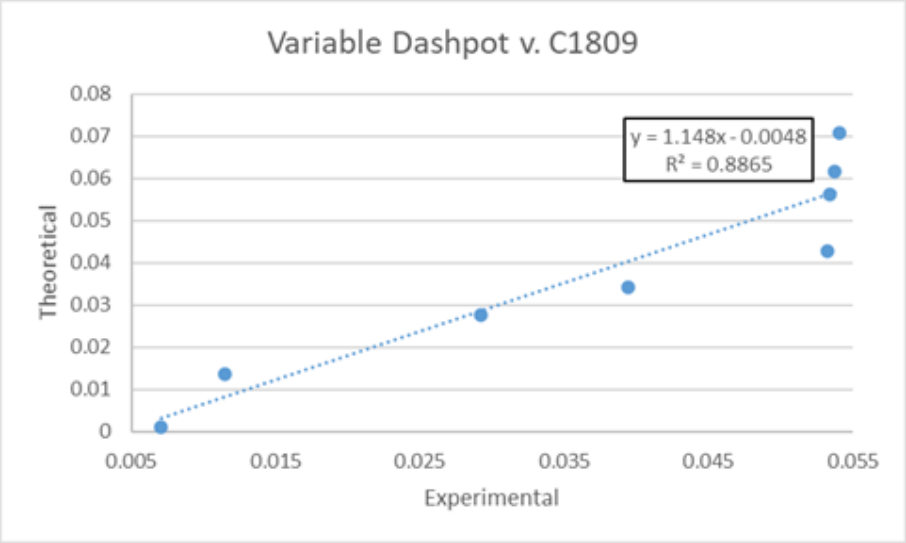

Figure 3.50: C1809 Variable Dashpot vs. Experimental Linear Regression (RMSE: 0.008)

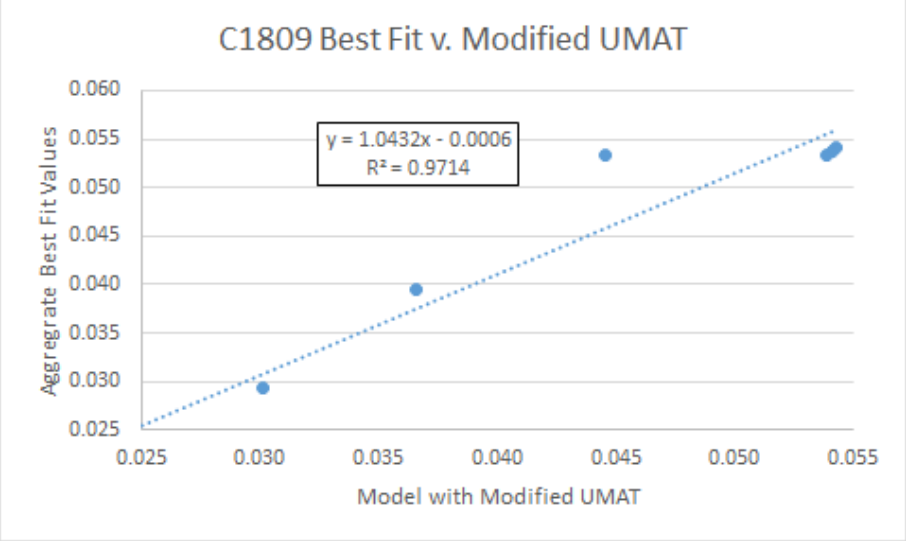

Figure 3.51: Performance of C1809 Modified UMAT: Linear Regression of Aggregate Tangent Delta values from Best Dashpots vs. Output of Model Running C1809 Modified UMAT

Chapter 4

DISCUSSION

4.1 IMF Scheme

There were two primary hopes for evolving the Complex Model into the IMF scheme: that it would make the model more biologically realistic and that it would produce tangent deltas more reminiscent of real world DMA values. The latter proved to not be the case, and it is not clear as to whether the former was fully accomplished either, which is a discussion worth having because the two hopes are probably not independent of each other in the statistical sense. Having a model that somehow perfectly captures the behavior of bone at this scale would undoubtedly converge on better tangent delta performance, but the partial or incomplete implementation of intermolecular forces could very well exacerbate rather than improve model fidelity. On the continuum towards realism it is possible that the model's performance could get worse before it gets better.

The IMF model that performed most closely to Mendoza's Normal D-Spacing model was the fully tie-constrained model, and this was completely expected because it allowed for the least amount of sliding between the nodes–every single adjacent node pair between parts was bound together like a single part just as all past iterations of the model were. Figures 3.1 and 3.2 reveal that the tie-constrained model performed very well at 1 Hz, performed very poorly in the middle frequencies and seems to be drifting back towards experimental at 15 Hz. High performance at 1 Hz was very surprising to see for this model, especially considering the $E_{effective}$ of 2 GPa, because past studies generally required some overall lower performing spring values to achieve

tangent deltas higher than about 0.055. While it is tempting to attribute more of the performance to the rheological parameter choices, the fidelity of the subsequent IMF models makes clear that the prime culprit of performance output lies in the methodology of attaching the separate parts.

The rest of the IMF models involved using the node randomization script (Appendix E.4) to implement different numbers of total crosslinks and Table 3.11 shows an overall poor performance when compared to experimental data and to the tie constraints model. While the R^2 value for the N=20 case clocks in at a slightly higher value than the rest of the randomized node models (roughly 0.81 vs 0.66), this isn't substantial and should be taken with a grain of salt. It should also be considered in light of the RMSE, which is more than double the fully tie-constrained model indicating significantly shoddier performance. It was of interest to this study to take stock of any delta in performance caused by the number of crosslinks. It was predicted that the greater number of crosslinks would result in tangent delta values closer to Mendoza's model and/or the fully tie-constrained model because the model would more closely resemble a configuration in which every adjacent surface was attached and could not move. This turned out to not be the case, with all randomized node RMSE values bearing a standard deviation of 0.000332, from N=20 to N=35, so those fifteen additional points of attachment did not seem to make a significant difference.

It bears repeating that each total crosslink count (e.g. N=20) had three separate models created in Abaqus with three separate randomized configurations of crosslink nodes. The performance of each model at each frequency was then averaged. This eliminated the possibility of heavily skewed node randomization producing bad data, and along with the uniformity of the RMSE for randomized node models makes the case that the IMF scheme approach needs improvement.

Approaches from Siegmund et al. were adopted for various reasons, but one important one was that the research managed to show a clear relationship between the number of non-enzymatic crosslinks and mechanical properties such as elastic moduli and toughness and I thought it likely that these clear differences would also manifest as differences in the viscoelastic properties in the Complex Model [69]. While it is difficult to deduce a relationship between toughness and viscoelastic properties, there are some clues about what crosslinking ought to do to the Complex Model's performance. In industrial rheology, especially in the studies of composites, it is generally found that crosslinking and additional attachments can have serious effects on viscoelasticity: crosslinks contribute a materials ability to "store" energy (i.e. resist displacement via elasticity) [63]. In terms of the viscoelastic properties stated in section 1.5, this means that a crosslinked material's complex modulus E^*, or its overall resistance to deformation, is composed of a larger E' or storage modulus component, sports a smaller phase angle δ, and a smaller tangent delta value than its non-crosslinked counterpart, all else being equal. All else being equal is important to remember because while it is initially tempting to try and make comparisons between the tie-constrained model and the randomized crosslink models it should be emphasized that the randomized crosslink models are not just the tie-constrained models with more connections–in fact they are *only* connected at their enzymatic crosslink sites whereas the tie-constrained model is connected at every adjacent node pair between parts. It is a shame then that the results showed a negligible difference in tangent delta performance between the IMF models with varying node counts, but there a few things that might readily be blamed for that.

The major potential pitfall are the differences in implementation itself of crosslinking between Siegmund's research and that conducted for this thesis. The former implementation was varied in its accounting of all present forces and each was relatively sophisticated in implementation [69]. Siegmund et al. accounted for various

forces, including a layer of structural water between the mineral and collagen that facilitates hydrogen bonding, collagen to collagen weak interactions mediated by hydrogen bonding, non-collagenous proteins and general electrostatic interactions, and of course enzymatic and non-enzymatic crosslinks [69]. Interactions in the model are defined across interfaces by a cohesive law that utilizes tractions and a potential function, which is a level of complexity that would require writing custom contact property equations in Abaqus to reproduce [69]. Each kind of interaction is intelligently placed spatially based on biology, and because this was primarily a study about failure each kind of interaction is rated at a sensible value for maximum shear and tensile stress based on literature values for hydrogen bond, N-C and C-C bond strength [69][70]. Because mechanical failure and delamination are not focuses of the Complex Model, accounting for such things as ultimate bond strength and tractions were not attempted to be implemented, and crosslinks were modeled as unbreakable tie constraints, with surfaces between them allowed to slide with basic default contact properties that allowed for friction and prevented overclosure between interfaces.

Considering the complexity and number of interactions happening in vivo between tropocollagen and minerals, it is possible that in a model unconcerned with mechanical failure, treating a half unit cell or long sequence of them as a single part may more closely approximate real intermolecular forces than eliminating those intermolecular forces and reintroducing them in a flawed or partial way. There do seem to be ways in Abaqus to better account for more of these forces in more elegant ways by way of custom interaction equations, but this approach requires considerable knowledge and skill, both of Abaqus and of modeling interactions. It remains an unfortunate fact that as long as the Complex Model is expressed as a single perfectly bonded part, no research involving a non-uniform distribution of strong interactions like crosslinks on viscoelastic performance is possible.

While it is tempting to encourage more research in this direction, there are serious caveats to prescribe. This approach deviates from the evolutionary trajectory of the Complex Model substantially in an approach laden with enough complexity and requisite effort that it might eclipse the practical timeline of any one Masters thesis. This is underlined by the fact that the IMF scheme was only carried out on the half-unit cell scale. Had it shown more promising performance, adopting utilities to the 2x100 Complex Model like the model generation script (Appendix E.3) and node randomization script (Appendix E.4) would prove to be colossal undertakings.

4.2 Variable Dashpot Scheme

Various previous studies have encouraged the investigation of a variable dashpot [35][74][16]. The most recent of these efforts executed extensive analysis on many different values of E_{HA}, E_1 and E_2 and achieved some configurations with very good fit, but still suggested that the dashpot value needed tuning and that it appeared to be frequency dependent because when holding all other rheological parameter values static, certain dashpot values yielded far better performance for different frequencies [74]. In the biased opinion of this researcher this seems like a smart suggestion for several reasons. In a system with multiple adjustable parameters such as the Complex Model, when choosing values it is only natural to start with what you know and move towards what you do not. As a start, the model's geometry, the boundary conditions and applied load are somewhat realistic and easy enough to correlate to real anatomy and DMA testing. Of the material values, if we allow for the interpretation of literature collagen modulus values as $E_{effective}$ and we suppose that E_{HA} can be extrapolated from studying highly mineralized bone, what's left is the dashpot value for hydrated, in vivo tropocollagen, a value that's a lot harder to extract from literature [86].

It is with that reasoning that models for sham and OVX of the cranial region were generated with the purpose of exploring more dashpot values across more frequencies, hoping to get a general sense of performance trends. Considering that the complex modulus E^* is composed of viscous and elastic components and that tangent delta is the ratio of the loss (viscous) over the storage (elastic) modulus, we would expect that increasing the dashpot value, all else being equal, would cause the tangent delta value to increase as seen in equation 4.1. For many viscoelastic materials it is observed that the components of E^* change their relative contributions as the loading frequency increases, and for bone we see tangent delta decreasing as the storage modulus increases and the loss modulus decreases as seen in equation 4.2. The material overall becomes more elastic and stiffens, which can help to prevent plastic flow (fracture).

$$Tan\ \delta \uparrow = \frac{Loss\ Modulus \uparrow}{Storage\ Modulus} \tag{4.1}$$

$$Tan\ \delta \downarrow = \frac{Loss\ Modulus}{Storage\ Modulus \uparrow} \tag{4.2}$$

Unfortunately as past researchers have discovered, the explanatory power of the preceding relationships becomes limited when you consider them in light of the standard linear solid. The Kelvin Voigt version of the standard linear solid used in Richter's UMAT is useful for modeling viscoelastic properties like creep and stress relaxation, but it also has one of its spring elements in parallel with the dashpot that's being changed in this research [65]. Claiming that increasing the dashpot value should allow for a steerable, straightforward decrease of tangent delta is misleading in a system like this because being in parallel with a spring means that while the stress between the spring and the dashpot can be different, the strains are equivalent between the two. The strain of the dashpot is dictated by its viscosity value and the strain rate

applied to it, which means that the strain of the spring is also dependent on the viscosity value, which means the storage modulus is dependent on the viscosity value. So in short, tangent delta will change but not in a predictable manner, especially over different frequencies.

Both Sham and OVX best fit dashpot values produced lower tangent delta values, almost reaching an invisible ceiling in the case of 1, 2, and 3 Hz of about 0.55. The maximum value itself is not surprising–Luke Thompson found that models that used effective moduli of 2 GPa, while overall performed better if used across all frequencies, limited 1 Hz tangent delta values similarly. For some of the dashpot performance it is tempting to get the impression that a global best fit has been found, in the case of C2207 2 Hz (figure 3.15), which features what appears to be a gradual sloping up to a maximum value, but in other figures such as C2207 1 Hz (figure 3.14) it seems that there may yet be an untested value greater than 1.25 GPa-s yielding a new maximum, or in the case of C2207 9 Hz (figure 3.19) a better fitting dashpot value could be found by visual inspection.

Consulting figures 3.41 through 3.48 makes it easier to visualize how one particular dashpot value compares against DMA over every frequency. Most plots show interesting maximum behavior in the lower frequencies where a peak is observed, but eventually slope downwards towards zero as the frequency increases and elastic behavior begins to dominate. These maximum tangent deltas are the point at which the storage modulus overtakes the loss modulus, and the derivative of tangent delta is negative. A general trend for increasing dashpot values is that these maximums tend to occur at lower frequencies until the peak for the 1.25 GPa-s dashpots seem to be out of focus at a frequency closer to zero, which is contrary to what would have been expected: it would have been expected that for the same storage modulus, a larger dashpot value would mean a larger loss modulus, delaying the frequency at

which the loss modulus is overtaken. It is also easy to see that basically all chosen dashpots give poor performance in the low frequencies with the possible exception of 1.25 GPa-s, and for higher frequencies why values like 0.3125 GPa-s were such strong performers in past studies, especially for C2207. It is hopeful in a sense to see some of the irregularities and hints of unseen global maxima, because if the the variable dashpot is going to account for the low performance in low frequencies while using an effective modulus of 2 GPa, it seems likely that testing many more frequencies of dashpots, especially at 1 Hz, is necessary.

These very best performing dashpots were taken and plotted against frequency for C1809 (figure 2.21) and C2207 (figure 2.22) and a fifth order polynomial was fit for each and used to modify Richter's original UMAT. The UMAT would now take frequency as an input for that particular run, run it through the fitting polynomial, and pass Abaqus a dashpot value that had been shown to work best at that frequency. Figures 3.32 and 3.51 show how closely the modified UMAT managed to mimic the numbers of Tan Delta values that were aggregated from the best fit dashpots. It is immediately obvious that there are some imperfections with the line fit, especially with C1809. It is also obvious that C1809's best fit dashpot v. frequency plot looks significantly different from C2207's. Both plots indicate that drastically higher dashpots are better suited for 1 Hz, but where C2207 settles into mostly being covered by 0.3125 GPa-s, C1809 is better served by values above and below that, and at 15 Hz a much higher dashpot value of 1.25 GPa-s works best. Consulting figures 3.40 and 3.48 show that an even better, higher dashpot value is probably waiting to be tested for C1809 at 15 Hz and that 1.25 GPa-s coincides best with experimental DMA values at that frequency because it is one of the first dashpot values to cause a tangent delta peak in the lower frequencies, slowly drifting towards zero from early on.

Such stark differences between all of these plots between C1809 and C2207 is interesting, because Wallace et al. found that for specimen in the same anatomical region, the distinct difference between sham and OVX D-Spacing was not so much the mean but the variance [78]. Specifically the D-Spacing for sham C2207 used to build the model was a mean of 67.44 ± 1.16 nm compared to 67.15 ± 2.00 nm, which is a mean percent difference of just 0.43% but a standard deviation percent difference of 42%. This trend holds throughout the DMA data set. Table 4.1 shows that in general there is a significantly greater percent difference when comparing the standard deviations of sham vs. OVX than when comparing their means, in every case at least an order of magnitude greater. While this research has shown that there is much greater variability in the best fit dashpot values for OVX vs. sham, a natural next question would be: does this trend hold if more OVX and sham specimen models were generated and tested that share the same anatomical site? If this pattern persisted, it would definitely be tied to the difference in standard deviation of D-Spacing between sham and OVX, and it would fall to biologists to interpret why this might be the case.

Table 4.1: Experimental D-Spacing Mean and Standard Deviation: Averages were taken for the D-Spacing Means and Standard Deviations for each sector for a basis of comparison between sham and OVX with units in nm

		OVX	Control	% Diff
Sector 1 Cranial	Mean:	66.8560	68.8070	-2.8%
	Standard Dev:	2.0859	0.9828	112.2%
Sector 2 Cranial	Mean:	67.0223	68.0329	-1.5%
	Standard Dev:	1.2996	1.9564	-33.6%
Sector 5 Caudal	Mean:	68.0838	67.2632	1.2%
	Standard Dev:	1.0705	1.7909	-40.2%
Sector 6 Caudal	Mean:	67.0886	65.4439	2.5%
	Standard Dev:	1.8616	2.6988	-31.0%

Research from Les et al. has found that at frequencies 3 Hz and below, bone from OVX and Sham share small statistically significant differences in storage modulus and

by extension tangent delta, but that in higher frequencies the three year effects of ovariectomy the efficiency of damping oscillatory stresses in OVX trailed significantly [44]. This general pattern can be observed in figure 2.23, which shows both the models and the experimental values compared side by side and provides a sanity check for this research.

Zooming out, the original intention of the Complex Model is to establish a model of particular geometry, forces and material that matches experimental DMA results as closely as possible. The variable dashpot model has done so with respectable R^2 and RMSE values, which should be evaluated in the following context: while previous results have yielded higher fit, said results were aggregates of best fit values for the highest performing variables from all of the major adjustable material parameters (E_1, E_2, E_{HA} and η_1) in the Complex Model, which prompted Luke Thompson to remind readers that you can get the model to fit if you force it [74]. The variable dashpot scheme does manage something special in that material constants in the input file are held constant throughout every frequency for a given model. Using Kelvin Voigt standard linear solids to capture the viscoelastic behavior has great utility, but the model is still probably not ready to deduce biological insights from, because attributing spring and dashpot values in series and parallel to biological structures can prove dubious, even when zooming out to the storage and loss modulus because of the interdependent nature of E_2 and η_1.

There are a few areas for improvement and further research with the Complex Model, large and small endeavors. In the short term there is ample room for more digging to find an even more optimized dashpot values at each frequency. As previously mentioned, while many of the dashpot performance figures in the results section appear to present at least a local (and hopefully global) maximum tangent delta value, some of the figures seem to suggest that the maximum has not yet been found

and that with some more searching with existing utilities and input files a better polynomial could be achieved, perhaps one that could break the tangent delta upper limits seemingly imposed on the lower frequencies. Another short term approach that would use the current VD scheme would be to build models with D-Spacing values from specimen from other sectors. This would confirm or refute aforementioned suspicions about the differences between sham and OVX highlighted in table 4.1 manifesting stark differences in the best dashpot values vs. frequency for a model, and therefore the required polynomial. Showing that this is a trend across all specimen would make for a more compelling biological mystery to be solved.

For longer term investigations, the same treatment of frequency dependency could be applied to the effective modulus. On the upside, doing so would probably be a surefire way to break the tangent delta thresholds seen in the lower frequencies, as earlier studies were able to do so using lower values for the springs, in some cases going as low as 200 MPa [74]. The issues with doing this are that unlike the dashpot value, the springs values are, with some poetic interpretation, informed by material properties found in literature and interpreted into this model by way of an effective modulus. It might make a biological interpretation even more difficult, and it might cause the cycle of calibration to continue: imperfections in a fit of such a variable effective modulus would provoke tinkering of variable dashpot polynomials, and so on.

Any additional investigation would be greatly aided by a deeper knowledge and application of computer science. A large portion of the effort undertaken in this research involves the switching between scripts, file types, programs, in a lengthy workflow that is ripe for skilled streamlining. For the full-bore 2x100 half unit cell Complex Model, output database files are generally around 4.6 GB in size and take about 2.5 hours to run. So for example, exploring the performance of a new dashpot value

across eight frequencies for a particular model with all else held constant takes 20 hours of Abaqus computation time alone on the existing dedicated workstations, probably even longer on a virtual machine. This research made great use of things like batch files for running multiple jobs, but a skilled researcher could write scripts to do much more. The IMF scheme demonstrated some finicky but capable python scripts that read in parts of an input file, did some light calculations, and wrote the results, properly formatted, back into the input file. Using these same functions a python script could (and should) be written to take an input file devised for one frequency loading condition and generate input files for every other frequency loading condition. Currently, mercifully, batch scripts run successive Abaqus jobs, call other scripts to move around data and extract terminal node displacements from the ODB file to generate displacement text files for MATLAB post processing scripts, which are currently run manually, ten times, for every frequency, and the tangent delta value is manually copied to a spreadsheet for analysis. All of that could absolutely be done with python because it's possible to call MATLAB via python. Implementing some or all of these shortcuts would save a huge amount of time, and would reduce a currently daunting multi-step process into something that an undergrad with FEA coursework could easily execute and probably even troubleshoot.

This study has provided insight into the specific performance of dashpot values across a large spectrum and their marked differences between sham and OVX, and has generated variable dashpot UMATs that provide good fit with all other material constants held static. It has also highlighted the difficulty and nuance of modeling multiple intermolecular forces at randomized locations for even simple finite element models in Abaqus. It has also highlighted clear areas of future study and improvement in the service of a higher fidelity computational model of bone mechanics.

Chapter 5

CONCLUSION

The purpose of this undertaking and conception of the Complex Model in general was to better characterize the many factors that contribute to bone's mechanical behavior on the macroscale so that diseases like osteoporosis can be better understood by the nanoscale spatial changes that they are associated with, something relevant to a huge swath of the global population. A highly predictive and tractable model would allow for researchers to make predictions about how changes in material constants, geometry, and the mechanisms and number of interconnections would affect macroscale properties such as toughness and force dampening.

The study of viscoelastic composite materials is an incredibly complex endeavour. Bone being biological adds a number of new wrinkles including the difficulty of finding relevant literature values for material constants, attributing model behaviors to biological structures, and assuming that in vivo behavior can be reasonably reproduced in testing. The scale of the study also introduces issues for testing and assumptions about whether or not bone can be treated as a continuum, or whether to classify it as orthotropic or anisotropic.

An attempt to model the various forces that adhere collagen and tropocollagen together was explored, with clear lessons learned about the best direction for future developments of the Complex Model. Additionally at the suggestion of past researchers of the Complex Model a frequency dependent dashpot scheme was sought out, with eight dashpot values tested over eight frequencies between two models. Resultant experimental data were used to formulate a model-specific custom UMAT

whose dashpot value changed with frequency. This achieved models that fit C1809 OVX and C2207 Control experimental data with R^2 values of 0.89 and 0.87 and RMSE values of 0.008 and 0.012, respectively. These values were achieved by keeping all other material constants throughout eight frequencies.

It was found that the dashpot values that performed best at each frequency when compared to the control specimen were fairly consistent with the exception of the lower frequencies needing significantly larger dashpot values. The dashpot values that performed best at each frequency when compared against the sham specimen were much more variable than the control specimen. While the sham model also favored higher dashpot values at very low frequencies, 15 Hz also required a relatively large dashpot value. These stark differences in optimal dashpot values between control and OVX models suggest that a difference in D-Spacing standard deviation can change bone's ability to store and dissipate energy.

These details open doors for further research into the increased tuning of the Complex Model by way of refinement of the variable dashpot fits, variable dashpot fits for more specimen, biological interpretation of dashpot performance differences between control and OVX specimen, and a streamlining of the disparate portions of the protocol, hopefully evolving into a unified, highly usable program for predicting the macroscale viscoelastic behavior of bone.

BIBLIOGRAPHY

[1] National Academy, National Academy, and United States. A Subunit Model for the Tropocollagen Macromolecule Author (s): John A . Petruska and Alan J . Hodge Source : Proceedings of the National Academy of Sciences of the United States of America , Published by : National Academy of Sciences Stable URL : http. 51(5):871–876, 2020.

[2] F Akyildiz, R S Jones, and K Walters. Short Communications On the spring-dashpot representation of linear viscoelastic behaviour. Technical report, 1990.

[3] Yuehuei H. An. Mechanical properties of bone. In *Mechanical Testing of Bone and the Bone-Implant Interface*, pages 41–64. 1999.

[4] Asylum Research. Atomic Force Microscopes - MFP-3D Manual. pages 1–321, 2008.

[5] Allen J. Bailey, Robert Gordon Paul, and Lynda Knott. Mechanisms of maturation and ageing of collagen. *Mechanisms of Ageing and Development*, 106(1-2):1–56, dec 1998.

[6] Donald Bartel, Dwight Davy, and Tony M Keaveny. *Orthopaedic Biomechanics: Mechanics and Design in Musculoskeletal Systems*. Pearson, 1st editio edition, 2007.

[7] Martin K. Beyer. The mechanical strength of a covalent bond calculated by density functional theory. *The Journal of Chemical Physics*, 112(17):7307–7312, may 2000.

[8] Orlaith Brennan, Julia S. Kuliwaba, T. Clive Lee, Ian H. Parkinson, Nicola L. Fazzalari, Laoise M. McNamara, and Fergal J. O'Brien. Temporal Changes in Bone Composition, Architecture, and Strength Following Estrogen Deficiency in Osteoporosis. *Calcified Tissue International*, 91(6):440–449, dec 2012.

[9] P.E. Bunney, A.N. Zink, A.A. Holm, C.J. Billington, and C.M. Kotz. Orexin activation counteracts decreases in nonexercise activity thermogenesis (NEAT) caused by high-fat diet. *Physiology & Behavior*, 176(3):139–148, jul 2017.

[10] Albert H Burstein and Donald T Reilly. The elastic and ultimate properties of compact bone tissue. *Journal of Biomechanics*, 8(6):393–405, 1975.

[11] Linyi Cai, Xin Xiong, Xiangli Kong, and Jing Xie. The Role of the Lysyl Oxidases in Tissue Repair and Remodeling: A Concise Review. *Tissue engineering and regenerative medicine*, 14(1):15–30, feb 2017.

[12] Carter. Bone Compressive Strength : The Influence of Density and Strain Rate Author (s): Dennis R . Carter and Wilson C . Hayes. 194(4270):1174–1176, 2014.

[13] D R Carter and W C Hayes. The compressive behavior of bone as a two-phase porous structure. *The Journal of bone and joint surgery. American volume*, 59(7):954–62, oct 1977.

[14] Dennis R Carter and Wilson C Hayes. Bone Compressive Strength: The Influence of Density and Strain Rate. *Science*, 194(4270):1174–1176, 1976.

[15] A. C. Courtney, E. F. Wachtel, E. R. Myers, and W. C. Hayes. Effects of loading rate on strength of the proximal femur. *Calcified Tissue International*, 55(1):53–58, 1994.

[16] Austin Cummings. A Finite Element Analysis on the Viscoelasticity of Postmenopausal Compact Bone Utilizing a Complex Collagen D-Spacing Model. Master's thesis, California Polytechnic State University, 2015.

[17] John Currey. Incompatible mechanical properties in compact bone. *Theoretical Biology*, 231:569–580, 2004.

[18] John D. Currey. *The mechanical adaptations of bones*. Princeton University Press, 1984.

[19] John D. Currey. *The Mechanical Properties of Bone from The Mechanical Adaptations of Bones*. Princeton University Press, 1984.

[20] C. R. Deuel, A. A. Jamali, S. M. Stover, and S. J. Hazelwood. Alterations in femoral strain following hip resurfacing and total hip replacement. *Bone & Joint Journal*, 91-B(1), 2008.

[21] C. Ross Ethier and Craig A. Simmons. *Introductory Biomechanics*. Cambridge University Press, Cambridge, 1st edition, 2007.

[22] D. Eyre. Cross-Linking in Collagen and Elastin. *Annual Review of Biochemistry*, 53(1):717–748, jan 1984.

[23] D. R. Eyre, I. R. Dickson, and K. Van Ness. Collagen cross-linking in human bone and articular cartilage. Age-related changes in the content of mature hydroxypyridinium residues. *Biochemical Journal*, 252(2):495–500, 1988.

[24] David R. Eyre and Jiann-Jiu Wu. Collagen Cross-Links. In *Topics in Current Chemistry*, volume 247, pages 207–229. apr 2005.

[25] Ming Fang, Elizabeth L. Goldstein, A. Simon Turner, Clifford M. Les, Bradford G. Orr, Gary J. Fisher, Kathleen B. Welch, Edward D. Rothman, and Mark M. Banaszak Holl. Type i collagen D-spacing in fibril bundles of

dermis, tendon, and bone: Bridging between nano- and micro-level tissue hierarchy. *ACS Nano*, 6(11):9503–9514, 2012.

[26] Ming Fang, Kaitlin G Liroff, A Simon Turner, Clifford M Les, Bradford G Orr, and Mark M Banaszak Holl. Estrogen depletion results in nanoscale morphology changes in dermal collagen. *The Journal of investigative dermatology*, 132(7):1791–7, 2012.

[27] Liang Feng. *Multi-scale characterization of swine femoral cortical bone and long bone defect repair by regeneration*. PhD thesis, University of Illinois, 2010.

[28] William Findley, James Lai, and Kasif Onaran. *Creep and Relaxation of Nonlinear Viscoelastic Materials*. Dover Publications, Inc., New York, NY, 1989.

[29] Harold M. Frost. *Bone Remodelling Dynamics*. Charles C Thomas, 1963.

[30] Patrick Garnero, Olivier Borel, Evelyne Gineyts, Francois Duboeuf, Helene Solberg, Mary L. Bouxsein, Claus Christiansen, and Pierre D. Delmas. Extracellular post-translational modifications of collagen are major determinants of biomechanical properties of fetal bovine cortical bone. *Bone*, 38(3):300–309, mar 2006.

[31] Amit Gefen and Ramat Aviv. *Skeletal Aging and Osteoporosis*, volume 5. Volume 5 edition, 2013.

[32] Ronald F. Gibson, Yu Chen, and Hui Zhao. Improvement of Vibration Damping Capacity and Fracture Toughness in Composite Laminates by the Use of Polymeric Interleaves. *Journal of Engineering Materials and Technology*, 123(3):309–314, jan 2001.

[33] J E Gordon. *Structures or Why things don't fall down.* Springer US, Boston, MA, 1978.

[34] Alan Grafen and Rosie Hails. *Modern Statistics for the Life Sciences.* Oxford University Press, New York, NY, 2002.

[35] Christopher Ha. Modeling Viscoelastic Behavior in Compact Bone through a Distribution of Collagen D-Spacing: A Finite Element Analysis. Master's thesis, California Polytechnic State University, San Luis Obispo, 2015.

[36] Daniel Wayne Hale. *The Effects of Bisphosphonates on Bone Remodeling: Analysis of Microdamage Targeting by BMUs, BMU Velocity and Crack Surface Density.* PhD thesis, California Polytechnic State University, San Luis Obispo, California, jun 2008.

[37] Elham Hamed, Yikhan Lee, and Iwona Jasiuk. Multiscale modeling of elastic properties of cortical bone. *Acta Mechanica*, 213(1-2):131–154, 2010.

[38] Ingomar Jäger and Peter Fratzl. Mineralized Collagen Fibrils: A Mechanical Model with a Staggered Arrangement of Mineral Particles. *Biophysical Journal*, 79(4):1737–1746, oct 2000.

[39] Saul J. Kaplan, Wilson C. Hayes, John L. Stone, and Gary S. Beaupré. Tensile strength of bovine trabecular bone. *Journal of Biomechanics*, 18(9):723–727, jan 1985.

[40] T M Keaveny and W C Hayes. *A 20-year perspective on the mechanical properties of trabecular bone*, volume 115. American Society of Mechanical Engineers, nov 1993.

[41] Tariq A Khraishi and Yu-lin Shen. *Continuum Mechanics Basic Principles of Vectors, Tensors, and Deformation.* Cognella Academic Publishing, 2015.

[42] Lynda Knott and A. J. Bailey. Collagen cross-links in mineralizing tissues: A review of their chemistry, function, and clinical relevance. *Bone*, 22(3):181–187, 1998.

[43] Roderic S. Lakes. *Viscoelastic Materials*. Cambridge University Press, New York, NY, 2009.

[44] C. M. Les, J. L. Vance, G. T. Christopherson, A. S. Turner, G. W. Divine, and D. P. Fyhrie. Long-term ovariectomy decreases ovine compact bone viscoelasticity. *Journal of Orthopaedic Research*, 23(4):869–876, jul 2005.

[45] C. M. Les, J. L. Vance, G. T. Christopherson, A. S. Turner, G. W. Divine, and D. P. Fyhrie. Long-term ovariectomy decreases ovine compact bone viscoelasticity. *Journal of Orthopaedic Research*, 23(4):869–876, jul 2005.

[46] C.M. Les, C.A. Spence, J.L. Vance, G.T. Christopherson, B. Patel, A.S. Turner, G.W. Divine, and D.P. Fyhrie. Determinants of ovine compact bone viscoelastic properties: effects of architecture, mineralization, and remodeling. *Bone*, 35(3):729–738, sep 2004.

[47] Stavros C. Manolagas. The Role of IL-6 Type Cytokines and Their Receptors in Bonea a. *Annals of the New York Academy of Sciences*, 840(1):194–204, may 1998.

[48] Elaine N. Marieb and Katja N. Hoehn. *Human Anatomy and Physiology*. Benjamin-Cummings Publishing Company, 9th edition, 2012.

[49] R. Bruce Martin, David B. Burr, Neil A. Sharkey, and David P. Fyhrie. *Skeletal Tissue Mechanics*. Springer New York, New York, NY, second edition, 2015.

[50] G Mase and George Mase. *Continuum Mechanics for Engineers*. CRC Press, third edit edition, jun 1999.

[51] H Mcelhaney and West Virginia. Dynamic response_. 2018.

[52] J H McElhaney. Dynamic response of bone and muscle tissue. *Journal of Applied Physiology*, 21(4):1231–1236, jul 1966.

[53] J H McElhaney. Dynamic response of bone and muscle tissue. *Journal of Applied Physiology*, 21(4):1231–1236, dec 2017.

[54] Miguel Mendoza. The Effects of Variation in Collagen D-Spacing on Compact Bone Viscoelasticity: A Finite Element Analysis. Master's thesis, California Polytechnic State University, San Luis Obispo, 2013.

[55] Lisa M. Miller, Vidyasagar Vairavamurthy, Mark R. Chance, Richard Mendelsohn, Eleftherios P. Paschalis, Foster Betts, and Adele L. Boskey. In situ analysis of mineral content and crystallinity in bone using infrared micro-spectroscopy of the ν4 PO43- vibration. *Biochimica et Biophysica Acta - General Subjects*, 1527(1-2):11–19, 2001.

[56] Elise F Morgan and Tony M Keaveny. Dependence of yield strain of human trabecular bone on anatomic site. *Journal of Biomechanics*, 34(5):569–577, may 2001.

[57] Ralph Müller, H. Van Campenhout, B. Van Damme, G. Van Der Perre, J. Dequeker, T. Hildebrand, and P. Rüegsegger. Morphometric analysis of human bone biopsies: A quantitative structural comparison of histological sections and micro-computed tomography. *Bone*, 23(1):59–66, 1998.

[58] National Osteoporosis Foundation. Learn What Osteoporosis Is and What It's Caused by, 2019.

[59] A. M. Parfitt. Quantum concept of bone remodeling and turnover: Implications for the pathogenesis of osteoporosis. *Calcified Tissue International*, 28(1):1–5, dec 1979.

[60] John A. Petruska and Alan J. Hodge. A SUBUNIT MODEL FOR THE TROPOCOLLAGEN MACROMOLECULE. *Proceedings of the National Academy of Sciences*, 51(5), 1964.

[61] G. E. Piérard, T. Hermanns-Lê, P. Paquet, and C. Piérard-Franchimont. Skin viscoelasticity during hormone replacement therapy for climacteric ageing. *International Journal of Cosmetic Science*, 36(1):88–92, feb 2014.

[62] G. E. Piérard, C. Piérard-Franchimont, S. Vanderplaetsen, N. Franchimont, U. Gaspard, and M. Malaise. Relationship between bone mass density and tensile strength of the skin in women. *European Journal of Clinical Investigation*, 31(8):731–735, 2001.

[63] G.U. Raju, C.G. Rajeswari, R. Balannavar, and K.G. Kodancha. An investigation of fracture toughness and dynamic mechanical analysis of polymer nano-composites. *International Journal of Engineering, Science and Technology*, 10(2):30, 2018.

[64] J.C. Rice, S.C. Cowin, and J.A. Bowman. On the dependence of the elasticity and strength of cancellous bone on apparent density. *Journal of Biomechanics*, 21(2):155–168, jan 1988.

[65] Frank Richter. *Upsetting and Viscoelasticity of Vitreous SiO2: Experiments, Interpretation and Simulation*. PhD thesis, Technischen Universität, Berlin, 2006.

[66] R. Rizzoli, J. P. Bonjour, and S. L. Ferrari. Osteoporosis, genetics and hormones. *Journal of Molecular Endocrinology*, 26(2):79–94, 2001.

[67] Mitsuru Saito and Keishi Marumo. Effects of Collagen Crosslinking on Bone Material Properties in Health and Disease. *Calcified Tissue International*, 97(3):242–261, 2015.

[68] D.W. Saunders. *Creep and relaxation of nonlinear viscoelastic materials*, volume 19. Dover Publications, Inc., 1978.

[69] Thomas Siegmund, Matthew R. Allen, and David B. Burr. Failure of mineralized collagen fibrils: Modeling the role of collagen cross-linking. *Journal of Biomechanics*, 41(7):1427–1435, jan 2008.

[70] Thomas Siegmund, Matthew R. Allen, and David B. Burr. Failure of mineralized collagen fibrils: Modeling the role of collagen cross-linking. *Journal of Biomechanics*, 41(7):1427–1435, jan 2008.

[71] Tang S.Y. and D. Vashishth. The relative contributions of non-enzymatic glycation and cortical porosity on the fracture toughness of aging bone. *Journal of Biomechanics*, 44(2):330–336, jan 2011.

[72] S Y Tang, U Zeenath, and D Vashishth. Effects of non-enzymatic glycation on cancellous bone fragility. *Bone*, 40(4):1144–51, apr 2007.

[73] R. V. Thakker. *Genetics of bone biology and skeletal disease*. Academic Press, 2013.

[74] Luke Stanwood Thompson. The Effects of Hydroxyapatite and Tropocollagen's Material Constants on Compact Bone Viscoelasticity: A Finite Element Analysis. Master's thesis, California Polytechnic State University, San Luis Obispo, 2017.

[75] Joseph M. Wallace. Applications of atomic force microscopy for the assessment of nanoscale morphological and mechanical properties of bone. *Bone*, 50(1):420–427, 2012.

[76] Joseph M. Wallace. Applications of atomic force microscopy for the assessment of nanoscale morphological and mechanical properties of bone. *Bone*, 50(1):420–427, 2012.

[77] Joseph M. Wallace. Skeletal Hard Tissue Biomechanics. In *Basic and Applied Bone Biology*, pages 115–130. Elsevier, 2014.

[78] Joseph M. Wallace, Blake Erickson, Clifford M. Les, Bradford G. Orr, and Mark M. Banaszak Holl. Distribution of type I collagen morphologies in bone: Relation to estrogen depletion. *Bone*, 46(5):1349–1354, 2010.

[79] Joseph M. Wallace, Bradford G. Orr, Joan C. Marini, and Mark M Banaszak Holl. Nanoscale morphology of Type I collagen is altered in the Brtl mouse model of Osteogenesis Imperfecta. *Journal of Structural Biology*, 173(1):146–152, 2011.

[80] Xiaodu Wang, L. I. Xiaoe, Xinmei Shen, and C. Mauli Agrawal. Age-Related Changes of Noncalcified Collagen in Human Cortical Bone. *Annals of Biomedical Engineering*, 31(11):1365–1371, 2003.

[81] Marco P.E. Wenger, Laurent Bozec, Michael A. Horton, and Patrick Mesquidaz. Mechanical properties of collagen fibrils. *Biophysical Journal*, 93(4):1255–1263, 2007.

[82] Beth Winkelstein. *Orthopaedic Biomechanics*. CRC Press, dec 2012.

[83] Junro Yamashita, Benjamin R. Furman, H. Ralph Rawls, Xiaodu Wang, and C. Mauli Agrawal. The use of dynamic mechanical analysis to assess the

viscoelastic properties of human cortical bone. *Journal of Biomedical Materials Research*, 58(1):47–53, 2001.

[84] Ji Yean, Hisashi Naito, Takeshi Matsumoto, and Masao Tanak. Osteocyte Apoptosis-Induced Bone Resorption in Mechanical Remodeling Simulation - Computational Model for Trabecular Bone Structure. In *Apoptosis and Medicine*. InTech, aug 2012.

[85] M R Zarrinkalam, H Beard, C G Schultz, and R J Moore. Validation of the sheep as a large animal model for the study of vertebral osteoporosis. *European spine journal : official publication of the European Spine Society, the European Spinal Deformity Society, and the European Section of the Cervical Spine Research Society*, 18(2):244–53, feb 2009.

[86] P. Zioupos, J. D. Currey, A. Casinos, and V. De Buffrénil. Mechanical properties of the rostrum of the whale Mesoplodon densirostris, a remarkably dense bony tissue. *Journal of Zoology*, 241(4):725–737, 1997.

APPENDICES

Appendix A

EXPERIMENTAL PROTOCOL

A.1 Variable Dashpot Scheme

The following is for generating a 2x100 half-unit cell model, against which the VD scheme was tested, though the variable dashpot modified Richter UMAT can be used on the smaller model as well, or any model with tropocollagen custom user material with six variable inputs. This protocol has remained mostly unchanged since originally conceived by Cummings and Ha [16][35].

1. **Open Command Line Window:** Start → Intel 64 Visual Studio 2010 Model
2. **Set directory path:** Type "cd <directory_path_here>", then hit enter. Copy the path by selecting the desired directory in the Windows Explorer and copying the path in the address bar at the top of the window. When pasting into the command line, right-click, then select "Paste". When Abaqus is opened, the working directory will automatically be set to this directory.
3. **Open Abaqus CAE:** Type "abaqus cae" and hit enter. This will open the GUI for Abaqus CAE. Select "With Standard/Explicit Model" under Create Model Database
4. **Running Model Generation Script:** File → Run Script. Select desired Python script (.py). Currently this is "FINALPYTHONEDIT.py". A Model will be generated and a message display at in the Messages area of the GUI (at

the bottom of the window). The model can be used if "Model is now valid; Good for Analysis" is displayed. If "Reject model: Currently Biologically Invalid" is displayed, repeat step 4 until a biologically valid model is generated. Record the "LengthF = <number>", which will be used later. D-spacing distribution can be manually changed by editing the python script file. Editors used include WordPad, NotePad, and Notepad++ and Sublime.

5. **Completing the model:** The generated model is not completed. Due to the Gaussian distribution of d-spacings and random generation, the material properties for Collagen and Hydroxyapatite/mineral must be manually assigned. This is time consuming. The properties and sections have been created for each material already. Under the "Property" Module select "Assign Section", then select all regions for one of the two materials (Shift+click each one). Once all regions for the first material are selected, click "Done" at the bottom of the Viewport. This will bring up the Edit Section Assignment window. Under Section, select the chosen material and "Thickness Assignment: From Section", then hit OK. The selected regions will now turn turquoise. Select the regions associated with the other material and assign sections. Under the "Load" Module, create a new Boundary Condition called "YSYM" that is Mechanical and "Symmetry/Antisymmetry/Encastre" that is applied during the "Apply Load" step, then press "Continue...". Select the entire bottom edge of the model and click "Done". Next check the "YSYMM (U2=UR1=UR3=0)" option in the Edit Boundary Condition window, then select OK. Now create a new Load, still under the "Load" Module. Name it "PRESSURE" and select the "APPLY LOAD" step, check the mechanical option under property, and select Pressure for types of Selected Step, then click "Continue...". Select the entire right edge of the model in the viewport and click "Done". Leaving the Distribution as "Uniform", enter -3.36E-6 for Magnitude, then select "SINUSOIDAL"

from the Amplitude dropdown menu, and press OK. Now under the "Mesh" Module, select Seed Part, which brings up the "Global Seeds" window. Set "Approximate global size" to 0.0005. Accept the other default settings and click ok. Under the "Mesh" dropdown menu, select "Element Type...", select the entire model, and click "Done". This opens the "Element Type Window". Select the follow options: Element Library=Standard, Family=Plane Strain, Geometric Order=Quadratic, Quad=deselect both Hybrid formulation and Reduced Integration and accept the other default options. This should create a CPE8 (8-node biquadratic plan strain quadrilateral) element type. Click OK. A message window may come up, select OK again, the "Done". Select "Mesh Part" and "OK". The model set up has now been completed.

6. **Creating an input file:** Under the "Job" module, select "Create Job". Name the file according to treatment, location, version, frequency, modulus, and other specifying parameters. Select the just made model under "Source" dropdown menu, then "Continue...". The "Edit Job" window opens, accept all default settings, and click OK. Expand the "Job" tree under "Analysis" and right click on the job name, then select "Write Input". This will create an input file (.inp) in the current working directory. This input file can be edited using one of the previously mentioned editors. Due to the length of these files, using the Find and Replace command is very useful for making edits.

   Areas to update:

   (a) *Amplitude Line

   (b) *User Material

   (c) ** STEP: APPLY LOAD

   (d) Then add 400 steps

7. **Running the input file:** Open a new command line window (Start -¿ Intel 64 Visual Studio 2010 Model). Using the cd command, copy the path to the directory with the input file and hit enter. Also within that directory, place the "RichterUMATv2.f" file, which is the user-subroutine that defines how the collagen viscoelasticity will be calculated. Make any further changes to the input file now that are desired, then save the file. Once the input file is reading to be ran, type in the command "abaqus job=<input_file_name> user=RichterUMATv2.f cpus=8" then press enter. The cpus=8 command will allow all 8 processor units to be ran at a higher rate. If multiple jobs are desired to be ran one after the other, such as during the night, there is an easy way to do this, but I will add it in later. Bring up the Task Manager to see that the file is being compiled (an executive process should be running and taking up roughly 13% of the CPUs during pre-processing and up to 99% once the standard.exe process begins). This will take several hours to run, depending on the size of the file and the computer you are using (as of now it takes about 2.5-3 on FEA 3 and FEA 4, and around 5.5-6 on FEA 1). Within the directory, multiple files of different type should have been created, the important one being the Output Database (.odb) file, which will grow in memory to about 4.5 GB.

8. **Extracting the Displacement Data, Post processing, and MATLAB:** Open the ODB in Abaqus and check the nodes at the end of the model and make a node of their node number. Put these numbers into whichever postproccessing.py file is being used, and also make sure the file includes the path to your recently created .odb file. Open that in Abaqus in File - "Run Script", (can also from Abaqus Command cd to dir containing your .odb and type abaqus cae noGui=postprocessing_vTGM.py) It will make a folder with all the nodes when it is finished. Move that main folder somewhere safe (it will get replaced if not moved). Put the 3 matlab files into that folder. Update the main Matlab

file accordingly with the nodes. Play the file 10 times and record each Tangent Delta number in excel. Save the excel sheet.

A.2 IMF Scheme

The IMF approach differed most in the initial steps because a custom model needs to be made in dimensions reminiscent of Mendoza's original models, which of course are based on Siegmund's [54][69]. Read the all the steps before starting.

1. Make a model with specific dimensions and make sure the material assignments, seed length element type, boundary conditions and loads are all the same as the VD approach. Make sure that everything is a separate part or instance with in the assembly. Make sure the names match what the noderandomizer.py requires. Select all the nodes at the top and bottom of the surfaces of the parts and name them as a set in accordance with the noderandomizer.py script. Across from those node sets, make a surface to tie to eventually.

2. Create it as a job and choose the option to write to input.

3. Edit the noderandomizer.py script to make sure that it's opening the right input file, that the instance and node set names match.

4. Run the script in a terminal in the directory with the input file.

5. Check the input file. New node sets should have been created.

6. Open the model again back in Abaqus using the option to import from input file.

7. Set up general surface contact as seen in the Methods section. Use tie constraints to tie the newly created random crosslink node sets to the adjacent surface.

This protocol then dovetails into step 6 of the VD protocol with minor, intuitive adjustments to post proccessing.

Appendix B

EXPERIMENTAL OVINE DATA

Table B.1: C1809 Experimental DMA Data

Frequency (Hz)	Tan delta	Frequency (Hz)	Tan delta	Frequency (Hz)	Tan Delta	Frequency (Hz)	Tan delta
1	0.0706662	5.8	0.03774473	10.6	0.01882163	15.4	1.19E-05
1.2	0.06834705	6	0.03818411	10.8	0.01806933	15.6	1.18E-05
1.4	0.06677857	6.2	0.03771987	11	0.01637048	15.8	1.17E-05
1.6	0.06344343	6.4	0.03825383	11.2	0.01687733	16	1.16E-05
1.8	0.06292523	6.6	0.03436791	11.4	0.01506744	16.2	1.16E-05
2	0.06149181	6.8	0.03556467	11.6	0.01463229	16.4	1.15E-05
2.2	0.0606753	7	0.03427221	11.8	0.01346046	16.6	1.14E-05
2.4	0.06080037	7.2	0.03413199	12	0.01375842	16.8	1.13E-05
2.6	0.05781499	7.4	0.03326195	12.2	0.0118651	17	1.13E-05
2.8	0.05784416	7.6	0.03323054	12.4	0.01090571	17.2	1.12E-05
3	0.0561057	7.8	0.03256895	12.6	0.009315197	17.4	1.11E-05
3.2	0.04899304	8	0.03211624	12.8	0.009888422	17.6	1.11E-05
3.4	0.04779724	8.2	0.02990523	13	0.009356661	17.8	1.10E-05
3.6	0.04826368	8.4	0.02935176	13.2	0.009711676	18	1.09E-05
3.8	0.04603701	8.6	0.02774885	13.4	0.008760741	18.2	1.08E-05
4	0.04596044	8.8	0.02964686	13.6	0.007176988	18.4	1.08E-05
4.2	0.04423509	9	0.02773098	13.8	0.005908904	18.6	1.07E-05
4.4	0.04546543	9.2	0.02581526	14	0.004956043	18.8	1.06E-05
4.6	0.04416	9.4	0.02589232	14.2	0.004492137	19	1.05E-05
4.8	0.04285443	9.6	0.02245827	14.4	0.003436594	19.2	1.05E-05
5	0.04280069	9.8	0.02123512	14.6	0.002380318	19.4	1.04E-05
5.2	0.04060858	10	0.02048146	14.8	0.001196195	19.6	1.03E-05
5.4	0.04035063	10.2	0.01990271	15	0.001090865	19.8	1.02E-05
5.6	0.04050958	10.4	0.01960202	15.2	0.000659007	20	1.02E-05

Table B.2: C2207 Experimental DMA Data

Frequency (Hz)	Tan delta	Frequency (Hz)	Tan delta	Frequency (Hz)	Tan delta	Frequency (Hz)	Tan delta
1	0.07764606	5.8	0.05338047	10.6	0.03722326	15.4	0.02148215
1.2	0.07459385	6	0.05210621	10.8	0.03595282	15.6	0.02059606
1.4	0.073872	6.2	0.05076499	11	0.03576613	15.8	0.0194107
1.6	0.07016801	6.4	0.05108142	11.2	0.03498803	16	0.01652068
1.8	0.0714322	6.6	0.05188696	11.4	0.03362441	16.2	0.01445805
2	0.07140123	6.8	0.05121481	11.6	0.03391378	16.4	0.01471029
2.2	0.06824054	7	0.0483047	11.8	0.03243059	16.6	0.01454129
2.4	0.07110521	7.2	0.05045638	12	0.03248027	16.8	0.01389745
2.6	0.0692185	7.4	0.04872261	12.2	0.03221185	17	0.01230788
2.8	0.06727657	7.6	0.05003944	12.4	0.03068836	17.2	0.01235553
3	0.06824974	7.8	0.04866206	12.6	0.03019382	17.4	0.0107688
3.2	0.05765026	8	0.04857968	12.8	0.03113315	17.6	0.01119587
3.4	0.05771432	8.2	0.04661258	13	0.03083545	17.8	0.01008766
3.6	0.05752941	8.4	0.04582217	13.2	0.02876508	18	0.00956969
3.8	0.05760765	8.6	0.04481993	13.4	0.02878607	18.2	0.008104168
4	0.05632584	8.8	0.04344526	13.6	0.02872786	18.4	0.0068742
4.2	0.05621213	9	0.04348338	13.8	0.0274863	18.6	0.006983924
4.4	0.05563031	9.2	0.04234273	14	0.02648006	18.8	0.0055203
4.6	0.05574835	9.4	0.04254957	14.2	0.02688778	19	0.004292245
4.8	0.05493149	9.6	0.04023409	14.4	0.02623913	19.2	0.004663999
5	0.05458837	9.8	0.03933285	14.6	0.02488049	19.4	0.003869897
5.2	0.05344719	10	0.03890172	14.8	0.02365935	19.6	0.003714493
5.4	0.05460013	10.2	0.03854911	15	0.02313216	19.8	1.16922E-05
5.6	0.05395101	10.4	0.03753306	15.2	0.02189439	20	1.16236E-05

Appendix C

RUN DATA

Table C.1: IMF Run Data 1

Run Descriptor	Frequency (Hz)	Tangent δ												
		DMA	St Dev	Mean	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
Tie Constrained	1	7.656E-02	4.932E-03	**7.822E-02**	7.987E-02	7.988E-02	6.342E-02	7.986E-02	7.984E-02	7.987E-02	7.983E-02	7.989E-02	7.988E-02	7.987E-02
Tie Constrained	3	6.437E-02	4.441E-03	**3.002E-02**	2.850E-02	2.857E-02	2.855E-02	2.856E-02	2.851E-02	2.856E-02	2.856E-02	2.852E-02	4.335E-02	2.857E-02
Tie Constrained	9	3.947E-02	3.238E-05	**9.141E-03**	9.169E-03	9.169E-03	9.069E-03	9.141E-03	9.148E-03	9.135E-03	9.153E-03	9.182E-03	9.099E-03	9.147E-03
Tie Constrained	15	1.876E-02	2.661E-05	**4.967E-03**	5.004E-03	4.990E-03	4.996E-03	4.986E-03	4.922E-03	4.953E-03	4.953E-03	4.968E-03	4.972E-03	4.928E-03
AddedBC201	1	7.656E-02	1.444E-02	**1.217E-02**	4.912E-03	4.996E-03	4.986E-03	4.335E-02	4.968E-03	4.972E-03	3.858E-02	4.967E-03	4.967E-03	4.960E-03
AddedBC201	3	6.437E-02	1.263E-02	**5.573E-03**	1.383E-03	1.399E-03	1.399E-03	1.335E-03	4.345E-02	1.335E-03	1.378E-03	1.378E-03	1.335E-03	1.335E-03
AddedBC201	9	3.947E-02	1.294E-02	**4.643E-03**	3.599E-04	3.501E-04	3.434E-04	3.576E-04	3.274E-04	3.155E-04	2.759E-04	4.345E-02	3.448E-04	3.049E-04
AddedBC201	15	1.876E-02	2.477E-05	**1.794E-04**	2.052E-04	1.764E-04	1.816E-04	1.288E-04	2.051E-04	1.868E-04	1.425E-04	1.902E-04	1.723E-04	2.051E-04
AddedBC202	1	7.656E-02	3.025E-05	**3.322E-04**	3.501E-04	3.434E-04	3.576E-04	3.897E-04	3.274E-04	3.129E-04	3.155E-04	2.759E-04	3.448E-04	3.049E-04
AddedBC202	3	6.437E-02	2.285E-05	**3.323E-04**	3.559E-04	3.360E-04	3.367E-04	2.862E-04	3.553E-04	3.197E-04	3.427E-04	3.427E-04	2.967E-04	3.507E-04
AddedBC202	9	3.947E-02	1.970E-05	**3.374E-04**	3.160E-04	3.408E-04	3.449E-04	3.602E-04	3.248E-04	3.246E-04	3.002E-04	3.453E-04	3.679E-04	3.490E-04
AddedBC202	15	1.876E-02	2.828E-05	**3.084E-04**	2.779E-04	3.530E-04	3.169E-04	2.696E-04	3.367E-04	3.472E-04	2.901E-04	2.820E-04	3.142E-04	2.967E-04
AddedBC203	1	7.656E-02	1.525E-05	**5.514E-03**	5.499E-03	5.503E-03	5.486E-03	5.507E-03	5.517E-03	5.528E-03	5.543E-03	5.519E-03	5.519E-03	5.520E-03
AddedBC203	3	6.437E-02	2.197E-05	**1.314E-03**	1.308E-03	1.337E-03	1.309E-03	1.338E-03	1.328E-03	1.281E-03	1.344E-03	1.288E-03	1.322E-03	1.286E-03
AddedBC203	9	3.947E-02	1.895E-05	**2.915E-04**	3.150E-04	3.039E-04	3.192E-04	2.764E-04	2.994E-04	2.596E-04	2.824E-04	2.746E-04	3.072E-04	2.775E-04
AddedBC203	15	1.876E-02	2.469E-05	**1.790E-04**	1.955E-04	2.057E-04	1.964E-04	1.365E-04	1.516E-04	1.832E-04	1.997E-04	1.996E-04	1.796E-04	1.418E-04
AddedBC251	1	7.656E-02	3.433E-05	**5.869E-03**	5.890E-03	5.830E-03	5.842E-03	5.867E-03	5.906E-03	5.846E-03	5.859E-03	5.820E-03	5.929E-03	5.901E-03
AddedBC251	3	6.437E-02	2.511E-05	**1.387E-03**	1.361E-03	1.416E-03	1.431E-03	1.384E-03	1.367E-03	1.400E-03	1.357E-03	1.357E-03	1.403E-03	1.398E-03
AddedBC251	9	3.947E-02	2.079E-05	**3.258E-04**	2.924E-04	3.001E-04	3.156E-04	3.486E-04	3.479E-04	3.479E-04	3.363E-04	3.452E-04	3.179E-04	3.061E-04
AddedBC251	15	1.876E-02	2.804E-05	**1.801E-04**	1.946E-04	1.846E-04	1.336E-04	2.199E-04	1.806E-04	2.095E-04	1.818E-04	2.044E-04	1.525E-04	1.393E-04
AddedBC252	1	7.656E-02	2.440E-05	**6.179E-03**	6.202E-03	6.168E-03	6.168E-03	6.168E-03	6.223E-03	6.143E-03	6.206E-03	6.149E-03	6.188E-03	6.173E-03
AddedBC252	3	6.437E-02	2.806E-05	**1.480E-03**	1.480E-03	1.497E-03	1.437E-03	1.516E-03	1.434E-03	1.487E-03	1.484E-03	1.494E-03	1.517E-03	1.453E-03
AddedBC252	9	3.947E-02	1.924E-05	**3.356E-04**	3.641E-04	3.523E-04	3.670E-04	3.388E-04	3.339E-04	3.200E-04	3.367E-04	3.163E-04	3.105E-04	3.166E-04
AddedBC252	15	1.876E-02	2.934E-05	**1.805E-04**	1.863E-04	2.147E-04	1.381E-04	1.453E-04	2.106E-04	1.996E-04	1.996E-04	1.372E-04	2.066E-04	1.672E-04
AddedBC253	1	7.656E-02	2.400E-05	**5.833E-03**	5.859E-03	5.821E-03	5.806E-03	5.864E-03	5.855E-03	5.823E-03	5.795E-03	5.843E-03	5.855E-03	5.809E-03
AddedBC253	3	6.437E-02	3.282E-05	**1.391E-03**	1.360E-03	1.332E-03	1.418E-03	1.420E-03	1.432E-03	1.407E-03	1.348E-03	1.413E-03	1.374E-03	1.404E-03
AddedBC253	9	3.947E-02	2.079E-05	**3.051E-04**	2.760E-04	2.950E-04	3.375E-04	3.031E-04	2.842E-04	2.994E-04	2.955E-04	3.019E-04	3.121E-04	3.463E-04
AddedBC253	15	1.876E-02	3.087E-05	**1.768E-04**	1.628E-04	2.181E-04	1.533E-04	1.474E-04	1.421E-04	1.971E-04	1.466E-04	1.670E-04	2.030E-04	2.307E-04

Table C.2: IMF Run Data 2

Run Descriptor	Frequency (Hz)	Tangent δ												
		DMA	St Dev	Mean	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
AddedBC301	1	7.656E-02	4.245E-05	**5.573E-03**	5.658E-03	5.588E-03	5.583E-03	5.609E-03	5.531E-03	5.540E-03	5.532E-03	5.619E-03	5.530E-03	5.545E-03
AddedBC301	3	6.437E-02	2.801E-05	**1.328E-03**	1.308E-03	1.331E-03	1.275E-03	1.354E-03	1.335E-03	1.366E-03	1.340E-03	1.361E-03	1.302E-03	1.305E-03
AddedBC301	9	3.947E-02	2.764E-05	**3.147E-04**	2.632E-04	2.820E-04	3.495E-04	3.290E-04	3.001E-04	3.587E-04	3.203E-04	3.203E-04	3.003E-04	3.237E-04
AddedBC301	15	1.876E-02	1.903E-05	**1.748E-04**	1.794E-04	1.696E-04	1.580E-04	2.006E-04	1.606E-04	1.958E-04	1.980E-04	1.588E-04	1.420E-04	1.852E-04
AddedBC302	1	7.656E-02	2.744E-05	**5.272E-03**	5.289E-03	5.296E-03	5.228E-03	5.264E-03	5.296E-03	5.304E-03	5.242E-03	5.298E-03	5.234E-03	5.270E-03
AddedBC302	3	6.437E-02	2.946E-05	**1.256E-03**	1.259E-03	1.267E-03	1.272E-03	1.238E-03	1.307E-03	1.272E-03	1.267E-03	1.254E-03	1.239E-03	1.188E-03
AddedBC302	9	3.947E-02	3.451E-05	**2.816E-04**	3.249E-04	3.113E-04	2.734E-04	2.883E-04	1.977E-04	3.092E-04	2.686E-04	3.068E-04	2.631E-04	2.723E-04
AddedBC302	15	1.876E-02	3.027E-05	**1.424E-04**	1.495E-04	1.886E-04	9.751E-05	1.326E-04	1.292E-04	1.384E-04	1.505E-04	1.149E-04	2.009E-04	1.217E-04
AddedBC303	1	7.656E-02	1.660E-02	**1.086E-02**	5.347E-03	5.357E-03	5.287E-03	5.316E-03	5.328E-03	5.290E-03	5.351E-03	5.341E-03	6.066E-02	5.275E-03
AddedBC303	3	6.437E-02	3.154E-05	**1.270E-03**	1.319E-03	1.223E-03	1.263E-03	1.265E-03	1.292E-03	1.294E-03	1.254E-03	1.308E-03	1.256E-03	1.222E-03
AddedBC303	9	3.947E-02	3.324E-05	**2.976E-04**	2.649E-04	3.331E-04	2.539E-04	3.596E-04	2.823E-04	3.093E-04	2.957E-04	3.155E-04	3.099E-04	2.516E-04
AddedBC303	15	1.876E-02	2.644E-05	**1.561E-04**	1.421E-04	1.952E-04	1.456E-04	1.919E-04	1.725E-04	1.388E-04	1.766E-04	1.044E-04	1.417E-04	1.524E-04
AddedBC351	1	7.656E-02	2.016E-02	**1.356E-02**	5.446E-03	5.505E-03	5.459E-03	5.508E-03	5.510E-03	5.520E-03	7.288E-02	5.509E-03	1.878E-02	5.511E-03
AddedBC351	3	6.437E-02	3.291E-05	**1.299E-03**	1.283E-03	1.276E-03	1.314E-03	1.283E-03	1.332E-03	1.236E-03	1.333E-03	1.336E-03	1.328E-03	1.264E-03
AddedBC351	9	3.947E-02	3.281E-05	**2.998E-04**	2.661E-04	3.383E-04	3.373E-04	2.882E-04	3.287E-04	2.387E-04	3.172E-04	3.271E-04	2.731E-04	2.829E-04
AddedBC351	15	1.876E-02	3.603E-05	**1.699E-04**	1.535E-04	1.678E-04	2.064E-04	9.644E-05	1.799E-04	2.007E-04	1.296E-04	2.267E-04	1.754E-04	1.630E-04
AddedBC352	1	7.656E-02	2.438E-05	**5.350E-03**	5.304E-03	5.341E-03	5.333E-03	5.349E-03	5.354E-03	5.384E-03	5.373E-03	5.320E-03	5.376E-03	5.364E-03
AddedBC352	3	6.437E-02	2.202E-05	**1.280E-03**	1.279E-03	1.300E-03	1.272E-03	1.243E-03	1.299E-03	1.248E-03	1.305E-03	1.260E-03	1.300E-03	1.295E-03
AddedBC352	9	3.947E-02	2.496E-05	**3.010E-04**	3.604E-04	2.920E-04	3.258E-04	3.016E-04	3.044E-04	3.038E-04	2.823E-04	2.840E-04	2.664E-04	2.888E-04
AddedBC352	15	1.876E-02	2.747E-05	**1.698E-04**	2.196E-04	1.934E-04	1.493E-04	1.326E-04	1.620E-04	1.561E-04	1.716E-04	2.110E-04	1.462E-04	1.557E-04
AddedBC353	1	7.656E-02	3.010E-05	**5.384E-03**	5.371E-03	5.334E-03	5.392E-03	5.370E-03	5.353E-03	5.405E-03	5.416E-03	5.356E-03	5.428E-03	5.415E-03
AddedBC353	3	6.437E-02	1.277E-02	**5.527E-03**	1.248E-03	1.313E-03	1.296E-03	1.279E-03	1.255E-03	1.240E-03	1.255E-03	4.383E-02	1.278E-03	1.278E-03
AddedBC353	9	3.947E-02	3.376E-05	**2.930E-04**	2.618E-04	3.316E-04	2.546E-04	3.367E-04	3.304E-04	2.860E-04	2.567E-04	2.756E-04	2.635E-04	3.331E-04
AddedBC353	15	1.876E-02	3.774E-05	**1.745E-04**	1.416E-04	1.508E-04	1.689E-04	1.869E-04	1.999E-04	2.671E-04	1.421E-04	1.948E-04	1.378E-04	1.548E-04

Table C.3: C2207 Dashpot Run Data

Run Descriptor	Frequency (Hz)	Tangent δ												
		DMA	St Dev	Mean	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
0.3125 GPa-s	2	7.140E-02	2.154E-06	**4.371E-02**	4.371E-02	4.371E-02	4.371E-02	4.371E-02	4.371E-02	4.372E-02	4.371E-02	4.372E-02	4.371E-02	4.371E-02
0.3125 GPa-s	5	5.459E-02	3.305E-06	**5.167E-02**	5.167E-02	5.167E-02	5.167E-02	5.167E-02	5.167E-02	5.167E-02	5.167E-02	5.167E-02	5.166E-02	5.167E-02
0.3125 GPa-s	7	4.830E-02	2.830E-06	**4.545E-02**	4.545E-02	4.545E-02	4.545E-02	4.545E-02	4.545E-02	4.545E-02	4.545E-02	4.545E-02	4.545E-02	4.545E-02
0.3125 GPa-s	12	3.248E-02	2.948E-06	**3.160E-02**	3.161E-02	3.160E-02	3.160E-02	3.160E-02	3.161E-02	3.160E-02	3.160E-02	3.160E-02	3.161E-02	3.160E-02	3.160E-02
0.450 GPa-s	1	7.765E-02	2.448E-06	**3.521E-02**	3.521E-02	3.521E-02	3.521E-02	3.521E-02	3.521E-02	3.521E-02	3.521E-02	3.521E-02	3.521E-02	3.521E-02	3.521E-02
0.450 GPa-s	2	7.140E-02	2.740E-06	**5.107E-02**	5.107E-02	5.107E-02	5.107E-02	5.107E-02	5.107E-02	5.107E-02	5.108E-02	5.107E-02	5.107E-02	5.107E-02
0.450 GPa-s	3	6.825E-02	1.703E-06	**5.306E-02**	5.306E-02	5.306E-02	5.306E-02	5.306E-02	5.306E-02	5.306E-02	5.306E-02	5.306E-02	5.306E-02	5.306E-02
0.450 GPa-s	5	5.459E-02	3.021E-06	**4.478E-02**	4.478E-02	4.478E-02	4.478E-02	4.478E-02	4.478E-02	4.478E-02	4.478E-02	4.478E-02	4.478E-02	4.479E-02
0.450 GPa-s	7	4.830E-02	2.447E-06	**3.613E-02**	3.613E-02	3.613E-02	3.613E-02	3.613E-02	3.613E-02	3.613E-02	3.613E-02	3.613E-02	3.613E-02	3.613E-02
0.450 GPa-s	9	4.348E-02	4.549E-04	**2.962E-02**	2.977E-02	2.977E-02	2.977E-02	2.977E-02	2.976E-02	2.825E-02	2.977E-02	2.977E-02	2.977E-02	2.977E-02
0.450 GPa-s	12	3.248E-02	2.494E-06	**2.313E-02**	2.314E-02	2.313E-02	2.313E-02	2.313E-02	2.314E-02	2.313E-02	2.313E-02	2.313E-02	2.313E-02	2.314E-02
0.450 GPa-s	15	2.313E-02	3.363E-06	**1.893E-02**	1.893E-02	1.893E-02	1.894E-02	1.893E-02	1.893E-02	1.893E-02	1.893E-02	1.893E-02	1.893E-02	1.893E-02
0.5875 GPa-s	1	7.765E-02	3.093E-06	**4.217E-02**	4.216E-02	4.217E-02	4.217E-02	4.217E-02	4.216E-02	4.216E-02	4.216E-02	4.217E-02	4.217E-02	4.216E-02
0.5875 GPa-s	2	7.140E-02	3.048E-06	**5.324E-02**	5.323E-02	5.324E-02	5.324E-02	5.323E-02	5.323E-02	5.324E-02	5.324E-02	5.324E-02	5.324E-02	5.323E-02
0.5875 GPa-s	3	6.825E-02	2.716E-06	**4.997E-02**	4.997E-02	4.997E-02	4.998E-02	4.998E-02	4.998E-02	4.998E-02	4.997E-02	4.997E-02	4.997E-02	4.997E-02
0.5875 GPa-s	5	5.459E-02	3.498E-06	**3.797E-02**	3.798E-02	3.797E-02	3.797E-02	3.797E-02	3.797E-02	3.797E-02	3.797E-02	3.797E-02	3.796E-02	3.798E-02
0.5875 GPa-s	7	4.830E-02	2.369E-06	**2.932E-02**	2.932E-02	2.931E-02	2.932E-02	2.931E-02	2.932E-02	2.932E-02	2.932E-02	2.932E-02	2.932E-02	2.932E-02
0.5875 GPa-s	9	4.348E-02	2.597E-06	**2.366E-02**	2.366E-02	2.366E-02	2.366E-02	2.366E-02	2.366E-02	2.365E-02	2.366E-02	2.366E-02	2.366E-02	2.366E-02
0.5875 GPa-s	12	3.248E-02	2.906E-06	**1.809E-02**	1.809E-02	1.809E-02	1.809E-02	1.809E-02	1.809E-02	1.809E-02	1.808E-02	1.809E-02	1.809E-02	1.809E-02
0.5875 GPa-s	15	2.313E-02	2.088E-06	**1.471E-02**	1.471E-02	1.470E-02	1.471E-02	1.471E-02	1.471E-02	1.471E-02	1.470E-02	1.471E-02	1.471E-02	1.471E-02
0.725 GPa-s	1	7.765E-02	2.425E-06	**4.715E-02**	4.716E-02	4.715E-02	4.715E-02	4.716E-02	4.715E-02	4.715E-02	4.716E-02	4.715E-02	4.716E-02	4.716E-02
0.725 GPa-s	2	7.140E-02	3.782E-06	**5.246E-02**	5.247E-02	5.246E-02	5.246E-02	5.246E-02	5.246E-02	5.246E-02	5.246E-02	5.246E-02	5.246E-02	5.246E-02
0.725 GPa-s	3	6.825E-02	3.781E-06	**4.567E-02**	4.567E-02	4.566E-02	4.567E-02	4.567E-02	4.567E-02	4.567E-02	4.567E-02	4.567E-02	4.567E-02	4.567E-02
0.725 GPa-s	5	5.459E-02	2.433E-06	**3.247E-02**	3.248E-02	3.247E-02	3.247E-02	3.247E-02	3.247E-02	3.248E-02	3.247E-02	3.247E-02	3.247E-02	3.247E-02
0.725 GPa-s	7	4.830E-02	3.249E-06	**2.446E-02**	2.446E-02	2.446E-02	2.446E-02	2.446E-02	2.446E-02	2.447E-02	2.446E-02	2.447E-02	2.447E-02	2.447E-02
0.725 GPa-s	9	4.348E-02	3.040E-06	**1.953E-02**	1.953E-02	1.953E-02	1.953E-02	1.953E-02	1.953E-02	1.954E-02	1.953E-02	1.953E-02	1.953E-02	1.953E-02
0.725 GPa-s	12	3.248E-02	3.038E-06	**1.480E-02**	1.480E-02	1.480E-02	1.480E-02	1.480E-02	1.480E-02	1.480E-02	1.480E-02	1.481E-02	1.480E-02	1.480E-02
0.725 GPa-s	15	2.313E-02	2.919E-06	**1.200E-02**	1.200E-02	1.200E-02	1.200E-02	1.200E-02	1.200E-02	1.200E-02	1.200E-02	1.200E-02	1.200E-02	1.200E-02
0.8625 GPa-s	1	7.765E-02	2.659E-06	**5.043E-02**	5.043E-02	5.043E-02	5.043E-02	5.043E-02	5.043E-02	5.043E-02	5.043E-02	5.043E-02	5.043E-02	5.043E-02
0.8625 GPa-s	2	7.140E-02	3.241E-06	**5.025E-02**	5.025E-02	5.025E-02	5.024E-02	5.025E-02	5.025E-02	5.025E-02	5.024E-02	5.025E-02	5.025E-02	5.025E-02
0.8625 GPa-s	3	6.825E-02	2.590E-06	**4.138E-02**	4.138E-02	4.138E-02	4.139E-02	4.138E-02	4.138E-02	4.138E-02	4.138E-02	4.138E-02	4.139E-02	4.138E-02
0.8625 GPa-s	5	5.459E-02	2.799E-06	**2.817E-02**	2.817E-02	2.817E-02	2.817E-02	2.817E-02	2.817E-02	2.817E-02	2.817E-02	2.817E-02	2.817E-02	2.818E-02
0.8625 GPa-s	7	4.830E-02	2.170E-06	**2.091E-02**	2.091E-02	2.091E-02	2.091E-02	2.091E-02	2.091E-02	2.091E-02	2.091E-02	2.091E-02	2.090E-02	2.091E-02
0.8625 GPa-s	9	4.348E-02	2.455E-06	**1.660E-02**	1.660E-02	1.659E-02	1.659E-02	1.660E-02	1.660E-02	1.659E-02	1.659E-02	1.660E-02	1.660E-02	1.659E-02
0.8625 GPa-s	12	3.248E-02	3.129E-06	**1.250E-02**	1.250E-02	1.251E-02	1.250E-02	1.251E-02	1.250E-02	1.250E-02	1.251E-02	1.250E-02	1.251E-02	1.250E-02
0.8625 GPa-s	15	2.313E-02	2.481E-06	**1.013E-02**	1.013E-02	1.012E-02	1.013E-02	1.012E-02	1.013E-02	1.013E-02	1.013E-02	1.012E-02	1.013E-02	1.013E-02
1.25 GPa-s	2	7.140E-02	2.110E-06	**4.217E-02**	4.217E-02	4.217E-02	4.217E-02	4.218E-02	4.217E-02	4.217E-02	4.218E-02	4.217E-02	4.217E-02	4.218E-02
1.25 GPa-s	5	5.459E-02	3.178E-06	**2.025E-02**	2.025E-02	2.025E-02	2.025E-02	2.025E-02	2.025E-02	2.025E-02	2.026E-02	2.025E-02	2.025E-02	2.025E-02
1.25 GPa-s	7	4.830E-02	2.797E-06	**1.473E-02**	1.473E-02	1.473E-02	1.473E-02	1.473E-02	1.473E-02	1.474E-02	1.473E-02	1.473E-02	1.473E-02	1.473E-02
1.25 GPa-s	12	3.248E-02	3.696E-06	**8.665E-03**	8.661E-03	8.670E-03	8.658E-03	8.662E-03	8.669E-03	8.662E-03	8.664E-03	8.668E-03	8.662E-03	8.668E-03

Table C.4: C1809 Dashpot Run Data

Run Descriptor	Frequency (Hz)	Tangent δ												
		DMA	St Dev	Mean	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
0.0125 GPa-s	2	6.149E-02	1.891E-06	**2.235E-03**	2.236E-03	2.234E-03	2.235E-03	2.236E-03	2.237E-03	2.233E-03	2.235E-03	2.232E-03	2.236E-03	2.239E-03
0.0125 GPa-s	5	4.280E-02	3.206E-06	**5.675E-03**	5.679E-03	5.673E-03	5.669E-03	5.678E-03	5.677E-03	5.672E-03	5.675E-03	5.673E-03	5.674E-03	5.678E-03
0.0125 GPa-s	7	3.427E-02	2.759E-06	**7.952E-03**	7.949E-03	7.956E-03	7.955E-03	7.950E-03	7.952E-03	7.947E-03	7.955E-03	7.953E-03	7.952E-03	7.950E-03
0.0125 GPa-s	12	1.376E-02	2.051E-06	**1.352E-02**	1.352E-02	1.352E-02	1.352E-02	1.352E-02	1.351E-02	1.352E-02	1.352E-02	1.352E-02	1.352E-02	1.352E-02
0.125 GPa-s	2	6.149E-02	4.106E-06	**2.193E-02**	2.193E-02	2.193E-02	2.193E-02	2.194E-02	2.193E-02	2.193E-02	2.193E-02	2.194E-02	2.194E-02	2.194E-02
0.125 GPa-s	5	4.280E-02	2.680E-06	**4.455E-02**	4.455E-02	4.456E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02
0.125 GPa-s	7	3.427E-02	2.549E-06	**5.154E-02**	5.155E-02	5.155E-02	5.154E-02	5.154E-02	5.155E-02	5.154E-02	5.155E-02	5.154E-02	5.155E-02	5.155E-02
0.125 GPa-s	12	1.376E-02	4.230E-06	**5.288E-02**	5.288E-02	5.289E-02	5.289E-02	5.289E-02	5.288E-02	5.289E-02	5.288E-02	5.288E-02	5.289E-02	5.288E-02
0.3125 GPa-s	2	6.149E-02	2.304E-06	**4.455E-02**	4.456E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02	4.455E-02
0.3125 GPa-s	5	4.280E-02	2.860E-06	**5.241E-02**	5.241E-02	5.241E-02	5.241E-02	5.241E-02	5.241E-02	5.241E-02	5.241E-02	5.241E-02	5.241E-02	5.241E-02
0.3125 GPa-s	7	3.427E-02	2.978E-06	**4.604E-02**	4.603E-02	4.604E-02	4.603E-02	4.604E-02	4.604E-02	4.603E-02	4.604E-02	4.604E-02	4.604E-02	4.604E-02
0.3125 GPa-s	12	1.376E-02	3.050E-06	**3.197E-02**	3.197E-02	3.197E-02	3.197E-02	3.197E-02	3.197E-02	3.197E-02	3.197E-02	3.196E-02	3.197E-02	3.197E-02
0.450 GPa-s	1	7.067E-02	2.566E-06	**3.593E-02**	3.593E-02	3.593E-02	3.593E-02	3.593E-02	3.593E-02	3.593E-02	3.593E-02	3.592E-02	3.592E-02	3.593E-02
0.450 GPa-s	2	6.149E-02	2.896E-06	**5.196E-02**	5.195E-02	5.196E-02	5.195E-02	5.196E-02	5.196E-02	5.196E-02	5.196E-02	5.196E-02	5.196E-02	5.196E-02
0.450 GPa-s	3	5.611E-02	3.038E-06	**5.386E-02**	5.386E-02	5.386E-02	5.386E-02	5.386E-02	5.386E-02	5.385E-02	5.385E-02	5.386E-02	5.386E-02	5.386E-02
0.450 GPa-s	5	4.280E-02	3.072E-06	**4.536E-02**	4.536E-02	4.536E-02	4.536E-02	4.536E-02	4.535E-02	4.535E-02	4.536E-02	4.535E-02	4.536E-02	4.535E-02
0.450 GPa-s	7	3.427E-02	2.680E-06	**3.656E-02**	3.656E-02	3.656E-02	3.656E-02	3.656E-02	3.656E-02	3.656E-02	3.656E-02	3.656E-02	3.656E-02	3.656E-02
0.450 GPa-s	9	2.773E-02	2.292E-06	**3.011E-02**	3.010E-02	3.011E-02	3.011E-02	3.011E-02	3.011E-02	3.011E-02	3.011E-02	3.011E-02	3.011E-02	3.011E-02
0.450 GPa-s	12	1.376E-02	2.575E-06	**2.339E-02**	2.339E-02	2.340E-02	2.339E-02	2.340E-02	2.339E-02	2.340E-02	2.339E-02	2.339E-02	2.340E-02	2.340E-02
0.450 GPa-s	15	1.091E-03	2.734E-06	**1.914E-02**	1.914E-02	1.914E-02	1.914E-02	1.914E-02	1.914E-02	1.914E-02	1.914E-02	1.914E-02	1.914E-02	1.914E-02
0.5875 GPa-s	1	7.067E-02	2.134E-06	**4.299E-02**	4.299E-02	4.299E-02	4.299E-02	4.298E-02	4.298E-02	4.299E-02	4.298E-02	4.298E-02	4.299E-02	4.298E-02
0.5875 GPa-s	2	6.149E-02	2.662E-06	**5.408E-02**	5.408E-02	5.407E-02	5.408E-02	5.408E-02	5.408E-02	5.408E-02	5.408E-02	5.408E-02	5.408E-02	5.408E-02
0.5875 GPa-s	3	5.611E-02	2.323E-06	**5.066E-02**	5.066E-02	5.066E-02	5.066E-02	5.066E-02	5.066E-02	5.066E-02	5.066E-02	5.066E-02	5.066E-02	5.066E-02
0.5875 GPa-s	5	4.280E-02	2.955E-06	**3.843E-02**	3.843E-02	3.843E-02	3.842E-02	3.843E-02	3.843E-02	3.843E-02	3.843E-02	3.843E-02	3.843E-02	3.843E-02
0.5875 GPa-s	7	3.427E-02	2.217E-06	**2.966E-02**	2.966E-02	2.966E-02	2.965E-02	2.966E-02	2.966E-02	2.966E-02	2.965E-02	2.965E-02	2.965E-02	2.966E-02
0.5875 GPa-s	9	2.773E-02	2.304E-06	**2.392E-02**	2.393E-02	2.393E-02	2.392E-02	2.393E-02	2.393E-02	2.392E-02	2.392E-02	2.392E-02	2.392E-02	2.392E-02
0.5875 GPa-s	12	1.376E-02	3.549E-06	**1.829E-02**	1.829E-02	1.829E-02	1.829E-02	1.828E-02	1.829E-02	1.829E-02	1.828E-02	1.829E-02	1.829E-02	1.829E-02
0.5875 GPa-s	15	1.091E-03	2.693E-06	**1.487E-02**	1.487E-02	1.486E-02	1.487E-02	1.487E-02	1.487E-02	1.486E-02	1.487E-02	1.487E-02	1.486E-02	1.487E-02
0.725 GPa-s	1	7.067E-02	4.011E-06	**4.802E-02**	4.802E-02	4.803E-02	4.802E-02	4.802E-02	4.802E-02	4.802E-02	4.802E-02	4.803E-02	4.803E-02	4.802E-02
0.725 GPa-s	2	6.149E-02	2.309E-06	**5.323E-02**	5.323E-02	5.324E-02	5.323E-02	5.323E-02	5.324E-02	5.324E-02	5.323E-02	5.323E-02	5.324E-02	5.323E-02
0.725 GPa-s	3	5.611E-02	2.652E-06	**4.626E-02**	4.626E-02	4.626E-02	4.626E-02	4.626E-02	4.626E-02	4.626E-02	4.626E-02	4.626E-02	4.626E-02	4.626E-02
0.725 GPa-s	5	4.280E-02	2.237E-06	**3.285E-02**	3.285E-02	3.285E-02	3.285E-02	3.285E-02	3.285E-02	3.285E-02	3.286E-02	3.286E-02	3.285E-02	3.285E-02
0.725 GPa-s	7	3.427E-02	2.411E-06	**2.474E-02**	2.474E-02	2.474E-02	2.474E-02	2.474E-02	2.474E-02	2.474E-02	2.474E-02	2.474E-02	2.473E-02	2.474E-02
0.725 GPa-s	9	2.773E-02	2.990E-06	**1.975E-02**	1.975E-02	1.975E-02	1.974E-02	1.975E-02	1.975E-02	1.975E-02	1.974E-02	1.975E-02	1.975E-02	1.975E-02
0.725 GPa-s	12	1.376E-02	2.782E-06	**1.496E-02**	1.497E-02	1.496E-02	1.497E-02	1.496E-02	1.497E-02	1.496E-02	1.496E-02	1.496E-02	1.496E-02	1.497E-02
0.725 GPa-s	15	1.091E-03	2.627E-06	**1.213E-02**	1.213E-02	1.213E-02	1.213E-02	1.213E-02	1.213E-02	1.213E-02	1.213E-02	1.213E-02	1.213E-02	1.213E-02
0.8625 GPa-s	1	7.067E-02	2.629E-06	**5.132E-02**	5.132E-02	5.131E-02	5.132E-02	5.132E-02	5.132E-02	5.131E-02	5.132E-02	5.131E-02	5.131E-02	5.132E-02
0.8625 GPa-s	2	6.149E-02	1.961E-06	**5.094E-02**	5.094E-02	5.094E-02	5.095E-02	5.094E-02	5.094E-02	5.095E-02	5.094E-02	5.094E-02	5.094E-02	5.094E-02
0.8625 GPa-s	3	5.611E-02	1.862E-06	**4.189E-02**	4.189E-02	4.189E-02	4.189E-02	4.189E-02	4.189E-02	4.189E-02	4.189E-02	4.189E-02	4.189E-02	4.189E-02
0.8625 GPa-s	5	4.280E-02	3.128E-06	**2.849E-02**	2.849E-02	2.849E-02	2.850E-02	2.849E-02	2.850E-02	2.849E-02	2.849E-02	2.849E-02	2.849E-02	2.850E-02
0.8625 GPa-s	7	3.427E-02	2.913E-06	**2.114E-02**	2.114E-02	2.114E-02	2.114E-02	2.114E-02	2.114E-02	2.115E-02	2.115E-02	2.114E-02	2.114E-02	2.114E-02
0.8625 GPa-s	9	2.773E-02	2.442E-06	**1.678E-02**	1.677E-02	1.678E-02	1.677E-02	1.678E-02	1.677E-02	1.678E-02	1.678E-02	1.678E-02	1.678E-02	1.678E-02
0.8625 GPa-s	12	1.376E-02	2.400E-06	**1.264E-02**	1.264E-02	1.264E-02	1.264E-02	1.264E-02	1.264E-02	1.264E-02	1.264E-02	1.264E-02	1.264E-02	1.264E-02
0.8625 GPa-s	15	1.091E-03	2.243E-06	**1.024E-02**	1.023E-02	1.024E-02	1.024E-02	1.024E-02	1.024E-02	1.023E-02	1.023E-02	1.024E-02	1.023E-02	1.023E-02
1.25 GPa-s	2	6.149E-02	3.857E-06	**4.270E-02**	4.270E-02	4.270E-02	4.270E-02	4.270E-02	4.270E-02	4.270E-02	4.271E-02	4.270E-02	4.270E-02	4.269E-02
1.25 GPa-s	5	4.280E-02	3.088E-06	**2.048E-02**	2.048E-02	2.047E-02	2.048E-02	2.048E-02	2.047E-02	2.048E-02	2.048E-02	2.048E-02	2.047E-02	2.047E-02
1.25 GPa-s	7	3.427E-02	2.481E-06	**1.489E-02**	1.489E-02	1.489E-02	1.489E-02	1.489E-02	1.489E-02	1.489E-02	1.489E-02	1.489E-02	1.489E-02	1.489E-02
1.25 GPa-s	12	1.376E-02	2.003E-06	**8.759E-03**	8.760E-03	8.759E-03	8.755E-03	8.759E-03	8.759E-03	8.760E-03	8.756E-03	8.761E-03	8.757E-03	8.761E-03

Table C.5: Variable Dashpot Run Data

Run Descriptor	Frequency (Hz)	Tangent δ												
		DMA	St Dev	Mean	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
C1809 VD	1	7.067E-02	2.090E-06	**5.409E-02**	5.409E-02	5.409E-02	5.409E-02	5.409E-02	5.409E-02	5.409E-02	5.409E-02	5.409E-02	5.409E-02	5.409E-02
C1809 VD	2	6.149E-02	2.089E-06	**5.376E-02**	5.376E-02	5.375E-02	5.376E-02	5.375E-02	5.376E-02	5.375E-02	5.376E-02	5.376E-02	5.376E-02	5.375E-02
C1809 VD	3	5.611E-02	2.093E-06	**5.344E-02**	5.344E-02	5.344E-02	5.344E-02	5.344E-02	5.344E-02	5.345E-02	5.344E-02	5.344E-02	5.344E-02	5.344E-02
C1809 VD	5	4.280E-02	7.825E-05	**5.328E-02**	5.325E-02	5.344E-02	5.324E-02	5.344E-02	5.325E-02	5.325E-02	5.324E-02	5.324E-02	5.325E-02	5.324E-02
C1809 VD	7	3.427E-02	2.008E-06	**3.947E-02**	3.947E-02	3.947E-02	3.947E-02	3.947E-02	3.948E-02	3.948E-02	3.947E-02	3.947E-02	3.947E-02	3.948E-02
C1809 VD	9	2.773E-02	2.736E-06	**2.926E-02**	2.927E-02	2.926E-02	2.927E-02	2.926E-02	2.926E-02	2.927E-02	2.926E-02	2.926E-02	2.926E-02	2.926E-02
C1809 VD	12	1.376E-02	2.238E-06	**1.149E-02**	1.149E-02	1.149E-02	1.149E-02	1.149E-02	1.149E-02	1.149E-02	1.149E-02	1.149E-02	1.149E-02	1.149E-02
C1809 VD	15	1.091E-03	3.056E-06	**7.086E-03**	7.085E-03	7.087E-03	7.091E-03	7.083E-03	7.090E-03	7.082E-03	7.084E-03	7.085E-03	7.083E-03	7.090E-03
C2207 VD	1	7.765E-02	2.942E-06	**5.326E-02**	5.326E-02	5.326E-02	5.325E-02	5.326E-02	5.326E-02	5.326E-02	5.326E-02	5.326E-02	5.326E-02	5.326E-02
C2207 VD	2	7.140E-02	4.606E-06	**5.315E-02**	5.314E-02	5.315E-02	5.314E-02	5.314E-02	5.315E-02	5.315E-02	5.314E-02	5.314E-02	5.314E-02	5.315E-02
C2207 VD	3	6.825E-02	3.003E-06	**5.333E-02**	5.333E-02	5.333E-02	5.333E-02	5.333E-02	5.334E-02	5.333E-02	5.333E-02	5.333E-02	5.333E-02	5.334E-02
C2207 VD	5	5.459E-02	2.675E-06	**5.148E-02**	5.148E-02	5.149E-02	5.149E-02	5.148E-02	5.149E-02	5.148E-02	5.149E-02	5.149E-02	5.149E-02	5.148E-02
C2207 VD	7	4.830E-02	2.228E-06	**4.368E-02**	4.368E-02	4.369E-02	4.368E-02	4.369E-02	4.369E-02	4.368E-02	4.368E-02	4.368E-02	4.369E-02	4.368E-02
C2207 VD	9	4.348E-02	2.996E-06	**4.094E-02**	4.095E-02	4.094E-02	4.094E-02	4.095E-02	4.095E-02	4.095E-02	4.094E-02	4.095E-02	4.094E-02	4.094E-02
C2207 VD	12	3.248E-02	3.000E-06	**3.116E-02**	3.117E-02	3.116E-02	3.116E-02	3.116E-02	3.116E-02	3.116E-02	3.115E-02	3.116E-02	3.116E-02	3.116E-02
C2207 VD	15	2.313E-02	2.280E-06	**1.897E-02**	1.897E-02	1.897E-02	1.897E-02	1.897E-02	1.897E-02	1.897E-02	1.897E-02	1.897E-02	1.897E-02	1.897E-02

Appendix D

VISCOELASTIC EQUATIONS

As stated in methods, the rheological choice for this model is known as the Kelvin-Voigt Standard Linear Solid (figure D.1) and it accounts for both creep and stress relaxation.

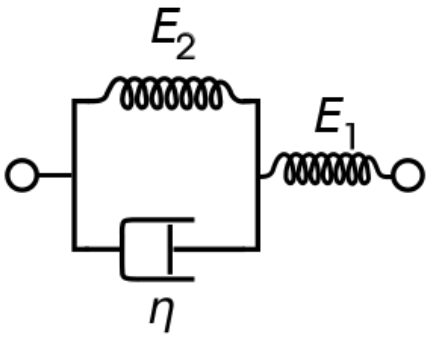

Figure D.1: The Kelvin-Voigt Standard Linear Solid

Governing Equation:

$$\eta E_1 \dot{\epsilon} + E_1 E_2 \epsilon = \eta \dot{\sigma} + (E_1 + E_2)\sigma \tag{D.1}$$

D.1 Creep Response

Creep behavior is defined as a continued displacement of a material with time in spite of a constantly applied stress, meaning that the stress rate $\dot{\sigma} = 0$, which simplifies the governing equation, which once integrated gives:

$$\epsilon = \frac{\sigma_0}{E_1} + \frac{\sigma_0}{E_2}[1 - e^{\left(\frac{-t}{\tau}\right)}] \tag{D.2}$$

The retardation time τ of a viscoelastic material is defined as a delayed response to an applied force and formulated as:

$$\tau = \frac{\eta}{E_2} \tag{D.3}$$

And so equation C.2 can be rewritten as:

$$\epsilon = \frac{1}{E_1} + \frac{1}{E_2}[1 - e^{\left(\frac{-t}{\tau}\right)}]\sigma_0 \tag{D.4}$$

D.2 Stress Relaxation

Stress relaxation is a property of a viscoelastic behavior where stress diminishes over time during the application of constant strain due to a change in the material's internals. This means that the strain rate $\dot{\epsilon} = 0$ which allows the governing equation to be integrated in the following way:

$$\sigma = \frac{E_1\epsilon_0}{E_1 + E_2}[E_2 + E_1 \cdot e^{\left(\frac{-t}{\tau}\right)}] \tag{D.5}$$

The relaxation time τ is the amount of time that it takes the stress to reach an equilibrium value after the strain is initially applied. It can be formulated as:

$$\tau = \frac{\eta}{E_1 + E_2} \tag{D.6}$$

Which allows the SLS stress to be solved as:

$$\sigma = \frac{E_1}{E_1 + E_2}[E_2 + E_1 \cdot e^{\left(\frac{-t}{\tau}\right)}]\epsilon_0 \tag{D.7}$$

Appendix E

CODE

E.1 Abaqus Input File

E.2 Richter User Material Subroutine

```
C
C  SDVINI SUBROUTINE TO INITIALIZE AND KEEP TRACK OF STRESS INCREMENTS
C  FROM THE PREVIOUS CALCULATION
C
      SUBROUTINE SDVINI(STATEV,COORDS,NSTATV,NCRDS,NOEL,NPT,
     1 LAYER,KSPT)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION STATEV(NSTATV),COORDS(NCRDS)
C
C  STATEV 1, 2, AND 3 CORRESPOND TO DSTRES 1, 2, AND 3 IN THAT ORDER
C  WRITE STATEMENTS WERE UTILIZED FOR DEBUGGING PURPOSES
C
      STATEV(1) = 0.0
      STATEV(2) = 0.0
      STATEV(3) = 0.0
C
      RETURN
      END
C
C  3D FORMULATION OF THE STANDARD LINEAR SOLID (KELVIN BODY)
C
      SUBROUTINE UMAT(STRESS,STATEV,DDSDDE,SSE,SPD,SCD,
     1 RPL,DDSDDT,DRPLDE,DRPLDT,
     2 STRAN,DSTRAN,TIME,DTIME,TEMP,DTEMP,PREDEF,DPRED,CMNAME,
     3 NDI,NSHR,NTENS,NSTATV,PROPS,NPROPS,COORDS,DROT,PNEWDT,
     4 CELENT,DFGRD0,DFGRD1,NOEL,NPT,LAYER,KSPT,KSTEP,KINC)
```

```
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*8 CMNAME
      DIMENSION STRESS(NTENS),STATEV(NSTATV),
     1 DDSDDE(NTENS,NTENS),
     2 DDSDDT(NTENS),DRPLDE(NTENS),
     3 STRAN(NTENS),DSTRAN(NTENS),TIME(2),PREDEF(1),DPRED(1),
     4 PROPS(NPROPS),COORDS(3),DROT(3,3),DFGRD0(3,3),DFGRD1(3,3)
      DIMENSION DSTRES(6),D(3,3)
      REAL K_E, G_E,
     1 K_Ke, G_Ke,
     2 Eta_B, Eta_S
C
C  ADDITIONAL CONSTANTS ARE LISTED AS FOLLOWS:
C       K_E IS THE BULK MODULUS OF THE SPRING
C       G_E IS THE SHEAR MODULUS OF THE SPRING
C       K_Ke IS THE BULK MODULUS OF THE SPRING IN THE KELVIN BODY
C       G_Ke IS THE SHEAR MODULUS OF THE SPRING IN THE KELVIN BODY
C       Eta_B IS THE BULK VISCOCITY OF THE DASHPOT IN THE KELVIN BODY
C       Eta_S IS THE SHEAR VISCOSITY OF THE DASHPOT IN THE KELVIN BODY
C
C  CALCULATE MATERIAL PROPERTIES BASED ON USER DEFINED CONSTANTS
C
      K_E = PROPS(1)/(3*(1 - 2*PROPS(4)))
      G_E = PROPS(1)/(2*(1 + PROPS(4)))
      K_Ke = PROPS(2)/(3*(1 - 2*PROPS(5)))
      G_Ke = PROPS(2)/(2*(1 + PROPS(5)))
      Eta_B = PROPS(3)/(3*(1 - 2*PROPS(6)))
      Eta_S = PROPS(3)/(2*(1 + PROPS(6)))
C
C  USER  DEFINED CONSTANTS REFER TO:
C       PROPS(1): THE ELASTIC MODULUS OF THE SPRING
C       PROPS(2): THE ELASTIC MODULUS OF THE SPRING IN THE KELVIN BODY
C       PROPS(3): THE VISCOCITY OF THE DASHPOT IN THE KELVIN BODY
C       PROPS(4): POISSONS RATIO OF THE SPRING
C       PROPS(5): POISSONS RATIO OF THE SPRING IN THE KELVIN BODY
C       PROPS(6): POISSONS RATIO OF THE DASHPOT IN THE KELVIN BODY
C
C  EVALUATE NEW STRESS TENSOR
C
```

```
      EV = 0
      DEV = 0
      SV = 0
      DSV = 0
C
C      WRITE(*,*) 'KINC = ',KINC
C      WRITE(*,*) 'KSTEP = ',KSTEP
C
      DO K1=1,NDI
         EV = EV + STRAN(K1)
         DEV = DEV + DSTRAN(K1)
         SV = SV + STRESS(K1)
         DSV = DSV + STATEV(K1)
      END DO
C
C      WRITE(*,*) 'EV = ',EV
C      WRITE(*,*) 'DEV = ',DEV
C      WRITE(*,*) 'SV = ',SV
C      WRITE(*,*) 'DSV = ',DSV
C
C  EVALUATE DIRECT STRESS COMPONENTS
C
      TERM1A = (6*DTIME*K_E*G_E)/(3*DTIME*K_E*G_E + 2*DTIME*K_E*G_Ke
     1                 + 4*K_E*Eta_S + DTIME*G_E*K_Ke + 2*G_E*Eta_B)
      TERM2 = (G_Ke + ((2*Eta_S)/DTIME))
      TERM3 = (3*K_Ke - 2*G_Ke)/6 + (3*Eta_B - 2*Eta_S)/(3*DTIME)
      TERM4 = (2*G_Ke)
      TERM5 = (3*K_Ke - 2*G_Ke)/3
      TERM6 = (1+(G_Ke/G_E))
      TERM7 = (K_Ke/K_E - G_Ke/G_E)/3
      TERM8 = (K_Ke/K_E - G_Ke/G_E)/6
     1          + (Eta_B/K_E - Eta_S/G_E)/(3*DTIME)
C
      DO K1=1,NDI
         DSTRES(K1) = TERM1A*(TERM2*DSTRAN(K1) + TERM3*DEV
     1     + TERM4*STRAN(K1) + TERM5*EV - TERM6*STRESS(K1) - TERM7*SV
     2     - TERM8*(DSV - STATEV(K1)))
         STRESS(K1) = STRESS(K1) + DSTRES(K1)
      END DO
C
C  SAVE CURRENT STRESS INCREMENTS FOR THE NEXT STRESS CALCULATION
```

```
111   C
112         DO K1 = 1,NDI
113            STATEV(K1) = DSTRES(K1)
114         END DO
115   C
116   C      WRITE(*,*) 'STATEV(1) = ',STATEV(1)
117   C      WRITE(*,*) 'STATEV(2) = ',STATEV(2)
118   C      WRITE(*,*) 'STATEV(3) = ',STATEV(3)
119   C
120   C  EVALUATE SHEAR STRESS COMPONENTS
121   C
122         TERM1B = ((2*DTIME*G_E)/(DTIME*G_E + DTIME*G_Ke + 2*Eta_S))
123         TERM2B = TERM2/2
124         TERM3B = TERM4/2
125         TERM4B = TERM6
126         I1 = NDI
127   C
128         DO K1=1,NSHR
129            I1 = I1+1
130            DSTRES(I1) = TERM1B*(TERM2B*DSTRAN(I1) + TERM3B*STRAN(I1)
131        1     - TERM4B*STRESS(I1))
132            STRESS(I1) = STRESS(I1)+DSTRES(I1)
133         END DO
134   C
135   C  CREATE NEW JACOBIAN
136   C
137         TERM2C = TERM1A*(6*DTIME*G_Ke + 12*Eta_S + 3*DTIME*K_Ke
138        1           - 2*DTIME*G_Ke + 6*Eta_B - 4*Eta_s)/(6*DTIME)
139         TERM3C = TERM1A*(3*DTIME*K_Ke - 2*DTIME*G_Ke + 6*Eta_B
140        1           - 4*Eta_S)/(6*DTIME)
141   C
142         DO K1=1,NTENS
143            DO K2=1,NTENS
144               DDSDDE(K2,K1) = 0
145   C            WRITE(*,*) 'K1 = ',K1
146   C            WRITE(*,*) 'K2 = ',K2
147            END DO
148         END DO
149   C
150         DO K1=1,NDI
151            DDSDDE(K1,K1) = TERM2C
```

```
      END DO
C
      DO K1=2,NDI
         N2 = K1-1
         DO K2=1,N2
            DDSDDE(K2,K1) = TERM3C
            DDSDDE(K1,K2) = TERM3C
C            WRITE(*,*) 'K1 = ',K1
C            WRITE(*,*) 'K2 = ',K2
         END DO
      END DO
C
      TERM4C = TERM1B*(DTIME*G_Ke + 2*Eta_S)/(2*DTIME)
      I1 = NDI
C
      DO K1=1,NSHR
         I1 = I1+1
         DDSDDE(I1,I1) = TERM4C
C         WRITE(*,*) 'I1 = ',I1
      END DO
C
C      WRITE(*,*) 'NTENS = ',NTENS
C      WRITE(*,*) 'NDI = ',NDI
C      WRITE(*,*) 'NSHR = ',NSHR
C      WRITE(*,*) 'G_E = ',G_E
C      WRITE(*,*) 'K_E = ',K_E
C      WRITE(*,*) 'G_Ke = ',G_Ke
C      WRITE(*,*) 'K_Ke = ',K_Ke
C      WRITE(*,*) 'Eta_S = ',Eta_S
C      WRITE(*,*) 'Eta_B = ',Eta_B
C      WRITE(*,*) 'DTIME = ',DTIME
C      WRITE(*,*) 'TERM1A = ',TERM1A
C      WRITE(*,*) 'TERM2 = ',TERM2
C      WRITE(*,*) 'TERM3 = ',TERM3
C      WRITE(*,*) 'TERM4 = ',TERM4
C      WRITE(*,*) 'TERM5 = ',TERM5
C      WRITE(*,*) 'TERM6 = ',TERM6
C      WRITE(*,*) 'TERM7 = ',TERM7
C      WRITE(*,*) 'TERM8 = ',TERM8
C      WRITE(*,*) 'TERM1B = ',TERM1B
C      WRITE(*,*) 'TERM2B = ',TERM2B
```

```
C      WRITE(*,*) 'TERM3B = ',TERM3B
C      WRITE(*,*) 'TERM4B = ',TERM4B
C      WRITE(*,*) 'TERM2C = ',TERM2C
C      WRITE(*,*) 'TERM3C = ',TERM3C
C      WRITE(*,*) 'TERM4C = ',TERM4C
C
       RETURN
       END
```

E.2.1 Variable Dashpot UMAT

```
C
C  SDVINI SUBROUTINE TO INITIALIZE AND KEEP TRACK OF STRESS INCREMENTS
C  FROM THE PREVIOUS CALCULATION
C
       SUBROUTINE SDVINI(STATEV,COORDS,NSTATV,NCRDS,NOEL,NPT,
      1 LAYER,KSPT)
C
       INCLUDE 'ABA_PARAM.INC'
C
       DIMENSION STATEV(NSTATV),COORDS(NCRDS)
C
C  STATEV 1, 2, AND 3 CORRESPOND TO DSTRES 1, 2, AND 3 IN THAT ORDER
C  WRITE STATEMENTS WERE UTILIZED FOR DEBUGGING PURPOSES
C
       STATEV(1) = 0.0
       STATEV(2) = 0.0
       STATEV(3) = 0.0
C
       RETURN
       END
C
C  3D FORMULATION OF THE STANDARD LINEAR SOLID (KELVIN BODY)
C
       SUBROUTINE UMAT(STRESS,STATEV,DDSDDE,SSE,SPD,SCD,
      1 RPL,DDSDDT,DRPLDE,DRPLDT,
      2 STRAN,DSTRAN,TIME,DTIME,TEMP,DTEMP,PREDEF,DPRED,CMNAME,
      3 NDI,NSHR,NTENS,NSTATV,PROPS,NPROPS,COORDS,DROT,PNEWDT,
      4 CELENT,DFGRD0,DFGRD1,NOEL,NPT,LAYER,KSPT,KSTEP,KINC)
```

```
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*8 CMNAME
      DIMENSION STRESS(NTENS),STATEV(NSTATV),
     1 DDSDDE(NTENS,NTENS),
     2 DDSDDT(NTENS),DRPLDE(NTENS),
     3 STRAN(NTENS),DSTRAN(NTENS),TIME(2),PREDEF(1),DPRED(1),
     4 PROPS(NPROPS),COORDS(3),DROT(3,3),DFGRD0(3,3),DFGRD1(3,3)
      DIMENSION DSTRES(6),D(3,3)
      REAL K_E, G_E,
     1 K_Ke, G_Ke,
     2 Eta_B, Eta_S,
     3 Eta_var

C
C  VARIABLE VISCOSITY OF THE DASHPOT DEFINITION
C  WITH PROPS(3) SERVING AS FREQUENCY INPUT AS FOLLOWS:
C
C        C1809 VARIABLE DASHPOT SCHEME
C      Eta_var = (9.08775E-5*((PROPS(3))**5.))
C     1 - (0.00245779*((PROPS(3))**4.))
C     2 + (0.0153295*((PROPS(3))**3.))
C     3 + (0.0684851*((PROPS(3))**2.))
C     4 - (0.83055*(PROPS(3))) + 1.97746

C   C2207 VARIABLE DASHPOT SCHEME
      Eta_var = (-6.61581E-5*((PROPS(3))**5.))
     1 + (2.972E-3*((PROPS(3))**4.))
     2 - (5.016075E-2*((PROPS(3))**3.))
     3 + (3.960115E-1*((PROPS(3))**2.))
     4 - (1.461599*(PROPS(3))) + 2.344206

C   Ensure the polynomial doesn't make Eta_var negative
      Eta_var = abs(Eta_var)
C   Get Eta_var into the correct units for Abaqus
          Eta_var = (Eta_var/1000.)
C
C  ADDITIONAL CONSTANTS ARE LISTED AS FOLLOWS:
C        K_E IS THE BULK MODULUS OF THE SPRING
C        G_E IS THE SHEAR MODULUS OF THE SPRING
```

```
70  C        K_Ke IS THE BULK MODULUS OF THE SPRING IN THE KELVIN BODY
71  C        G_Ke IS THE SHEAR MODULUS OF THE SPRING IN THE KELVIN BODY
72  C        Eta_B IS THE BULK VISCOCITY OF THE DASHPOT IN THE KELVIN BODY
73  C        Eta_S IS THE SHEAR VISCOSITY OF THE DASHPOT IN THE KELVIN BODY
74  C
75  C  CALCULATE MATERIAL PROPERTIES BASED ON USER DEFINED CONSTANTS
76  C
77        K_E = PROPS(1)/(3*(1 - 2*PROPS(4)))
78        G_E = PROPS(1)/(2*(1 + PROPS(4)))
79        K_Ke = PROPS(2)/(3*(1 - 2*PROPS(5)))
80        G_Ke = PROPS(2)/(2*(1 + PROPS(5)))
81        Eta_B = Eta_var/(3*(1 - 2*PROPS(6)))
82        Eta_S = Eta_var/(2*(1 + PROPS(6)))
83  C
84  C  USER  DEFINED CONSTANTS REFER TO:
85  C        PROPS(1): THE ELASTIC MODULUS OF THE SPRING
86  C        PROPS(2): THE ELASTIC MODULUS OF THE SPRING IN THE KELVIN BODY
87  C        PROPS(3): THE VISCOCITY OF THE DASHPOT IN THE KELVIN BODY
88  C        PROPS(4): POISSONS RATIO OF THE SPRING
89  C        PROPS(5): POISSONS RATIO OF THE SPRING IN THE KELVIN BODY
90  C        PROPS(6): POISSONS RATIO OF THE DASHPOT IN THE KELVIN BODY
91  C
92  C  EVALUATE NEW STRESS TENSOR
93  C
94        EV = 0
95        DEV = 0
96        SV = 0
97        DSV = 0
98  C
99  C      WRITE(*,*) 'KINC = ',KINC
100 C      WRITE(*,*) 'KSTEP = ',KSTEP
101 C
102       DO K1=1,NDI
103          EV = EV + STRAN(K1)
104          DEV = DEV + DSTRAN(K1)
105          SV = SV + STRESS(K1)
106          DSV = DSV + STATEV(K1)
107       END DO
108 C
109 C      WRITE(*,*) 'EV = ',EV
110 C      WRITE(*,*) 'DEV = ',DEV
```

```
C      WRITE(*,*) 'SV = ',SV
C      WRITE(*,*) 'DSV = ',DSV
C
C  EVALUATE DIRECT STRESS COMPONENTS
C
       TERM1A = (6*DTIME*K_E*G_E)/(3*DTIME*K_E*G_E + 2*DTIME*K_E*G_Ke
      1                 + 4*K_E*Eta_S + DTIME*G_E*K_Ke + 2*G_E*Eta_B)
       TERM2 = (G_Ke + ((2*Eta_S)/DTIME))
       TERM3 = (3*K_Ke - 2*G_Ke)/6 + (3*Eta_B - 2*Eta_S)/(3*DTIME)
       TERM4 = (2*G_Ke)
       TERM5 = (3*K_Ke - 2*G_Ke)/3
       TERM6 = (1+(G_Ke/G_E))
       TERM7 = (K_Ke/K_E - G_Ke/G_E)/3
       TERM8 = (K_Ke/K_E - G_Ke/G_E)/6
      1          + (Eta_B/K_E - Eta_S/G_E)/(3*DTIME)
C
       DO K1=1,NDI
          DSTRES(K1) = TERM1A*(TERM2*DSTRAN(K1) + TERM3*DEV
      1     + TERM4*STRAN(K1) + TERM5*EV - TERM6*STRESS(K1) - TERM7*SV
      2     - TERM8*(DSV - STATEV(K1)))
          STRESS(K1) = STRESS(K1) + DSTRES(K1)
       END DO
C
C  SAVE CURRENT STRESS INCREMENTS FOR THE NEXT STRESS CALCULATION
C
       DO K1 = 1,NDI
          STATEV(K1) = DSTRES(K1)
       END DO
C
C      WRITE(*,*) 'STATEV(1) = ',STATEV(1)
C      WRITE(*,*) 'STATEV(2) = ',STATEV(2)
C      WRITE(*,*) 'STATEV(3) = ',STATEV(3)
C
C  EVALUATE SHEAR STRESS COMPONENTS
C
       TERM1B = ((2*DTIME*G_E)/(DTIME*G_E + DTIME*G_Ke + 2*Eta_S))
       TERM2B = TERM2/2
       TERM3B = TERM4/2
       TERM4B = TERM6
       I1 = NDI
C
```

```
152         DO K1=1,NSHR
153            I1 = I1+1
154            DSTRES(I1) = TERM1B*(TERM2B*DSTRAN(I1) + TERM3B*STRAN(I1)
155        1      - TERM4B*STRESS(I1))
156            STRESS(I1) = STRESS(I1)+DSTRES(I1)
157         END DO
158   C
159   C  CREATE NEW JACOBIAN
160   C
161         TERM2C = TERM1A*(6*DTIME*G_Ke + 12*Eta_S + 3*DTIME*K_Ke
162        1           - 2*DTIME*G_Ke + 6*Eta_B - 4*Eta_s)/(6*DTIME)
163         TERM3C = TERM1A*(3*DTIME*K_Ke - 2*DTIME*G_Ke + 6*Eta_B
164        1           - 4*Eta_S)/(6*DTIME)
165   C
166         DO K1=1,NTENS
167            DO K2=1,NTENS
168               DDSDDE(K2,K1) = 0
169   C            WRITE(*,*) 'K1 = ',K1
170   C            WRITE(*,*) 'K2 = ',K2
171            END DO
172         END DO
173   C
174         DO K1=1,NDI
175            DDSDDE(K1,K1) = TERM2C
176         END DO
177   C
178         DO K1=2,NDI
179            N2 = K1-1
180            DO K2=1,N2
181               DDSDDE(K2,K1) = TERM3C
182               DDSDDE(K1,K2) = TERM3C
183   C            WRITE(*,*) 'K1 = ',K1
184   C            WRITE(*,*) 'K2 = ',K2
185            END DO
186         END DO
187   C
188         TERM4C = TERM1B*(DTIME*G_Ke + 2*Eta_S)/(2*DTIME)
189         I1 = NDI
190   C
191         DO K1=1,NSHR
192            I1 = I1+1
```

```
          DDSDDE(I1,I1) = TERM4C
C          WRITE(*,*) 'I1 = ',I1
       END DO
C
C      WRITE(*,*) 'NTENS = ',NTENS
C      WRITE(*,*) 'NDI = ',NDI
C      WRITE(*,*) 'NSHR = ',NSHR
C      WRITE(*,*) 'G_E = ',G_E
C      WRITE(*,*) 'K_E = ',K_E
C      WRITE(*,*) 'G_Ke = ',G_Ke
C      WRITE(*,*) 'K_Ke = ',K_Ke
C      WRITE(*,*) 'Eta_S = ',Eta_S
C      WRITE(*,*) 'Eta_B = ',Eta_B
C      WRITE(*,*) 'DTIME = ',DTIME
C      WRITE(*,*) 'TERM1A = ',TERM1A
C      WRITE(*,*) 'TERM2 = ',TERM2
C      WRITE(*,*) 'TERM3 = ',TERM3
C      WRITE(*,*) 'TERM4 = ',TERM4
C      WRITE(*,*) 'TERM5 = ',TERM5
C      WRITE(*,*) 'TERM6 = ',TERM6
C      WRITE(*,*) 'TERM7 = ',TERM7
C      WRITE(*,*) 'TERM8 = ',TERM8
C      WRITE(*,*) 'TERM1B = ',TERM1B
C      WRITE(*,*) 'TERM2B = ',TERM2B
C      WRITE(*,*) 'TERM3B = ',TERM3B
C      WRITE(*,*) 'TERM4B = ',TERM4B
C      WRITE(*,*) 'TERM2C = ',TERM2C
C      WRITE(*,*) 'TERM3C = ',TERM3C
C      WRITE(*,*) 'TERM4C = ',TERM4C
C
       RETURN
       END
```

E.3 Model Generation Script

```
# coding=utf-8
# Do not delete the following import lines
from abaqus import *
from abaqusConstants import *
```

```
import __main__

def TWOXONEHUNDREDFINAL():
    import section
    import regionToolset
    import displayGroupMdbToolset as dgm
    import part
    import material
    import assembly
    import step
    import interaction
    import load
    import mesh
    import job
    import sketch
    import visualization
    import xyPlot
    import displayGroupOdbToolset as dgo
    import connectorBehavior
    import random

#Dspacing mean and standard dev (Control Cran C2207)
    #DSmean=0.0674433
    #DSstdev=0.0011569

#Dspacing mean and standard dev (Control Cran)
    #DSmean=0.06841994
    #DSstdev=0.001281148
#Dspacing mean and standard dev (Control Caud)
#    DSmean=0.066353
#    DSstdev=0.001688634
#Dspacing mean and standard dev (OVX Cran)
    DSmean=0.06715122
   DSstdev=0.00200597
#Dspacing mean and standard dev (OVX Caud)
#    DSmean=0.0675862
#    DSstdev=0.0012142
#Dspacing mean and standard dev specimen C2222
    #DSmean=0.06800059
    #DSstdev=0.00137705
#Random dspacing
```

```
46      ds1 =random.gauss (DSmean,DSstdev)
47      ds2 =random.gauss (DSmean,DSstdev)
48      ds3 =random.gauss (DSmean,DSstdev)
49      ds4 =random.gauss (DSmean,DSstdev)
50      ds5 =random.gauss (DSmean,DSstdev)
51      ds6 =random.gauss (DSmean,DSstdev)
52      ds7 =random.gauss (DSmean,DSstdev)
53      ds8 =random.gauss (DSmean,DSstdev)
54      ds9 =random.gauss (DSmean,DSstdev)
55      ds10 =random.gauss (DSmean,DSstdev)
56      ds11 =random.gauss (DSmean,DSstdev)
57      ds12 =random.gauss (DSmean,DSstdev)
58      ds13 =random.gauss (DSmean,DSstdev)
59      ds14 =random.gauss (DSmean,DSstdev)
60      ds15 =random.gauss (DSmean,DSstdev)
61      ds16 =random.gauss (DSmean,DSstdev)
62      ds17 =random.gauss (DSmean,DSstdev)
63      ds18 =random.gauss (DSmean,DSstdev)
64      ds19 =random.gauss (DSmean,DSstdev)
65      ds20 =random.gauss (DSmean,DSstdev)
66      ds21 =random.gauss (DSmean,DSstdev)
67      ds22 =random.gauss (DSmean,DSstdev)
68      ds23 =random.gauss (DSmean,DSstdev)
69      ds24 =random.gauss (DSmean,DSstdev)
70      ds25 =random.gauss (DSmean,DSstdev)
71      ds26 =random.gauss (DSmean,DSstdev)
72      ds27 =random.gauss (DSmean,DSstdev)
73      ds28 =random.gauss (DSmean,DSstdev)
74      ds29 =random.gauss (DSmean,DSstdev)
75      ds30 =random.gauss (DSmean,DSstdev)
76      ds31 =random.gauss (DSmean,DSstdev)
77      ds32 =random.gauss (DSmean,DSstdev)
78      ds33 =random.gauss (DSmean,DSstdev)
79      ds34 =random.gauss (DSmean,DSstdev)
80      ds35 =random.gauss (DSmean,DSstdev)
81      ds36 =random.gauss (DSmean,DSstdev)
82      ds37 =random.gauss (DSmean,DSstdev)
83      ds38 =random.gauss (DSmean,DSstdev)
84      ds39 =random.gauss (DSmean,DSstdev)
85      ds40 =random.gauss (DSmean,DSstdev)
86      ds41 =random.gauss (DSmean,DSstdev)
```

```
87      ds42 =random.gauss (DSmean,DSstdev)
88      ds43 =random.gauss (DSmean,DSstdev)
89      ds44 =random.gauss (DSmean,DSstdev)
90      ds45 =random.gauss (DSmean,DSstdev)
91      ds46 =random.gauss (DSmean,DSstdev)
92      ds47 =random.gauss (DSmean,DSstdev)
93      ds48 =random.gauss (DSmean,DSstdev)
94      ds49 =random.gauss (DSmean,DSstdev)
95      ds50 =random.gauss (DSmean,DSstdev)
96      ds51 =random.gauss (DSmean,DSstdev)
97      ds52 =random.gauss (DSmean,DSstdev)
98      ds53 =random.gauss (DSmean,DSstdev)
99      ds54 =random.gauss (DSmean,DSstdev)
100     ds55 =random.gauss (DSmean,DSstdev)
101     ds56 =random.gauss (DSmean,DSstdev)
102     ds57 =random.gauss (DSmean,DSstdev)
103     ds58 =random.gauss (DSmean,DSstdev)
104     ds59 =random.gauss (DSmean,DSstdev)
105     ds60 =random.gauss (DSmean,DSstdev)
106     ds61 =random.gauss (DSmean,DSstdev)
107     ds62 =random.gauss (DSmean,DSstdev)
108     ds63 =random.gauss (DSmean,DSstdev)
109     ds64 =random.gauss (DSmean,DSstdev)
110     ds65 =random.gauss (DSmean,DSstdev)
111     ds66 =random.gauss (DSmean,DSstdev)
112     ds67 =random.gauss (DSmean,DSstdev)
113     ds68 =random.gauss (DSmean,DSstdev)
114     ds69 =random.gauss (DSmean,DSstdev)
115     ds70 =random.gauss (DSmean,DSstdev)
116     ds71 =random.gauss (DSmean,DSstdev)
117     ds72 =random.gauss (DSmean,DSstdev)
118     ds73 =random.gauss (DSmean,DSstdev)
119     ds74 =random.gauss (DSmean,DSstdev)
120     ds75 =random.gauss (DSmean,DSstdev)
121     ds76 =random.gauss (DSmean,DSstdev)
122     ds77 =random.gauss (DSmean,DSstdev)
123     ds78 =random.gauss (DSmean,DSstdev)
124     ds79 =random.gauss (DSmean,DSstdev)
125     ds80 =random.gauss (DSmean,DSstdev)
126     ds81 =random.gauss (DSmean,DSstdev)
127     ds82 =random.gauss (DSmean,DSstdev)
```

```
128       ds83 =random.gauss (DSmean,DSstdev)
129       ds84 =random.gauss (DSmean,DSstdev)
130       ds85 =random.gauss (DSmean,DSstdev)
131       ds86 =random.gauss (DSmean,DSstdev)
132       ds87 =random.gauss (DSmean,DSstdev)
133       ds88 =random.gauss (DSmean,DSstdev)
134       ds89 =random.gauss (DSmean,DSstdev)
135       ds90 =random.gauss (DSmean,DSstdev)
136       ds91 =random.gauss (DSmean,DSstdev)
137       ds92 =random.gauss (DSmean,DSstdev)
138       ds93 =random.gauss (DSmean,DSstdev)
139       ds94 =random.gauss (DSmean,DSstdev)
140       ds95 =random.gauss (DSmean,DSstdev)
141       ds96 =random.gauss (DSmean,DSstdev)
142       ds97 =random.gauss (DSmean,DSstdev)
143       ds98 =random.gauss (DSmean,DSstdev)
144       ds99 =random.gauss (DSmean,DSstdev)
145      ds100 =random.gauss (DSmean,DSstdev)
146      ds101 =random.gauss (DSmean,DSstdev)
147      ds102 =random.gauss (DSmean,DSstdev)
148      ds103 =random.gauss (DSmean,DSstdev)
149      ds104 =random.gauss (DSmean,DSstdev)
150      ds105 =random.gauss (DSmean,DSstdev)
151      ds106 =random.gauss (DSmean,DSstdev)
152      ds107 =random.gauss (DSmean,DSstdev)
153      ds108 =random.gauss (DSmean,DSstdev)
154      ds109 =random.gauss (DSmean,DSstdev)
155      ds110 =random.gauss (DSmean,DSstdev)
156      ds111 =random.gauss (DSmean,DSstdev)
157      ds112 =random.gauss (DSmean,DSstdev)
158      ds113 =random.gauss (DSmean,DSstdev)
159      ds114 =random.gauss (DSmean,DSstdev)
160      ds115 =random.gauss (DSmean,DSstdev)
161      ds116 =random.gauss (DSmean,DSstdev)
162      ds117 =random.gauss (DSmean,DSstdev)
163      ds118 =random.gauss (DSmean,DSstdev)
164      ds119 =random.gauss (DSmean,DSstdev)
165      ds120 =random.gauss (DSmean,DSstdev)
166      ds121 =random.gauss (DSmean,DSstdev)
167      ds122 =random.gauss (DSmean,DSstdev)
168      ds123 =random.gauss (DSmean,DSstdev)
```

```
169       ds124 =random.gauss (DSmean,DSstdev)
170       ds125 =random.gauss (DSmean,DSstdev)
171       ds126 =random.gauss (DSmean,DSstdev)
172       ds127 =random.gauss (DSmean,DSstdev)
173       ds128 =random.gauss (DSmean,DSstdev)
174       ds129 =random.gauss (DSmean,DSstdev)
175       ds130 =random.gauss (DSmean,DSstdev)
176       ds131 =random.gauss (DSmean,DSstdev)
177       ds132 =random.gauss (DSmean,DSstdev)
178       ds133 =random.gauss (DSmean,DSstdev)
179       ds134 =random.gauss (DSmean,DSstdev)
180       ds135 =random.gauss (DSmean,DSstdev)
181       ds136 =random.gauss (DSmean,DSstdev)
182       ds137 =random.gauss (DSmean,DSstdev)
183       ds138 =random.gauss (DSmean,DSstdev)
184       ds139 =random.gauss (DSmean,DSstdev)
185       ds140 =random.gauss (DSmean,DSstdev)
186       ds141 =random.gauss (DSmean,DSstdev)
187       ds142 =random.gauss (DSmean,DSstdev)
188       ds143 =random.gauss (DSmean,DSstdev)
189       ds144 =random.gauss (DSmean,DSstdev)
190       ds145 =random.gauss (DSmean,DSstdev)
191       ds146 =random.gauss (DSmean,DSstdev)
192       ds147 =random.gauss (DSmean,DSstdev)
193       ds148 =random.gauss (DSmean,DSstdev)
194       ds149 =random.gauss (DSmean,DSstdev)
195       ds150 =random.gauss (DSmean,DSstdev)
196       ds151 =random.gauss (DSmean,DSstdev)
197       ds152 =random.gauss (DSmean,DSstdev)
198       ds153 =random.gauss (DSmean,DSstdev)
199       ds154 =random.gauss (DSmean,DSstdev)
200       ds155 =random.gauss (DSmean,DSstdev)
201       ds156 =random.gauss (DSmean,DSstdev)
202       ds157 =random.gauss (DSmean,DSstdev)
203       ds158 =random.gauss (DSmean,DSstdev)
204       ds159 =random.gauss (DSmean,DSstdev)
205       ds160 =random.gauss (DSmean,DSstdev)
206       ds161 =random.gauss (DSmean,DSstdev)
207       ds162 =random.gauss (DSmean,DSstdev)
208       ds163 =random.gauss (DSmean,DSstdev)
209       ds164 =random.gauss (DSmean,DSstdev)
```

```
210       ds165 =random.gauss (DSmean,DSstdev)
211       ds166 =random.gauss (DSmean,DSstdev)
212       ds167 =random.gauss (DSmean,DSstdev)
213       ds168 =random.gauss (DSmean,DSstdev)
214       ds169 =random.gauss (DSmean,DSstdev)
215       ds170 =random.gauss (DSmean,DSstdev)
216       ds171 =random.gauss (DSmean,DSstdev)
217       ds172 =random.gauss (DSmean,DSstdev)
218       ds173 =random.gauss (DSmean,DSstdev)
219       ds174 =random.gauss (DSmean,DSstdev)
220       ds175 =random.gauss (DSmean,DSstdev)
221       ds176 =random.gauss (DSmean,DSstdev)
222       ds177 =random.gauss (DSmean,DSstdev)
223       ds178 =random.gauss (DSmean,DSstdev)
224       ds179 =random.gauss (DSmean,DSstdev)
225       ds180 =random.gauss (DSmean,DSstdev)
226       ds181 =random.gauss (DSmean,DSstdev)
227       ds182 =random.gauss (DSmean,DSstdev)
228       ds183 =random.gauss (DSmean,DSstdev)
229       ds184 =random.gauss (DSmean,DSstdev)
230       ds185 =random.gauss (DSmean,DSstdev)
231       ds186 =random.gauss (DSmean,DSstdev)
232       ds187 =random.gauss (DSmean,DSstdev)
233       ds188 =random.gauss (DSmean,DSstdev)
234       ds189 =random.gauss (DSmean,DSstdev)
235       ds190 =random.gauss (DSmean,DSstdev)
236       ds191 =random.gauss (DSmean,DSstdev)
237       ds192 =random.gauss (DSmean,DSstdev)
238       ds193 =random.gauss (DSmean,DSstdev)
239       ds194 =random.gauss (DSmean,DSstdev)
240       ds195 =random.gauss (DSmean,DSstdev)
241       ds196 =random.gauss (DSmean,DSstdev)
242       ds197 =random.gauss (DSmean,DSstdev)
243       ds198 =random.gauss (DSmean,DSstdev)
244       ds199 =random.gauss (DSmean,DSstdev)
245       ds200 =random.gauss (DSmean,DSstdev)
246       y1=.84*ds1
247       y2=.84*ds2
248       y3=.84*ds3
249       y4=.84*ds4
250       y5=.84*ds5
```

```
251         y6=.84*ds6
252         y7=.84*ds7
253         y8=.84*ds8
254         y9=.84*ds9
255         y10=.84*ds10
256         y11=.84*ds11
257         y12=.84*ds12
258         y13=.84*ds13
259         y14=.84*ds14
260         y15=.84*ds15
261         y16=.84*ds16
262         y17=.84*ds17
263         y18=.84*ds18
264         y19=.84*ds19
265         y20=.84*ds20
266         y21=.84*ds21
267         y22=.84*ds22
268         y23=.84*ds23
269         y24=.84*ds24
270         y25=.84*ds25
271         y26=.84*ds26
272         y27=.84*ds27
273         y28=.84*ds28
274         y29=.84*ds29
275         y30=.84*ds30
276         y31=.84*ds31
277         y32=.84*ds32
278         y33=.84*ds33
279         y34=.84*ds34
280         y35=.84*ds35
281         y36=.84*ds36
282         y37=.84*ds37
283         y38=.84*ds38
284         y39=.84*ds39
285         y40=.84*ds40
286         y41=.84*ds41
287         y42=.84*ds42
288         y43=.84*ds43
289         y44=.84*ds44
290         y45=.84*ds45
291         y46=.84*ds46
```

```
292         y47=.84*ds47
293         y48=.84*ds48
294         y49=.84*ds49
295         y50=.84*ds50
296         y51=.84*ds51
297         y52=.84*ds52
298         y53=.84*ds53
299         y54=.84*ds54
300         y55=.84*ds55
301         y56=.84*ds56
302         y57=.84*ds57
303         y58=.84*ds58
304         y59=.84*ds59
305         y60=.84*ds60
306         y61=.84*ds61
307         y62=.84*ds62
308         y63=.84*ds63
309         y64=.84*ds64
310         y65=.84*ds65
311         y66=.84*ds66
312         y67=.84*ds67
313         y68=.84*ds68
314         y69=.84*ds69
315         y70=.84*ds70
316         y71=.84*ds71
317         y72=.84*ds72
318         y73=.84*ds73
319         y74=.84*ds74
320         y75=.84*ds75
321         y76=.84*ds76
322         y77=.84*ds77
323         y78=.84*ds78
324         y79=.84*ds79
325         y80=.84*ds80
326         y81=.84*ds81
327         y82=.84*ds82
328         y83=.84*ds83
329         y84=.84*ds84
330         y85=.84*ds85
331         y86=.84*ds86
332         y87=.84*ds87
```

```
333         y88=.84*ds88
334         y89=.84*ds89
335         y90=.84*ds90
336         y91=.84*ds91
337         y92=.84*ds92
338         y93=.84*ds93
339         y94=.84*ds94
340         y95=.84*ds95
341         y96=.84*ds96
342         y97=.84*ds97
343         y98=.84*ds98
344         y99=.84*ds99
345         y100=.84*ds100
346         y101=.84*ds101
347         y102=.84*ds102
348         y103=.84*ds103
349         y104=.84*ds104
350         y105=.84*ds105
351         y106=.84*ds106
352         y107=.84*ds107
353         y108=.84*ds108
354         y109=.84*ds109
355         y110=.84*ds110
356         y111=.84*ds111
357         y112=.84*ds112
358         y113=.84*ds113
359         y114=.84*ds114
360         y115=.84*ds115
361         y116=.84*ds116
362         y117=.84*ds117
363         y118=.84*ds118
364         y119=.84*ds119
365         y120=.84*ds120
366         y121=.84*ds121
367         y122=.84*ds122
368         y123=.84*ds123
369         y124=.84*ds124
370         y125=.84*ds125
371         y126=.84*ds126
372         y127=.84*ds127
373         y128=.84*ds128
```

```
374       y129=.84*ds129
375       y130=.84*ds130
376       y131=.84*ds131
377       y132=.84*ds132
378       y133=.84*ds133
379       y134=.84*ds134
380       y135=.84*ds135
381       y136=.84*ds136
382       y137=.84*ds137
383       y138=.84*ds138
384       y139=.84*ds139
385       y140=.84*ds140
386       y141=.84*ds141
387       y142=.84*ds142
388       y143=.84*ds143
389       y144=.84*ds144
390       y145=.84*ds145
391       y146=.84*ds146
392       y147=.84*ds147
393       y148=.84*ds148
394       y149=.84*ds149
395       y150=.84*ds150
396       y151=.84*ds151
397       y152=.84*ds152
398       y153=.84*ds153
399       y154=.84*ds154
400       y155=.84*ds155
401       y156=.84*ds156
402       y157=.84*ds157
403       y158=.84*ds158
404       y159=.84*ds159
405       y160=.84*ds160
406       y161=.84*ds161
407       y162=.84*ds162
408       y163=.84*ds163
409       y164=.84*ds164
410       y165=.84*ds165
411       y166=.84*ds166
412       y167=.84*ds167
413       y168=.84*ds168
414       y169=.84*ds169
```

```
415       y170=.84*ds170
416       y171=.84*ds171
417       y172=.84*ds172
418       y173=.84*ds173
419       y174=.84*ds174
420       y175=.84*ds175
421       y176=.84*ds176
422       y177=.84*ds177
423       y178=.84*ds178
424       y179=.84*ds179
425       y180=.84*ds180
426       y181=.84*ds181
427       y182=.84*ds182
428       y183=.84*ds183
429       y184=.84*ds184
430       y185=.84*ds185
431       y186=.84*ds186
432       y187=.84*ds187
433       y188=.84*ds188
434       y189=.84*ds189
435       y190=.84*ds190
436       y191=.84*ds191
437       y192=.84*ds192
438       y193=.84*ds193
439       y194=.84*ds194
440       y195=.84*ds195
441       y196=.84*ds196
442       y197=.84*ds197
443       y198=.84*ds198
444       y199=.84*ds199
445       y200=.84*ds200
446       x1=ds1-y1
447       x2=ds2-y2
448       x3=ds3-y3
449       x4=ds4-y4
450       x5=ds5-y5
451       x6=ds6-y6
452       x7=ds7-y7
453       x8=ds8-y8
454       x9=ds9-y9
455       x10=ds10-y10
```

```
x11=ds11-y11
x12=ds12-y12
x13=ds13-y13
x14=ds14-y14
x15=ds15-y15
x16=ds16-y16
x17=ds17-y17
x18=ds18-y18
x19=ds19-y19
x20=ds20-y20
x21=ds21-y21
x22=ds22-y22
x23=ds23-y23
x24=ds24-y24
x25=ds25-y25
x26=ds26-y26
x27=ds27-y27
x28=ds28-y28
x29=ds29-y29
x30=ds30-y30
x31=ds31-y31
x32=ds32-y32
x33=ds33-y33
x34=ds34-y34
x35=ds35-y35
x36=ds36-y36
x37=ds37-y37
x38=ds38-y38
x39=ds39-y39
x40=ds40-y40
x41=ds41-y41
x42=ds42-y42
x43=ds43-y43
x44=ds44-y44
x45=ds45-y45
x46=ds46-y46
x47=ds47-y47
x48=ds48-y48
x49=ds49-y49
x50=ds50-y50
x51=ds51-y51
```

```
x52=ds52-y52
x53=ds53-y53
x54=ds54-y54
x55=ds55-y55
x56=ds56-y56
x57=ds57-y57
x58=ds58-y58
x59=ds59-y59
x60=ds60-y60
x61=ds61-y61
x62=ds62-y62
x63=ds63-y63
x64=ds64-y64
x65=ds65-y65
x66=ds66-y66
x67=ds67-y67
x68=ds68-y68
x69=ds69-y69
x70=ds70-y70
x71=ds71-y71
x72=ds72-y72
x73=ds73-y73
x74=ds74-y74
x75=ds75-y75
x76=ds76-y76
x77=ds77-y77
x78=ds78-y78
x79=ds79-y79
x80=ds80-y80
x81=ds81-y81
x82=ds82-y82
x83=ds83-y83
x84=ds84-y84
x85=ds85-y85
x86=ds86-y86
x87=ds87-y87
x88=ds88-y88
x89=ds89-y89
x90=ds90-y90
x91=ds91-y91
x92=ds92-y92
```

```
x93=ds93-y93
x94=ds94-y94
x95=ds95-y95
x96=ds96-y96
x97=ds97-y97
x98=ds98-y98
x99=ds99-y99
x100=ds100-y100
x101=ds101-y101
x102=ds102-y102
x103=ds103-y103
x104=ds104-y104
x105=ds105-y105
x106=ds106-y106
x107=ds107-y107
x108=ds108-y108
x109=ds109-y109
x111=ds111-y111
x112=ds112-y112
x113=ds113-y113
x114=ds114-y114
x115=ds115-y115
x116=ds116-y116
x117=ds117-y117
x118=ds118-y118
x119=ds119-y119
x120=ds120-y120
x121=ds121-y121
x122=ds122-y122
x123=ds123-y123
x124=ds124-y124
x125=ds125-y125
x126=ds126-y126
x127=ds127-y127
x128=ds128-y128
x129=ds129-y129
x130=ds130-y130
x131=ds131-y131
x132=ds132-y132
x133=ds133-y133
x134=ds134-y134
```

```
x135=ds135-y135
x136=ds136-y136
x137=ds137-y137
x138=ds138-y138
x139=ds139-y139
x140=ds140-y140
x141=ds141-y141
x142=ds142-y142
x143=ds143-y143
x144=ds144-y144
x145=ds145-y145
x146=ds146-y146
x147=ds147-y147
x148=ds148-y148
x149=ds149-y149
x150=ds150-y150
x151=ds151-y151
x152=ds152-y152
x153=ds153-y153
x154=ds154-y154
x155=ds155-y155
x156=ds156-y156
x157=ds157-y157
x158=ds158-y158
x159=ds159-y159
x160=ds160-y160
x161=ds161-y161
x162=ds162-y162
x163=ds163-y163
x164=ds164-y164
x165=ds165-y165
x166=ds166-y166
x167=ds167-y167
x168=ds168-y168
x169=ds169-y169
x170=ds170-y170
x171=ds171-y171
x172=ds172-y172
x173=ds173-y173
x174=ds174-y174
x175=ds175-y175
```

```
x176=ds176-y176
x177=ds177-y177
x178=ds178-y178
x179=ds179-y179
x180=ds180-y180
x181=ds181-y181
x182=ds182-y182
x183=ds183-y183
x184=ds184-y184
x185=ds185-y185
x186=ds186-y186
x187=ds187-y187
x188=ds188-y188
x189=ds189-y189
x190=ds190-y190
x191=ds191-y191
x192=ds192-y192
x193=ds193-y193
x194=ds194-y194
x195=ds195-y195
x196=ds196-y196
x197=ds197-y197
x198=ds198-y198
x199=ds199-y199
x200=ds200-y200
set1=ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8+ds9+ds10
set2=set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18+ds19+ds20
set3=set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28+ds29+ds30
set4=set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38+ds39+ds30
set5=set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48+ds49+ds50
set6=set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58+ds59+ds60
set7=set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68+ds69+ds70
set8=set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78+ds79+ds80
set9=set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88+ds89+ds80
set10=set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98+ds99+ds100
set11=ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108+ds109+ds110
set12=set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118+ds119+ds120
set13=set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128+ds129+ds130
set14=set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138+ds139+ds140
set15=set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148+ds149+ds150
set16=set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158+ds159+ds160
```

```
661       set17=set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168+ds169+ds170
662       set18=set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178+ds179+ds180
663       set19=set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188+ds189+ds190
664       set20=set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198+ds199+ds200
665   #    print ('dspace1 =',ds1)
666   #    print ('dspace2 =',ds2)
667   #    print ('dspace3 =',ds3)
668   #    print ('dspace4 =',ds4)
669
670   #Model lengths
671       Length1= sum(ds1,ds2,ds3,ds4,ds5,ds6,ds7,ds8,ds9,ds10,
672                    ds11,ds12,ds13,ds14,ds15,ds16,ds17,ds18,ds19,ds20,
673                    ds21,ds22,ds23,ds24,ds25,ds26,ds27,ds28,ds29,ds30,
674                    ds31,ds32,ds33,ds34,ds35,ds36,ds37,ds38,ds39,ds40,
675                    ds41,ds42,ds43,ds44,ds45,ds46,ds47,ds48,ds49,ds50,
676                    ds51,ds52,ds53,ds54,ds55,ds56,ds57,ds58,ds59,ds60,
677                    ds61,ds62,ds63,ds64,ds65,ds66,ds67,ds68,ds69,ds70,
678                    ds71,ds72,ds73,ds74,ds75,ds76,ds77,ds78,ds79,ds80,
679                    ds81,ds82,ds83,ds84,ds85,ds86,ds87,ds88,ds89,ds90,
680                    ds91,ds92,ds93,ds94,ds95,ds96,ds97,ds98,ds99,ds100)
681       Length2= sum(ds101,ds102,ds103,ds104,ds105,ds106,ds107,ds108,ds109,ds110,
682                    ds111,ds112,ds113,ds114,ds115,ds116,ds117,ds118,ds119,ds120,
683                    ds121,ds122,ds123,ds124,ds125,ds126,ds127,ds128,ds129,ds130,
684                    ds131,ds132,ds133,ds134,ds135,ds136,ds137,ds138,ds139,ds140,
685                    ds141,ds142,ds143,ds144,ds145,ds146,ds147,ds148,ds149,ds150,
686                    ds151,ds152,ds153,ds154,ds155,ds156,ds157,ds158,ds159,ds160,
687                    ds161,ds162,ds163,ds164,ds165,ds166,ds167,ds168,ds169,ds170,
688                    ds171,ds172,ds173,ds174,ds175,ds176,ds177,ds178,ds179,ds180,
689                    ds181,ds182,ds183,ds184,ds185,ds186,ds187,ds188,ds189,ds190,
690                    ds191,ds192,ds193,ds194,ds195,ds196,ds197,ds198,ds199,ds200)
691       LengthF= max(Length1, Length2)
692   #    LengthFh=LengthF/2
693       print ('Length1 =',Length1)
694       print ('Length2 =',Length2)
695       print ('LengthF =',LengthF)
696       LengthDiff=Length1-Length2
697       print ('Difference in Row Length =',LengthDiff)
698   #LengthF=Length1 (ie. The Top Row is Larger; Bottom Row Has Spacer)
699       if LengthF == Length1:
700         s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
701             sheetSize=200.0)
```

```
        g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
        s1.setPrimaryObject(option=STANDALONE)
        s1.rectangle(point1=(0.0, 0.0), point2=(LengthF, 0.007))
#       session.viewports['Viewport: 1'].view.fitView()
        p = mdb.models['Model-1'].Part(name='Composite Bone',
            dimensionality=TWO_D_PLANAR, type=DEFORMABLE_BODY)
        p = mdb.models['Model-1'].parts['Composite Bone']
        p.BaseShell(sketch=s1)
        s1.unsetPrimaryObject()
        session.viewports['Viewport: 1'].setValues(displayedObject=p)
        del mdb.models['Model-1'].sketches['__profile__']
#       p = mdb.models['Model-1'].parts['Composite Bone']
#       p.DatumPointByCoordinate(coords=(0.0, 0.00125, 0.0))
#       p.DatumPointByCoordinate(coords=(0.0, 0.00275, 0.0))
#       p.DatumPointByCoordinate(coords=(0.0, 0.00425, 0.0))
#       p.DatumPointByCoordinate(coords=(0.0, 0.00575, 0.0))
#       p.DatumPointByCoordinate(coords=(0.05628, 0.007, 0.0))
#       p.DatumPointByCoordinate(coords=(0.067, 0.007, 0.0))
#       p.DatumPointByCoordinate(coords=(0.07772, 0.007, 0.0))
#       p.DatumPointByCoordinate(coords=(0.01072, 0.0, 0.0))
#       p.DatumPointByCoordinate(coords=(0.067, 0.0, 0.0))
#       p.DatumPointByCoordinate(coords=(0.12328, 0.0, 0.0))
#       p.DatumPointByCoordinate(coords=(0.13, 0.0, 0.0))
        f, e1, d2 = p.faces, p.edges, p.datums
        t = p.MakeSketchTransform(sketchPlane=f[0], sketchPlaneSide=SIDE1, origin=(
            0.0, 0.0, 0.0))
        s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
            sheetSize=0.268, gridSpacing=0.006, transform=t)
        g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
        s.sketchOptions.setValues(decimalPlaces=3)
        s.setPrimaryObject(option=SUPERIMPOSE)
        p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
        s.Line(point1=(0, 0.00125), point2=(LengthF, 0.00125))
        s.HorizontalConstraint(entity=g[6], addUndoState=False)
        s.Line(point1=(0, 0.00275), point2=(LengthF, 0.00275))
        s.HorizontalConstraint(entity=g[7], addUndoState=False)
        s.Line(point1=(0, 0.0035), point2=(LengthF, 0.0035))
        s.HorizontalConstraint(entity=g[8], addUndoState=False)
        s.Line(point1=(0, 0.00425), point2=(LengthF, 0.00425))
        s.HorizontalConstraint(entity=g[9], addUndoState=False)
        s.Line(point1=(0, 0.00575), point2=(LengthF, 0.00575))
```

```
743         s.HorizontalConstraint(entity=g[10], addUndoState=False)
744     #Top Row, Dspace 1-10
745         s.Line(point1=(y1, 0.007), point2=(y1, 0.00575))
746         s.Line(point1=(ds1, 0.007), point2=(ds1, 0.0035))
747         s.Line(point1=(ds1+x2, 0.007), point2=(ds1+x2, 0.00575))
748         s.Line(point1=(ds1+ds2, 0.007), point2=(ds1+ds2, 0.0035))
749         s.Line(point1=(ds1+ds2+y3, 0.007), point2=(ds1+ds2+y3, 0.00575))
750         s.Line(point1=(ds1+ds2+ds3, 0.007), point2=(ds1+ds2+ds3, 0.0035))
751         s.Line(point1=(ds1+ds2+ds3+x4, 0.007), point2=(ds1+ds2+ds3+x4, 0.00575))
752         s.Line(point1=(ds1+ds2+ds3+ds4, 0.007), point2=(ds1+ds2+ds3+ds4, 0.0035))
753         s.Line(point1=(ds1+ds2+ds3+ds4+y5, 0.007), point2=(ds1+ds2+ds3+ds4+y5, 0.00575))
754         s.Line(point1=(ds1+ds2+ds3+ds4+ds5, 0.007), point2=(ds1+ds2+ds3+ds4+ds5, 0.0035))
755         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+x6, 0.007), point2=(ds1+ds2+ds3+ds4+ds5+x6, 0.00575))
756         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+ds6, 0.007), point2=(ds1+ds2+ds3+ds4+ds5+ds6, 0.0035))
757         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+ds6+y7, 0.007), point2=(ds1+ds2+ds3+ds4+ds5+ds6+y7, 0.00575))
758         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+ds6+ds7, 0.007), point2=(ds1+ds2+ds3+ds4+ds5+ds6+ds7,
            ↪  0.0035))
759         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+x8, 0.007), point2=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+x8,
            ↪  0.00575))
760         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8, 0.007), point2=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8,
            ↪  0.0035))
761         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8+y9, 0.007),
            ↪  point2=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8+y9, 0.00575))
762         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8+ds9, 0.007),
            ↪  point2=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8+ds9, 0.0035))
763         s.Line(point1=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8+ds9+x10, 0.007),
            ↪  point2=(ds1+ds2+ds3+ds4+ds5+ds6+ds7+ds8+ds9+x10, 0.00575))
764         s.Line(point1=(set1, 0.007), point2=(set1, 0.0035))
765     #Top Row, Dspace 11-20
766         s.Line(point1=(set1+y11, 0.007), point2=(set1+y11, 0.00575))
767         s.Line(point1=(set1+ds11, 0.007), point2=(set1+ds11, 0.0035))
768         s.Line(point1=(set1+ds11+x12, 0.007), point2=(set1+ds11+x12, 0.00575))
769         s.Line(point1=(set1+ds11+ds12, 0.007), point2=(set1+ds11+ds12, 0.0035))
770         s.Line(point1=(set1+ds11+ds12+y13, 0.007), point2=(set1+ds11+ds12+y13, 0.00575))
771         s.Line(point1=(set1+ds11+ds12+ds13, 0.007), point2=(set1+ds11+ds12+ds13, 0.0035))
772         s.Line(point1=(set1+ds11+ds12+ds13+x14, 0.007), point2=(set1+ds11+ds12+ds13+x14, 0.00575))
773         s.Line(point1=(set1+ds11+ds12+ds13+ds14, 0.007), point2=(set1+ds11+ds12+ds13+ds14, 0.0035))
774         s.Line(point1=(set1+ds11+ds12+ds13+ds14+y15, 0.007), point2=(set1+ds11+ds12+ds13+ds14+y15,
            ↪  0.00575))
775         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15, 0.007), point2=(set1+ds11+ds12+ds13+ds14+ds15,
            ↪  0.0035))
```

```
776         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+x16, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+x16, 0.00575))
777         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+ds16, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+ds16, 0.0035))
778         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+ds16+y17, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+ds16+y17, 0.00575))
779         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17, 0.0035))
780         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+x18, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+x18, 0.00575))
781         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18, 0.0035))
782         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18+y19, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18+y19, 0.00575))
783         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18+ds19, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18+ds19, 0.0035))
784         s.Line(point1=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18+ds19+x20, 0.007),
            ↪  point2=(set1+ds11+ds12+ds13+ds14+ds15+ds16+ds17+ds18+ds19+x20, 0.00575))
785     #Top Row, Dspace 21-30
786
787         s.Line(point1=(set2, 0.007), point2=(set2, 0.0035))
788         s.Line(point1=(set2+y21, 0.007), point2=(set2+y21, 0.00575))
789         s.Line(point1=(set2+ds21, 0.007), point2=(set2+ds21, 0.0035))
790         s.Line(point1=(set2+ds21+x22, 0.007), point2=(set2+ds21+x22, 0.00575))
791         s.Line(point1=(set2+ds21+ds22, 0.007), point2=(set2+ds21+ds22, 0.0035))
792         s.Line(point1=(set2+ds21+ds22+y23, 0.007), point2=(set2+ds21+ds22+y23, 0.00575))
793         s.Line(point1=(set2+ds21+ds22+ds23, 0.007), point2=(set2+ds21+ds22+ds23, 0.0035))
794         s.Line(point1=(set2+ds21+ds22+ds23+x24, 0.007), point2=(set2+ds21+ds22+ds23+x24, 0.00575))
795         s.Line(point1=(set2+ds21+ds22+ds23+ds24, 0.007), point2=(set2+ds21+ds22+ds23+ds24, 0.0035))
796         s.Line(point1=(set2+ds21+ds22+ds23+ds24+y25, 0.007), point2=(set2+ds21+ds22+ds23+ds24+y25,
            ↪  0.00575))
797         s.Line(point1=(set2+ds21+ds22+ds23+ds24+ds25, 0.007), point2=(set2+ds21+ds22+ds23+ds24+ds25,
            ↪  0.0035))
798         s.Line(point1=(set2+ds21+ds22+ds23+ds24+ds25+x26, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+x26, 0.00575))
799         s.Line(point1=(set2+ds21+ds22+ds23+ds24+ds25+ds26, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+ds26, 0.0035))
800         s.Line(point1=(set2+ds21+ds22+ds23+ds24+ds25+ds26+y27, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+ds26+y27, 0.00575))
801         s.Line(point1=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27, 0.0035))
```

```
802         s.Line(point1=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+x28, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+x28, 0.00575))
803         s.Line(point1=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28, 0.0035))
804         s.Line(point1=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28+y29, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28+y29, 0.00575))
805         s.Line(point1=(set2++ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28+ds29, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28+ds29, 0.0035))
806         s.Line(point1=(set2++ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28+ds29+x30, 0.007),
            ↪  point2=(set2+ds21+ds22+ds23+ds24+ds25+ds26+ds27+ds28+ds29+x30, 0.00575))
807     #Top Row, Dspace 31-40
808         s.Line(point1=(set3, 0.007), point2=(set3, 0.0035))
809         s.Line(point1=(set3+y31, 0.007), point2=(set3+y31, 0.00575))
810         s.Line(point1=(set3+ds31, 0.007), point2=(set3+ds31, 0.0035))
811         s.Line(point1=(set3+ds31+x32, 0.007), point2=(set3+ds31+x32, 0.00575))
812         s.Line(point1=(set3+ds31+ds32, 0.007), point2=(set3+ds31+ds32, 0.0035))
813         s.Line(point1=(set3+ds31+ds32+y33, 0.007), point2=(set3+ds31+ds32+y33, 0.00575))
814         s.Line(point1=(set3+ds31+ds32+ds33, 0.007), point2=(set3+ds31+ds32+ds33, 0.0035))
815         s.Line(point1=(set3+ds31+ds32+ds33+x34, 0.007), point2=(set3+ds31+ds32+ds33+x34, 0.00575))
816         s.Line(point1=(set3+ds31+ds32+ds33+ds34, 0.007), point2=(set3+ds31+ds32+ds33+ds34, 0.0035))
817         s.Line(point1=(set3+ds31+ds32+ds33+ds34+y35, 0.007), point2=(set3+ds31+ds32+ds33+ds34+y35,
            ↪  0.00575))
818         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35, 0.007), point2=(set3+ds31+ds32+ds33+ds34+ds35,
            ↪  0.0035))
819         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+x36, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+x36, 0.00575))
820         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+ds36, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+ds36, 0.0035))
821         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+ds36+y37, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+ds36+y37, 0.00575))
822         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37, 0.0035))
823         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+x38, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+x38, 0.00575))
824         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38, 0.0035))
825         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38+y39, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38+y39, 0.00575))
826         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38+ds39, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38+ds39, 0.0035))
```

```
827         s.Line(point1=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38+ds39+x40, 0.007),
            ↪  point2=(set3+ds31+ds32+ds33+ds34+ds35+ds36+ds37+ds38+ds39+x40, 0.00575))
828     #Top Row, Dspace 41-50
829         s.Line(point1=(set4, 0.007), point2=(set4, 0.0035))
830         s.Line(point1=(set4+y41, 0.007), point2=(set4+y41, 0.00575))
831         s.Line(point1=(set4+ds41, 0.007), point2=(set4+ds41, 0.0035))
832         s.Line(point1=(set4+ds41+x42, 0.007), point2=(set4+ds41+x42, 0.00575))
833         s.Line(point1=(set4+ds41+ds42, 0.007), point2=(set4+ds41+ds42, 0.0035))
834         s.Line(point1=(set4+ds41+ds42+y43, 0.007), point2=(set4+ds41+ds42+y43, 0.00575))
835         s.Line(point1=(set4+ds41+ds42+ds43, 0.007), point2=(set4+ds41+ds42+ds43, 0.0035))
836         s.Line(point1=(set4+ds41+ds42+ds43+x44, 0.007), point2=(set4+ds41+ds42+ds43+x44, 0.00575))
837         s.Line(point1=(set4+ds41+ds42+ds43+ds44, 0.007), point2=(set4+ds41+ds42+ds43+ds44, 0.0035))
838         s.Line(point1=(set4+ds41+ds42+ds43+ds44+y45, 0.007), point2=(set4+ds41+ds42+ds43+ds44+y45,
            ↪  0.00575))
839         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45, 0.007), point2=(set4+ds41+ds42+ds43+ds44+ds45,
            ↪  0.0035))
840         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+x46, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+x46, 0.00575))
841         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+ds46, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+ds46, 0.0035))
842         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+ds46+y47, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+ds46+y47, 0.00575))
843         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47, 0.0035))
844         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+x48, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+x48, 0.00575))
845         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48, 0.0035))
846         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48+y49, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48+y49, 0.00575))
847         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48+ds49, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48+ds49, 0.0035))
848         s.Line(point1=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48+ds49+x50, 0.007),
            ↪  point2=(set4+ds41+ds42+ds43+ds44+ds45+ds46+ds47+ds48+ds49+x50, 0.00575))
849     #Top Row, Dspace 51-60
850         s.Line(point1=(set5, 0.007), point2=(set5, 0.0035))
851         s.Line(point1=(set5+y51, 0.007), point2=(set5+y51, 0.00575))
852         s.Line(point1=(set5+ds51, 0.007), point2=(set5+ds51, 0.0035))
853         s.Line(point1=(set5+ds51+x52, 0.007), point2=(set5+ds51+x52, 0.00575))
854         s.Line(point1=(set5+ds51+ds52, 0.007), point2=(set5+ds51+ds52, 0.0035))
855         s.Line(point1=(set5+ds51+ds52+y53, 0.007), point2=(set5+ds51+ds52+y53, 0.00575))
```

```
856         s.Line(point1=(set5+ds51+ds52+ds53, 0.007), point2=(set5+ds51+ds52+ds53, 0.0035))
857         s.Line(point1=(set5+ds51+ds52+ds53+x54, 0.007), point2=(set5+ds51+ds52+ds53+x54, 0.00575))
858         s.Line(point1=(set5+ds51+ds52+ds53+ds54, 0.007), point2=(set5+ds51+ds52+ds53+ds54, 0.0035))
859         s.Line(point1=(set5+ds51+ds52+ds53+ds54+y55, 0.007), point2=(set5+ds51+ds52+ds53+ds54+y55,
            ↪  0.00575))
860         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55, 0.007), point2=(set5+ds51+ds52+ds53+ds54+ds55,
            ↪  0.0035))
861         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+x56, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+x56, 0.00575))
862         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+ds56, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+ds56, 0.0035))
863         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+ds56+y57, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+ds56+y57, 0.00575))
864         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57, 0.0035))
865         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+x58, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+x58, 0.00575))
866         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58, 0.0035))
867         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58+y59, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58+y59, 0.00575))
868         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58+ds59, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58+ds59, 0.0035))
869         s.Line(point1=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58+ds59+x60, 0.007),
            ↪  point2=(set5+ds51+ds52+ds53+ds54+ds55+ds56+ds57+ds58+ds59+x60, 0.00575))
870     #Top Row, Dspace 61-70
871         s.Line(point1=(set6, 0.007), point2=(set6, 0.0035))
872         s.Line(point1=(set6+y61, 0.007), point2=(set6+y61, 0.00575))
873         s.Line(point1=(set6+ds61, 0.007), point2=(set6+ds61, 0.0035))
874         s.Line(point1=(set6+ds61+x62, 0.007), point2=(set6+ds61+x62, 0.00575))
875         s.Line(point1=(set6+ds61+ds62, 0.007), point2=(set6+ds61+ds62, 0.0035))
876         s.Line(point1=(set6+ds61+ds62+y63, 0.007), point2=(set6+ds61+ds62+y63, 0.00575))
877         s.Line(point1=(set6+ds61+ds62+ds63, 0.007), point2=(set6+ds61+ds62+ds63, 0.0035))
878         s.Line(point1=(set6+ds61+ds62+ds63+x64, 0.007), point2=(set6+ds61+ds62+ds63+x64, 0.00575))
879         s.Line(point1=(set6+ds61+ds62+ds63+ds64, 0.007), point2=(set6+ds61+ds62+ds63+ds64, 0.0035))
880         s.Line(point1=(set6+ds61+ds62+ds63+ds64+y65, 0.007), point2=(set6+ds61+ds62+ds63+ds64+y65,
            ↪  0.00575))
881         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65, 0.007), point2=(set6+ds61+ds62+ds63+ds64+ds65,
            ↪  0.0035))
882         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+x66, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+x66, 0.00575))
```

```
883         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+ds66, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+ds66, 0.0035))
884         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+ds66+y67, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+ds66+y67, 0.00575))
885         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67, 0.0035))
886         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+x68, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+x68, 0.00575))
887         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68, 0.0035))
888         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68+y69, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68+y69, 0.00575))
889         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68+ds69, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68+ds69, 0.0035))
890         s.Line(point1=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68+ds69+x70, 0.007),
            ↪  point2=(set6+ds61+ds62+ds63+ds64+ds65+ds66+ds67+ds68+ds69+x70, 0.00575))
891     #Top Row, Dspace 71-80
892         s.Line(point1=(set7, 0.007), point2=(set7, 0.0035))
893         s.Line(point1=(set7+y71, 0.007), point2=(set7+y71, 0.00575))
894         s.Line(point1=(set7+ds71, 0.007), point2=(set7+ds71, 0.0035))
895         s.Line(point1=(set7+ds71+x72, 0.007), point2=(set7+ds71+x72, 0.00575))
896         s.Line(point1=(set7+ds71+ds72, 0.007), point2=(set7+ds71+ds72, 0.0035))
897         s.Line(point1=(set7+ds71+ds72+y73, 0.007), point2=(set7+ds71+ds72+y73, 0.00575))
898         s.Line(point1=(set7+ds71+ds72+ds73, 0.007), point2=(set7+ds71+ds72+ds73, 0.0035))
899         s.Line(point1=(set7+ds71+ds72+ds73+x74, 0.007), point2=(set7+ds71+ds72+ds73+x74, 0.00575))
900         s.Line(point1=(set7+ds71+ds72+ds73+ds74, 0.007), point2=(set7+ds71+ds72+ds73+ds74, 0.0035))
901         s.Line(point1=(set7+ds71+ds72+ds73+ds74+y75, 0.007), point2=(set7+ds71+ds72+ds73+ds74+y75,
            ↪  0.00575))
902         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75, 0.007), point2=(set7+ds71+ds72+ds73+ds74+ds75,
            ↪  0.0035))
903         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+x76, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+x76, 0.00575))
904         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+ds76, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+ds76, 0.0035))
905         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+ds76+y77, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+ds76+y77, 0.00575))
906         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77, 0.0035))
907         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+x78, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+x78, 0.00575))
```

```
908         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78, 0.0035))
909         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78+y79, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78+y79, 0.00575))
910         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78+ds79, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78+ds79, 0.0035))
911         s.Line(point1=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78+ds79+x80, 0.007),
            ↪  point2=(set7+ds71+ds72+ds73+ds74+ds75+ds76+ds77+ds78+ds79+x80, 0.00575))
912     #Top Row, Dspace 81-90
913         s.Line(point1=(set8, 0.007), point2=(set8, 0.0035))
914         s.Line(point1=(set8+y81, 0.007), point2=(set8+y81, 0.00575))
915         s.Line(point1=(set8+ds81, 0.007), point2=(set8+ds81, 0.0035))
916         s.Line(point1=(set8+ds81+x82, 0.007), point2=(set8+ds81+x82, 0.00575))
917         s.Line(point1=(set8+ds81+ds82, 0.007), point2=(set8+ds81+ds82, 0.0035))
918         s.Line(point1=(set8+ds81+ds82+y83, 0.007), point2=(set8+ds81+ds82+y83, 0.00575))
919         s.Line(point1=(set8+ds81+ds82+ds83, 0.007), point2=(set8+ds81+ds82+ds83, 0.0035))
920         s.Line(point1=(set8+ds81+ds82+ds83+x84, 0.007), point2=(set8+ds81+ds82+ds83+x84, 0.00575))
921         s.Line(point1=(set8+ds81+ds82+ds83+ds84, 0.007), point2=(set8+ds81+ds82+ds83+ds84, 0.0035))
922         s.Line(point1=(set8+ds81+ds82+ds83+ds84+y85, 0.007), point2=(set8+ds81+ds82+ds83+ds84+y85,
            ↪  0.00575))
923         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85, 0.007), point2=(set8+ds81+ds82+ds83+ds84+ds85,
            ↪  0.0035))
924         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+x86, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+x86, 0.00575))
925         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+ds86, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+ds86, 0.0035))
926         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+ds86+y87, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+ds86+y87, 0.00575))
927         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87, 0.0035))
928         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+x88, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+x88, 0.00575))
929         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88, 0.0035))
930         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88+y89, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88+y89, 0.00575))
931         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88+ds89, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88+ds89, 0.0035))
932         s.Line(point1=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88+ds89+x90, 0.007),
            ↪  point2=(set8+ds81+ds82+ds83+ds84+ds85+ds86+ds87+ds88+ds89+x90, 0.00575))
933     #Top Row, Dspace 91-100
```

```
        s.Line(point1=(set9, 0.007), point2=(set9, 0.0035))
        s.Line(point1=(set9+y91, 0.007), point2=(set9+y91, 0.00575))
        s.Line(point1=(set9+ds91, 0.007), point2=(set9+ds91, 0.0035))
        s.Line(point1=(set9+ds91+x92, 0.007), point2=(set9+ds91+x92, 0.00575))
        s.Line(point1=(set9+ds91+ds92, 0.007), point2=(set9+ds91+ds92, 0.0035))
        s.Line(point1=(set9+ds91+ds92+y93, 0.007), point2=(set9+ds91+ds92+y93, 0.00575))
        s.Line(point1=(set9+ds91+ds92+ds93, 0.007), point2=(set9+ds91+ds92+ds93, 0.0035))
        s.Line(point1=(set9+ds91+ds92+ds93+x94, 0.007), point2=(set9+ds91+ds92+ds93+x94, 0.00575))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94, 0.007), point2=(set9+ds91+ds92+ds93+ds94, 0.0035))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+y95, 0.007), point2=(set9+ds91+ds92+ds93+ds94+y95,
        ↪  0.00575))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95, 0.007), point2=(set9+ds91+ds92+ds93+ds94+ds95,
        ↪  0.0035))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+x96, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+x96, 0.00575))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+ds96, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+ds96, 0.0035))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+ds96+y97, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+ds96+y97, 0.00575))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97, 0.0035))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+x98, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+x98, 0.00575))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98, 0.0035))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98+y99, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98+y99, 0.00575))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98+ds99, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98+ds99, 0.0035))
        s.Line(point1=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98+ds99+x100, 0.007),
        ↪  point2=(set9+ds91+ds92+ds93+ds94+ds95+ds96+ds97+ds98+ds99+x100, 0.00575))
    #Bottom Row, Dspace 1-10
        s.Line(point1=(x101, 0.0), point2=(x101, 0.00125))
        s.Line(point1=(ds101, 0.0), point2=(ds101, 0.0035))
        s.Line(point1=(ds101+y102, 0), point2=(ds101+y102, 0.00125))
        s.Line(point1=(ds101+ds102, 0.0), point2=(ds101+ds102, 0.0035))
        s.Line(point1=(ds101+ds102+x103, 0.0), point2=(ds101+ds102+x103, 0.00125))
        s.Line(point1=(ds101+ds102+ds103, 0.0), point2=(ds101+ds102+ds103, 0.0035))
        s.Line(point1=(ds101+ds102+ds103+y104, 0.0), point2=(ds101+ds102+ds103+y104, 0.00125))
        s.Line(point1=(ds101+ds102+ds103+ds104, 0.0), point2=(ds101+ds102+ds103+ds104, 0.0035))
```

```
963         s.Line(point1=(ds101+ds102+ds103+ds104+x105, 0.0), point2=(ds101+ds102+ds103+ds104+x105,
            ↪  0.00125))
964         s.Line(point1=(ds101+ds102+ds103+ds104+ds105, 0.0), point2=(ds101+ds102+ds103+ds104+ds105,
            ↪  0.0035))
965         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+y106, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+y106, 0.00125))
966         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+ds106, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+ds106, 0.0035))
967         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+ds106+x107, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+ds106+x107, 0.00125))
968         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+ds106+ds107, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+ds106+ds107, 0.0035))
969         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+y108, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+y108, 0.00125))
970         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108, 0.0035))
971         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108+x109, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108+x109, 0.00125))
972         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108+ds109, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108+ds109, 0.0035))
973         s.Line(point1=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108+ds109+y110, 0.0),
            ↪  point2=(ds101+ds102+ds103+ds104+ds105+ds106+ds107+ds108+ds109+y110, 0.00125))
974         s.Line(point1=(set11, 0.0), point2=(set11, 0.0035))
975     #Bottom Row, Dspace 11-20
976         s.Line(point1=(set11+x111, 0.0), point2=(set11+x111, 0.00125))
977         s.Line(point1=(set11+ds111, 0.0), point2=(set11+ds111, 0.0035))
978         s.Line(point1=(set11+ds111+y112, 0), point2=(set11+ds111+y112, 0.00125))
979         s.Line(point1=(set11+ds111+ds112, 0.0), point2=(set11+ds111+ds112, 0.0035))
980         s.Line(point1=(set11+ds111+ds112+x113, 0.0), point2=(set11+ds111+ds112+x113, 0.00125))
981         s.Line(point1=(set11+ds111+ds112+ds113, 0.0), point2=(set11+ds111+ds112+ds113, 0.0035))
982         s.Line(point1=(set11+ds111+ds112+ds113+y114, 0.0), point2=(set11+ds111+ds112+ds113+y114,
            ↪  0.00125))
983         s.Line(point1=(set11+ds111+ds112+ds113+ds114, 0.0), point2=(set11+ds111+ds112+ds113+ds114,
            ↪  0.0035))
984         s.Line(point1=(set11+ds111+ds112+ds113+ds114+x115, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+x115, 0.00125))
985         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115, 0.0035))
986         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+y116, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+y116, 0.00125))
```

```
987         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+ds116, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+ds116, 0.0035))
988         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+ds116+x117, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+ds116+x117, 0.00125))
989         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117, 0.0035))
990         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+y118, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+y118, 0.00125))
991         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118, 0.0035))
992         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118+x119, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118+x119, 0.00125))
993         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118+ds119, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118+ds119, 0.0035))
994         s.Line(point1=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118+ds119+y120, 0.0),
            ↪  point2=(set11+ds111+ds112+ds113+ds114+ds115+ds116+ds117+ds118+ds119+y120, 0.00125))
995         s.Line(point1=(set12, 0.0), point2=(set12, 0.0035))
996     #Bottom Row, Dspace 21-30
997         s.Line(point1=(set12+x121, 0.0), point2=(set12+x121, 0.00125))
998         s.Line(point1=(set12+ds121, 0.0), point2=(set12+ds121, 0.0035))
999         s.Line(point1=(set12+ds121+y122, 0), point2=(set12+ds121+y122, 0.00125))
1000        s.Line(point1=(set12+ds121+ds122, 0.0), point2=(set12+ds121+ds122, 0.0035))
1001        s.Line(point1=(set12+ds121+ds122+x123, 0.0), point2=(set12+ds121+ds122+x123, 0.00125))
1002        s.Line(point1=(set12+ds121+ds122+ds123, 0.0), point2=(set12+ds121+ds122+ds123, 0.0035))
1003        s.Line(point1=(set12+ds121+ds122+ds123+y124, 0.0), point2=(set12+ds121+ds122+ds123+y124,
            ↪  0.00125))
1004        s.Line(point1=(set12+ds121+ds122+ds123+ds124, 0.0), point2=(set12+ds121+ds122+ds123+ds124,
            ↪  0.0035))
1005        s.Line(point1=(set12+ds121+ds122+ds123+ds124+x125, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+x125, 0.00125))
1006        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125, 0.0035))
1007        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+y126, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+y126, 0.00125))
1008        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+ds126, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+ds126, 0.0035))
1009        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+ds126+x127, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+ds126+x127, 0.00125))
1010        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127, 0.0035))
```

```
1011        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+y128, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+y128, 0.00125))
1012        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128, 0.0035))
1013        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128+x129, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128+x129, 0.00125))
1014        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128+ds129, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128+ds129, 0.0035))
1015        s.Line(point1=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128+ds129+y130, 0.0),
            ↪  point2=(set12+ds121+ds122+ds123+ds124+ds125+ds126+ds127+ds128+ds129+y130, 0.00125))
1016        s.Line(point1=(set13, 0.0), point2=(set13, 0.0035))
1017    #Bottom Row, Dspace 31-40
1018        s.Line(point1=(set13+x131, 0.0), point2=(set13+x131, 0.00125))
1019        s.Line(point1=(set13+ds131, 0.0), point2=(set13+ds131, 0.0035))
1020        s.Line(point1=(set13+ds131+y132, 0), point2=(set13+ds131+y132, 0.00125))
1021        s.Line(point1=(set13+ds131+ds132, 0.0), point2=(set13+ds131+ds132, 0.0035))
1022        s.Line(point1=(set13+ds131+ds132+x133, 0.0), point2=(set13+ds131+ds132+x133, 0.00125))
1023        s.Line(point1=(set13+ds131+ds132+ds133, 0.0), point2=(set13+ds131+ds132+ds133, 0.0035))
1024        s.Line(point1=(set13+ds131+ds132+ds133+y134, 0.0), point2=(set13+ds131+ds132+ds133+y134,
            ↪  0.00125))
1025        s.Line(point1=(set13+ds131+ds132+ds133+ds134, 0.0), point2=(set13+ds131+ds132+ds133+ds134,
            ↪  0.0035))
1026        s.Line(point1=(set13+ds131+ds132+ds133+ds134+x135, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+x135, 0.00125))
1027        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135, 0.0035))
1028        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+y136, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+y136, 0.00125))
1029        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+ds136, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+ds136, 0.0035))
1030        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+ds136+x137, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+ds136+x137, 0.00125))
1031        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137, 0.0035))
1032        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+y138, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+y138, 0.00125))
1033        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138, 0.0035))
1034        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138+x139, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138+x139, 0.00125))
```

```
1035        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138+ds139, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138+ds139, 0.0035))
1036        s.Line(point1=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138+ds139+y140, 0.0),
            ↪  point2=(set13+ds131+ds132+ds133+ds134+ds135+ds136+ds137+ds138+ds139+y140, 0.00125))
1037        s.Line(point1=(set14, 0.0), point2=(set14, 0.0035))
1038    #Bottom Row, Dspace 41-50
1039        s.Line(point1=(set14+x141, 0.0), point2=(set14+x141, 0.00125))
1040        s.Line(point1=(set14+ds141, 0.0), point2=(set14+ds141, 0.0035))
1041        s.Line(point1=(set14+ds141+y142, 0), point2=(set14+ds141+y142, 0.00125))
1042        s.Line(point1=(set14+ds141+ds142, 0.0), point2=(set14+ds141+ds142, 0.0035))
1043        s.Line(point1=(set14+ds141+ds142+x143, 0.0), point2=(set14+ds141+ds142+x143, 0.00125))
1044        s.Line(point1=(set14+ds141+ds142+ds143, 0.0), point2=(set14+ds141+ds142+ds143, 0.0035))
1045        s.Line(point1=(set14+ds141+ds142+ds143+y144, 0.0), point2=(set14+ds141+ds142+ds143+y144,
            ↪  0.00125))
1046        s.Line(point1=(set14+ds141+ds142+ds143+ds144, 0.0), point2=(set14+ds141+ds142+ds143+ds144,
            ↪  0.0035))
1047        s.Line(point1=(set14+ds141+ds142+ds143+ds144+x145, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+x145, 0.00125))
1048        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145, 0.0035))
1049        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+y146, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+y146, 0.00125))
1050        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+ds146, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+ds146, 0.0035))
1051        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+ds146+x147, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+ds146+x147, 0.00125))
1052        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147, 0.0035))
1053        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+y148, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+y148, 0.00125))
1054        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148, 0.0035))
1055        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148+x149, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148+x149, 0.00125))
1056        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148+ds149, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148+ds149, 0.0035))
1057        s.Line(point1=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148+ds149+y150, 0.0),
            ↪  point2=(set14+ds141+ds142+ds143+ds144+ds145+ds146+ds147+ds148+ds149+y150, 0.00125))
1058        s.Line(point1=(set15, 0.0), point2=(set15, 0.0035))
1059    #Bottom Row, Dspace 51-60
1060        s.Line(point1=(set15+x151, 0.0), point2=(set15+x151, 0.00125))
```

```
        s.Line(point1=(set15+ds151, 0.0), point2=(set15+ds151, 0.0035))
        s.Line(point1=(set15+ds151+y152, 0), point2=(set15+ds151+y152, 0.00125))
        s.Line(point1=(set15+ds151+ds152, 0.0), point2=(set15+ds151+ds152, 0.0035))
        s.Line(point1=(set15+ds151+ds152+x153, 0.0), point2=(set15+ds151+ds152+x153, 0.00125))
        s.Line(point1=(set15+ds151+ds152+ds153, 0.0), point2=(set15+ds151+ds152+ds153, 0.0035))
        s.Line(point1=(set15+ds151+ds152+ds153+y154, 0.0), point2=(set15+ds151+ds152+ds153+y154,
        ↪  0.00125))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154, 0.0), point2=(set15+ds151+ds152+ds153+ds154,
        ↪  0.0035))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+x155, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+x155, 0.00125))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155, 0.0035))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+y156, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+y156, 0.00125))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+ds156, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+ds156, 0.0035))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+ds156+x157, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+ds156+x157, 0.00125))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157, 0.0035))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+y158, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+y158, 0.00125))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158, 0.0035))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158+x159, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158+x159, 0.00125))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158+ds159, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158+ds159, 0.0035))
        s.Line(point1=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158+ds159+y160, 0.0),
        ↪  point2=(set15+ds151+ds152+ds153+ds154+ds155+ds156+ds157+ds158+ds159+y160, 0.00125))
        s.Line(point1=(set16, 0.0), point2=(set16, 0.0035))
    #Bottom Row, Dspace 61{70
        s.Line(point1=(set16+x161, 0.0), point2=(set16+x161, 0.00125))
        s.Line(point1=(set16+ds161, 0.0), point2=(set16+ds161, 0.0035))
        s.Line(point1=(set16+ds161+y162, 0), point2=(set16+ds161+y162, 0.00125))
        s.Line(point1=(set16+ds161+ds162, 0.0), point2=(set16+ds161+ds162, 0.0035))
        s.Line(point1=(set16+ds161+ds162+x163, 0.0), point2=(set16+ds161+ds162+x163, 0.00125))
        s.Line(point1=(set16+ds161+ds162+ds163, 0.0), point2=(set16+ds161+ds162+ds163, 0.0035))
        s.Line(point1=(set16+ds161+ds162+ds163+y164, 0.0), point2=(set16+ds161+ds162+ds163+y164,
        ↪  0.00125))
```

```
1088        s.Line(point1=(set16+ds161+ds162+ds163+ds164, 0.0), point2=(set16+ds161+ds162+ds163+ds164,
            ↪  0.0035))
1089        s.Line(point1=(set16+ds161+ds162+ds163+ds164+x165, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+x165, 0.00125))
1090        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165, 0.0035))
1091        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+y166, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+y166, 0.00125))
1092        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+ds166, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+ds166, 0.0035))
1093        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+ds166+x167, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+ds166+x167, 0.00125))
1094        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167, 0.0035))
1095        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+y168,0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+y168, 0.00125))
1096        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168, 0.0035))
1097        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168+x169, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168+x169, 0.00125))
1098        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168+ds169, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168+ds169, 0.0035))
1099        s.Line(point1=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168+ds169+y170, 0.0),
            ↪  point2=(set16+ds161+ds162+ds163+ds164+ds165+ds166+ds167+ds168+ds169+y170, 0.00125))
1100        s.Line(point1=(set17, 0.0), point2=(set17, 0.0035))
1101    #Bottom Row, Dspace 71-80
1102        s.Line(point1=(set17+x171, 0.0), point2=(set17+x171, 0.00125))
1103        s.Line(point1=(set17+ds171, 0.0), point2=(set17+ds171, 0.0035))
1104        s.Line(point1=(set17+ds171+y172, 0), point2=(set17+ds171+y172, 0.00125))
1105        s.Line(point1=(set17+ds171+ds172, 0.0), point2=(set17+ds171+ds172, 0.0035))
1106        s.Line(point1=(set17+ds171+ds172+x173, 0.0), point2=(set17+ds171+ds172+x173, 0.00125))
1107        s.Line(point1=(set17+ds171+ds172+ds173, 0.0), point2=(set17+ds171+ds172+ds173, 0.0035))
1108        s.Line(point1=(set17+ds171+ds172+ds173+y174, 0.0), point2=(set17+ds171+ds172+ds173+y174,
            ↪  0.00125))
1109        s.Line(point1=(set17+ds171+ds172+ds173+ds174, 0.0), point2=(set17+ds171+ds172+ds173+ds174,
            ↪  0.0035))
1110        s.Line(point1=(set17+ds171+ds172+ds173+ds174+x175, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+x175, 0.00125))
1111        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175,
            ↪  0.0),point2=(set17+ds171+ds172+ds173+ds174+ds175, 0.0035))
```

```
1112        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+y176, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+y176, 0.00125))
1113        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+ds176, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+ds176, 0.0035))
1114        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+ds176+x177, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+ds176+x177, 0.00125))
1115        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177, 0.0035))
1116        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+y178, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+y178, 0.00125))
1117        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178, 0.0035))
1118        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178+x179, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178+x179, 0.00125))
1119        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178+ds179, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178+ds179, 0.0035))
1120        s.Line(point1=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178+ds179+y180, 0.0),
            ↪  point2=(set17+ds171+ds172+ds173+ds174+ds175+ds176+ds177+ds178+ds179+y180, 0.00125))
1121        s.Line(point1=(set18, 0.0), point2=(set18, 0.0035))
1122    #Bottom Row, Dspace 81-90
1123        s.Line(point1=(set18+x181, 0.0), point2=(set18+x181, 0.00125))
1124        s.Line(point1=(set18+ds181, 0.0), point2=(set18+ds181, 0.0035))
1125        s.Line(point1=(set18+ds181+y182, 0.0), point2=(set18+ds181+y182, 0.00125))
1126        s.Line(point1=(set18+ds181+ds182, 0.0), point2=(set18+ds181+ds182, 0.0035))
1127        s.Line(point1=(set18+ds181+ds182+x183, 0.0), point2=(set18+ds181+ds182+x183, 0.00125))
1128        s.Line(point1=(set18+ds181+ds182+ds183, 0.0), point2=(set18+ds181+ds182+ds183, 0.0035))
1129        s.Line(point1=(set18+ds181+ds182+ds183+y184, 0.0), point2=(set18+ds181+ds182+ds183+y184,
            ↪  0.00125))
1130        s.Line(point1=(set18+ds181+ds182+ds183+ds184, 0.0), point2=(set18+ds181+ds182+ds183+ds184,
            ↪  0.0035))
1131        s.Line(point1=(set18+ds181+ds182+ds183+ds184+x185, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+x185, 0.00125))
1132        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185, 0.0035))
1133        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+y186, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+y186, 0.00125))
1134        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+ds186, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+ds186, 0.0035))
1135        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+ds186+x187, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+ds186+x187, 0.00125))
```

```
1136        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187, 0.0035))
1137        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+y188, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+y188, 0.00125))
1138        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188, 0.0035))
1139        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188+x189, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188+x189, 0.00125))
1140        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188+ds189, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188+ds189, 0.0035))
1141        s.Line(point1=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188+ds189+y190, 0.0),
            ↪  point2=(set18+ds181+ds182+ds183+ds184+ds185+ds186+ds187+ds188+ds189+y190, 0.00125))
1142        s.Line(point1=(set19, 0.0), point2=(set19, 0.0035))
1143    #Bottom Row, Dspace 91-100
1144        s.Line(point1=(set19+x191, 0.0), point2=(set19+x191, 0.00125))
1145        s.Line(point1=(set19+ds191, 0.0), point2=(set19+ds191, 0.0035))
1146        s.Line(point1=(set19+ds191+y192, 0), point2=(set19+ds191+y192, 0.00125))
1147        s.Line(point1=(set19+ds191+ds192, 0.0), point2=(set19+ds191+ds192, 0.0035))
1148        s.Line(point1=(set19+ds191+ds192+x193, 0.0), point2=(set19+ds191+ds192+x193, 0.00125))
1149        s.Line(point1=(set19+ds191+ds192+ds193, 0.0), point2=(set19+ds191+ds192+ds193, 0.0035))
1150        s.Line(point1=(set19+ds191+ds192+ds193+y194, 0.0), point2=(set19+ds191+ds192+ds193+y194,
            ↪  0.00125))
1151        s.Line(point1=(set19+ds191+ds192+ds193+ds194, 0.0), point2=(set19+ds191+ds192+ds193+ds194,
            ↪  0.0035))
1152        s.Line(point1=(set19+ds191+ds192+ds193+ds194+x195, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+x195, 0.00125))
1153        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195, 0.0035))
1154        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+y196, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+y196, 0.00125))
1155        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+ds196, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+ds196, 0.0035))
1156        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+ds196+x197, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+ds196+x197, 0.00125))
1157        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197, 0.0035))
1158        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+y198, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+y198, 0.00125))
1159        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198, 0.0035))
```

```
1160        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198+x199, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198+x199, 0.00125))
1161        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198+ds199, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198+ds199, 0.0035))
1162        s.Line(point1=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198+ds199+y200, 0.0),
            ↪  point2=(set19+ds191+ds192+ds193+ds194+ds195+ds196+ds197+ds198+ds199+y200, 0.00125))
1163        s.Line(point1=(set20, 0.0), point2=(set20, 0.0035))
1164    #SPACER SUBSTITUTION
1165        print 'Top Row larger than Bottom Row'
1166        print ('Dspace200 =', ds200)
1167        LengthS=LengthF-(set20)
1168        spacer=DSmean-1*DSstdev
1169        hys=0.84*spacer
1170        LengthS2=LengthS-spacer
1171        LengthS3=LengthS2-spacer
1172        LengthS4=LengthS3-spacer
1173        LengthS5=LengthS4-spacer
1174        LengthS6=LengthS5-spacer
1175        LengthS7=LengthS6-spacer
1176        LengthS8=LengthS7-spacer
1177        #Spacer Remainder work--is the model still biologically valid?
1178        spacerremainder=LengthF-(set20)
1179        NewArea=(ds200+spacerremainder)*(0.0035)
1180        NewRatio=(ds200*0.84)*(1.25E-3)/(NewArea)
1181        if NewRatio < .25:
1182         print 'REJECT MODEL: Currently Biologically Invalid'
1183        if NewRatio >= .25:
1184         print '*MODEL IS NOW VALID; Good for Analysis*'
1185        if LengthS > spacer:
1186          print 'Added FIRST spcer to bottom row'
1187        #creates boundary line for new DSpace in spacer substitution case
1188          s.Line(point1=(set20+spacer, 0.0), point2=(set20+spacer, 0.0035))
1189        #creates hydrox line for new DSpace in spacer substitution case
1190          s.Line(point1=(set20+spacer-hys, 0.0), point2=(set20+spacer-hys, 0.00125))
1191          spacerremainder=LengthF-(set20+spacer)
1192          NewArea=(spacer+spacerremainder)*(0.0035)
1193          NewRatio=(spacer*0.84)*(1.25E-3)/(NewArea)
1194          if NewRatio < .25:
1195           print 'REJECT MODEL: Currently Biologically Invalid'
1196          if NewRatio >= .25:
1197           print '*MODEL IS NOW VALID; Good for Analysis*'
```

```
        if LengthS2 > spacer:
         print 'Added SECOND spacer to bottom row'
        #creates boundary line for new DSpace in SECOND spacer substitution case
         s.Line(point1=(set20+spacer+spacer, 0.0), point2=(set20+spacer+spacer, 0.0035))
        #creates hydrox line for new DSpace in SECOND spacer substitution case
         s.Line(point1=(set20+spacer+hys, 0.0), point2=(set20+spacer+hys, 0.00125))
         spacerremainder=LengthF-(set20+spacer+spacer)
         NewArea=(spacer+spacerremainder)*(0.0035)
         NewRatio=(spacer*0.84)*(1.25E-3)/(NewArea)
         if NewRatio < .25:
          print 'REJECT MODEL: Currently Biologically Invalid'
         if NewRatio >= .25357:
          print '*MODEL IS NOW VALID; Good for Analysis*'
        if LengthS3 > spacer:
         print 'Added THIRD spacer to bottom row'
        #creates boundary line for new DSpace in THIRD spacer substitution case
         s.Line(point1=(set20+spacer+spacer+spacer, 0.0), point2=(set20+spacer+spacer+spacer, 0.0035))
        #creates hydrox line for new DSpace in THIRD spacer substitution case
         s.Line(point1=(set20+spacer+spacer+spacer-hys, 0.0), point2=(set20+spacer+spacer+spacer-hys,
         ↪  0.00125))
         spacerremainder=LengthF-(set20+spacer+spacer+spacer)
        if LengthS4 > spacer:
         print 'Added FOURTH spacer to bottom row'
        #creates boundary line for new DSpace in FOURTH spacer substitution case
        #creates hydrox line for new DSpace in FOURTH spacer substitution case
         s.Line(point1=(set20+spacer+spacer+spacer+hys, 0.0), point2=(set20+spacer+spacer+spacer+hys,
         ↪  0.00125))
         spacerremainder=LengthF-(set20+spacer+spacer+spacer+spacer)
        if LengthS5 > spacer:
         print 'Added FIFTH spacer to bottom row'
        #creates boundary line for new DSpace in FIFTH spacer substitution case
         s.Line(point1=(set20+spacer+spacer+spacer+spacer, 0.0),
         ↪  point2=(set20+spacer+spacer+spacer+spacer, 0.0035))
        #creates hydrox line for new DSpace in FIFTH spacer substitution case
         s.Line(point1=(set20+spacer+spacer+spacer+spacer+spacer-hys, 0.0),
         ↪  point2=(set20+spacer+spacer+spacer+spacer+spacer-hys, 0.00125))
         spacerremainder=LengthF-(set20+spacer+spacer+spacer+spacer+spacer)
         if NewRatio >= .25:
        if LengthS6 > spacer:
         print 'Added SIXTH spacer to bottom row'
        #creates boundary line for new DSpace in SIXTH spacer substitution case
```

```
1235          s.Line(point1=(set20+spacer+spacer+spacer+spacer+spacer, 0.0),
              ↪  point2=(set20+spacer+spacer+spacer+spacer+spacer, 0.0035))
1236         #creates hydrox line for new DSpace in SIXTH spacer substitution case
1237          s.Line(point1=(set20+spacer+spacer+spacer+spacer+spacer+hys, 0.0),
              ↪  point2=(set20+spacer+spacer+spacer+spacer+spacer+hys, 0.00125))
1238          spacerremainder=LengthF-(set20+spacer+spacer+spacer+spacer+spacer+spacer)
1239         if LengthS7 > spacer:
1240          print 'Added SEVENTH spacer to bottom row'
1241         #creates boundary line for new DSpace in SEVENTH spacer substitution case
1242          s.Line(point1=(set20+spacer+spacer+spacer+spacer+spacer+spacer, 0.0),
              ↪  point2=(set20+spacer+spacer+spacer+spacer+spacer+spacer, 0.0035))
1243         #creates hydrox line for new DSpace in SEVENTH spacer substitution case
1244          s.Line(point1=(set20+spacer+spacer+spacer+spacer+spacer+spacer+spacer-hys, 0.0),
              ↪  point2=(set20+spacer+spacer+spacer+spacer+spacer+spacer+spacer-hys, 0.00125))
1245          spacerremainder=LengthF-(set20+spacer+spacer+spacer+spacer+spacer+spacer+spacer)
1246         if LengthS8 > spacer:
1247          print 'Added EIGTH spacer to bottom row NO WAY IS THIS FOR REAL?!'
1248         #creates boundary line for new DSpace in EIGTH spacer substitution case
1249          s.Line(point1=(set20+spacer+spacer+spacer+spacer+spacer+spacer+spacer, 0.0),
              ↪  point2=(set20+spacer+spacer+spacer+spacer+spacer+spacer+spacer, 0.0035))
1250         #creates hydrox line for new DSpace in EIGTH spacer substitution case
1251          s.Line(point1=(set20+spacer+spacer+spacer+spacer+spacer+spacer+spacer+hys, 0.0),
              ↪  point2=(set20+spacer+spacer+spacer+spacer+spacer+spacer+spacer+hys, 0.00125))
1252          spacerremainder=LengthF-(set20+spacer+spacer+spacer+spacer+spacer+spacer+spacer+spacer)
1253         f = p.faces
1254         pickedFaces = f.getSequenceFromMask(mask=('[#1 ]', ), )
1255         e, d1 = p.edges, p.datums
1256         p.PartitionFaceBySketch(faces=pickedFaces, sketch=s)
1257         s.unsetPrimaryObject()
1258    #SET CREATION
1259         faces = f.getSequenceFromMask(mask=('[#7f7b5 ]', ), )
1260         p.Set(faces=faces, name='COLLAGEN SET')
1261         faces = f.getSequenceFromMask(mask=('[#84a ]', ), )
1262         p.Set(faces=faces, name='HYDROXYAPATITE SET')
1263    #MATERIAL CREATION
1264         mdb.models['Model-1'].Material(name='COLLAGEN')
1265         mdb.models['Model-1'].materials['COLLAGEN'].Depvar(n=3)
1266         mdb.models['Model-1'].materials['COLLAGEN'].UserMaterial(mechanicalConstants=(
1267             0.003, 0.006, 0.004, 0.2, 0.2, 0.2))
1268         mdb.models['Model-1'].Material(name='HYDROXYAPATITE')
1269         mdb.models['Model-1'].materials['HYDROXYAPATITE'].Elastic(table=((0.1, 0.28),
```

```
            ))
        mdb.models['Model-1'].HomogeneousSolidSection(name='COLLAGEN SECTION',
            material='COLLAGEN', thickness=None)
        mdb.models['Model-1'].HomogeneousSolidSection(name='HYDROXYAPATITE SECTION',
            material='HYDROXYAPATITE', thickness=None)
#SECTION ASSINGMENT
#       region = p.sets['COLLAGEN SET']
#       p.SectionAssignment(region=region, sectionName='COLLAGEN SECTION', offset=0.0,
#           offsetType=MIDDLE_SURFACE, offsetField='',
#           thicknessAssignment=FROM_SECTION)
#       region = p.sets['HYDROXYAPATITE SET']
#       p.SectionAssignment(region=region, sectionName='HYDROXYAPATITE SECTION',
#           offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
#INSTANCE SET AND XSYM MAKER
        session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=OFF, bcs=OFF,
            predefinedFields=OFF, connectors=OFF)
        a = mdb.models['Model-1'].rootAssembly
        a.DatumCsysByDefault(CARTESIAN)
        a.Instance(name='Composite Bone-1', part=p, dependent=ON)
        session.viewports['Viewport: 1'].assemblyDisplay.setValues(
            adaptiveMeshConstraints=ON)
        mdb.models['Model-1'].StaticStep(name='APPLY LOAD', previous='Initial',
            timePeriod=0.05, maxNumInc=100000, initialInc=0.05, minInc=1e-10,
            maxInc=0.05)
            step='APPLY LOAD')
        session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
            predefinedFields=ON, connectors=ON, adaptiveMeshConstraints=OFF)
        mdb.models['Model-1'].PeriodicAmplitude(name='SINUSOIDAL', timeSpan=TOTAL,
            frequency=6.28319, start=0.0, a_0=0.6875, data=((0.0, 0.3125), ))
        session.viewports['Viewport: 1'].view.setValues(nearPlane=12.8063,
            farPlane=12.8435, width=0.201952, height=0.110266, viewOffsetX=3.20047,
            viewOffsetY=-0.00229728)
        session.viewports['Viewport: 1'].view.setValues(nearPlane=12.8134,
            farPlane=12.8364, width=0.1195, height=0.065247, viewOffsetX=3.20271,
            viewOffsetY=-0.00322843)
        s1 = a.instances['Composite Bone-1'].edges
        side1Edges1 = s1.getSequenceFromMask(mask=(
            '[#0:58 #4000000 #8000 #20000 #80000240 ]', ), )
        region = regionToolset.Region(side1Edges=side1Edges1)
        mdb.models['Model-1'].Pressure(name='PRESSURE', createStepName='APPLY LOAD',
            region=region, distributionType=UNIFORM, field='', magnitude=-3.36e-06,
```

```
            amplitude='SINUSOIDAL')
        session.viewports['Viewport: 1'].view.fitView()
        session.viewports['Viewport: 1'].view.setValues(nearPlane=12.4455,
            farPlane=13.2043, width=3.94297, height=2.15286, viewOffsetX=-1.23672,
            viewOffsetY=-0.023348)
        session.viewports['Viewport: 1'].view.setValues(nearPlane=12.8161,
            farPlane=12.8338, width=0.096555, height=0.0527191,
            viewOffsetX=-3.20778, viewOffsetY=0.000928941)
        e1 = a.instances['Composite Bone-1'].edges
        edges1 = e1.getSequenceFromMask(mask=('[#42008020 #80004 ]', ), )
        region = regionToolset.Region(edges=edges1)
        mdb.models['Model-1'].XsymmBC(name='XSYM', createStepName='APPLY LOAD',
            region=region)
#LengthF=Length2 (ie. The Bottom Row is Larger; Top Row Has Spacer)
    elif LengthF==Length2:
#Top Row, Dspace 1-10
#Top Row, Dspace 11-20

 #Top Row, Dspace 31-40

#Top Row, Dspace 41-50
        s.Line(point1=(set10, 0.007), point2=(set10, 0.0035))
        print 'Bottom Row larger than Top Row'
        print ('Dspace100 =', ds100)
        LengthS=LengthF-(set10)
        #Spacer Remainder--is the model still biologically valid?
        spacerremainder=LengthF-(set10)
        NewArea=(ds100+spacerremainder)*(0.0035)
        NewRatio=(ds100*0.84)*(1.25E-3)/(NewArea)
         print 'Added FIRST spacer to top row'
         s.Line(point1=(set10+spacer, 0.007), point2=(set10+spacer, 0.0035))
         s.Line(point1=(set10+hys, 0.007), point2=(set10+hys, 0.00575))
         spacerremainder=LengthF-(set10+spacer)
         print 'Added SECOND spacer to top row'
         s.Line(point1=(set10+spacer+spacer, 0.007), point2=(set10+spacer+spacer, 0.0035))
         s.Line(point1=(set10+spacer+spacer-hys, 0.007), point2=(set10+spacer+spacer-hys, 0.00575))
         spacerremainder=LengthF-(set10+spacer+spacer)
         print 'Added THIRD spacer to top row'
         s.Line(point1=(set10+spacer+spacer+spacer, 0.007), point2=(set10+spacer+spacer+spacer, 0.0035))
         s.Line(point1=(set10+spacer+spacer+hys, 0.007), point2=(set10+spacer+spacer+hys, 0.00575))
         spacerremainder=LengthF-(set10+spacer+spacer+spacer)
```

```
1352          print 'Added FOURTH spacer to top row'
1353          s.Line(point1=(set10+spacer+spacer+spacer+spacer, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer, 0.0035))
1354          s.Line(point1=(set10+spacer+spacer+spacer+spacer-hys, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer-hys, 0.00575))
1355          spacerremainder=LengthF-(set10+spacer+spacer+spacer+spacer)
1356          print 'Added FIFTH spacer to top row'
1357          s.Line(point1=(set10+spacer+spacer+spacer+spacer+spacer, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer+spacer, 0.0035))
1358          s.Line(point1=(set10+spacer+spacer+spacer+spacer+hys, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer+hys, 0.00575))
1359          spacerremainder=LengthF-(set10+spacer+spacer+spacer+spacer+spacer)
1360          print 'Added SIXTH spacer to top row'
1361          s.Line(point1=(set10+spacer+spacer+spacer+spacer+spacer+spacer, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer+spacer+spacer, 0.0035))
1362          s.Line(point1=(set10+spacer+spacer+spacer+spacer+spacer+spacer-hys, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer+spacer+spacer-hys, 0.00575))
1363          spacerremainder=LengthF-(set10+spacer+spacer+spacer+spacer+spacer+spacer)
1364          print 'Added SEVENTH spacer to top row'
1365          s.Line(point1=(set10+spacer+spacer+spacer+spacer+spacer+spacer+spacer, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer+spacer+spacer+spacer, 0.0035))
1366          s.Line(point1=(set10+spacer+spacer+spacer+spacer+spacer+spacer+hys, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer+spacer+spacer+hys, 0.00575))
1367          spacerremainder=LengthF-(set10+spacer+spacer+spacer+spacer+spacer+spacer+spacer)
1368          print 'Added EIGTH spacer to top row AGAINST ALL ODDS WHY SO MANY SPACERS GOODNESS'
1369          s.Line(point1=(set10+spacer+spacer+spacer+spacer+spacer+spacer+spacer+spacer, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer+spacer+spacer+spacer+spacer, 0.0035))
1370          s.Line(point1=(set10+spacer+spacer+spacer+spacer+spacer+spacer+spacer+spacer-hys, 0.007),
              ↪  point2=(set10+spacer+spacer+spacer+spacer+spacer+spacer+spacer+spacer-hys, 0.00575))
1371          spacerremainder=LengthF-(set10+spacer+spacer+spacer+spacer+spacer+spacer+spacer+spacer)
1372        faces = f.getSequenceFromMask(mask=(
1373              '[#dbe7dbd6 #dbe7dbe7:3 #af97dbe7 #bd7ebebf #bd7ebd7e:7 #f67ebd7e #e7dbf3f6',
1374              ' #e7dbe7db:8 #7dabe7db #defebf ]', ), )
1375      session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=OFF, bcs=OFF,
1376          predefinedFields=OFF, connectors=OFF)
1377      a = mdb.models['Model-1'].rootAssembly
1378      a.DatumCsysByDefault(CARTESIAN)
1379      p = mdb.models['Model-1'].parts['Composite Bone']
1380      a.Instance(name='Composite Bone-1', part=p, dependent=ON)
1381      session.viewports['Viewport: 1'].assemblyDisplay.setValues(
1382          adaptiveMeshConstraints=ON)
```

```
    mdb.models['Model-1'].StaticStep(name='APPLY LOAD', previous='Initial',
        timePeriod=0.05, maxNumInc=100000, initialInc=0.05, minInc=1e-10,
        maxInc=0.05)
        step='APPLY LOAD')
    session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
        predefinedFields=ON, connectors=ON, adaptiveMeshConstraints=OFF)
    mdb.models['Model-1'].PeriodicAmplitude(name='SINUSOIDAL', timeSpan=TOTAL,
        frequency=6.28319, start=0.0, a_0=0.6875, data=((0.0, 0.3125), ))
    session.viewports['Viewport: 1'].view.setValues(nearPlane=12.8063,
        farPlane=12.8435, width=0.201952, height=0.110266, viewOffsetX=3.20047,
        viewOffsetY=-0.00229728)
    session.viewports['Viewport: 1'].view.setValues(nearPlane=12.8134,
        farPlane=12.8364, width=0.1195, height=0.065247, viewOffsetX=3.20271,
        viewOffsetY=-0.00322843)
    s1 = a.instances['Composite Bone-1'].edges
    side1Edges1 = s1.getSequenceFromMask(mask=(
        '[#0:58 #4000000 #8000 #20000 #80000240 ]', ), )
    region = regionToolset.Region(side1Edges=side1Edges1)
    mdb.models['Model-1'].Pressure(name='PRESSURE', createStepName='APPLY LOAD',
        region=region, distributionType=UNIFORM, field='', magnitude=-3.36e-06,
        amplitude='SINUSOIDAL')
    session.viewports['Viewport: 1'].view.fitView()
    session.viewports['Viewport: 1'].view.setValues(nearPlane=12.4455,
        farPlane=13.2043, width=3.94297, height=2.15286, viewOffsetX=-1.23672,
        viewOffsetY=-0.023348)
    session.viewports['Viewport: 1'].view.setValues(nearPlane=12.8161,
        farPlane=12.8338, width=0.096555, height=0.0527191,
        viewOffsetX=-3.20778, viewOffsetY=0.000928941)
    e1 = a.instances['Composite Bone-1'].edges
    edges1 = e1.getSequenceFromMask(mask=('[#42008020 #80004 ]', ), )
    region = regionToolset.Region(edges=edges1)
    mdb.models['Model-1'].XsymmBC(name='XSYM', createStepName='APPLY LOAD',
        region=region)
TWOXONEHUNDREDFINAL()
```

E.4 Node Randomization Script

```
# Script for randomly selecting node based crosslink locations
import random
```

```
import fileinput

# Create 3 lists, one for each instance surface node set
TopNodeSetMineral = []
TopNodeSetSmall = []
BottomNodeSet = []
TotalCrosslinks = 15

substrBottom = "nset=XlinkNodeBottom,".lower()
substrTop = "nset=XlinkNodeTop, instance=Mineral-1".lower()
substrTopS = "nset=XlinkNodeTop, instance=\"Collagen Small-1".lower()

pull_next_line = False

with open("AddedBC2Test.inp", "r+") as fp:

    for line in fp:
        # See if we've hit the termination condition
        # if so, unset the flag and continue to the next loop iteration
        if line.startswith("*Elset"):
            pull_next_line = False
            continue

        # See if we've hit the starting condition
        # if so, set the flag and continue to the next loop iteration
        if line.lower().find(substrTop) != -1:
            pull_next_line = True
            continue

        # If we're supposed to pull data from the file
        # then append the line to the list
        if pull_next_line:
            TopNodeSetMineral.append(list(map(int, line.strip('\n').split(','))))
            # strip whitespace off the line so we don't end up with duplicated newlines

fp.close()

pull_next_line = False

with open("AddedBC2Test.inp", "r+") as fp:

```

```
    for line in fp:
        # See if we've hit the termination condition
        # if so, unset the flag and continue to the next loop iteration
        if line.startswith("*Nset, nset=XLINKNODETOP, instance=MINERAL-1"):
            pull_next_line = False
            continue

        # See if we've hit the starting condition
        # if so, set the flag and continue to the next loop iteration
        if line.lower().find(substrTopS) != -1:
            pull_next_line = True
            continue

        # If we're supposed to pull data from the file
        # then append the line to the list
        if pull_next_line:
            TopNodeSetSmall.append(list(map(int, line.strip('\n').split(','))))
            # strip whitespace off the line so we don't end up with duplicated newlines

fp.close()

pull_next_line = False

with open("AddedBC2Test.inp", "r+") as fp:

    for line in fp:
        # See if we've hit the termination condition
        # if so, unset the flag and continue to the next loop iteration
        if line.startswith("*Nset, nset=XLINKNODETOP, instance=\"Collagen Small-1"):
            pull_next_line = False
            continue

        # See if we've hit the starting condition
        # if so, set the flag and continue to the next loop iteration
        if line.lower().find(substrBottom) != -1:
            pull_next_line = True
            continue

        # If we're supposed to pull data from the file
        # then append the line to the list
        if pull_next_line:
```

```
85              BottomNodeSet.append(list(map(int, line.strip('\n').split(','))))
86              # strip whitespace off the line so we don't end up with duplicated newlines
87
88
89  fp.close()
90
91  # Need to flatten the list of lists into list of ints
92  TopNodeSetMineral = [item for sublist in TopNodeSetMineral for item in sublist]
93  TopNodeSetSmall = [item for sublist in TopNodeSetSmall for item in sublist]
94  BottomNodeSet = [item for sublist in BottomNodeSet for item in sublist]
95
96  # Randomly pick how many xlinks are on top and bottom rows
97  NumXlinkBottom = random.randrange(0, TotalCrosslinks)
98  NumXlinkTopTotal = TotalCrosslinks - NumXlinkBottom
99  # Assign remaining xlinks to top proportionate to percent length
100 NumXlinkTopMineral = int(0.85 * NumXlinkTopTotal)
101 NumXlinkTopSmall = TotalCrosslinks - NumXlinkBottom - NumXlinkTopMineral
102
103
104 # Test Arrays
105 # TopNodeSetMineral = [1, 2, 3, 5, 12, 300, 43, 32, 500, 590, 1222, 12, 13, 14, 15, 16, 17, 18, 19, 20,
    ↪  1222, 12, 13, 14, 15, 16, 17, 18, 19, 20]
106 # TopNodeSetSmall = [1, 2, 3, 5, 12, 300, 43, 32, 500, 590, 1222, 12, 13, 14, 15, 16, 17, 18, 19, 20,
    ↪  1222, 12, 13, 14, 15, 16, 17, 18, 19, 20]
107 # BottomNodeSet = [1, 3, 4, 6, 89, 399, 455, 543, 700, 1222, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1222,
    ↪  12, 13, 14, 15, 16, 17, 18, 19, 20]
108
109 # Select nodes randomly from these sets without duplicates
110 XlinkTopNodesMin = random.sample(TopNodeSetMineral, NumXlinkTopMineral)
111 XlinkTopNodesSmall = random.sample(TopNodeSetSmall, NumXlinkTopSmall)
112 XlinkBottomNodes = random.sample(BottomNodeSet, NumXlinkBottom)
113
114 XlinkTopNodesMin.sort()
115 XlinkTopNodesSmall.sort()
116 XlinkBottomNodes.sort()
117
118 print("# Mineral Nodes:", NumXlinkTopMineral)
119 print("# Small Nodes:", NumXlinkTopSmall)
120 print("# Bottom Nodes:", NumXlinkBottom)
121 print("The top mineral nodes are:", XlinkTopNodesMin, ", the small nodes are:", XlinkTopNodesSmall)
122 print("and the bottom nodes are:", XlinkBottomNodes)
```

```
123
124 # Write-To-File Section
125 # Writes node sets to VERY specific sections by brute force
126 # Only run these once
127
128 write_next_line = False
129 numbernode = 0
130
131 file_name = 'AddedBC2Test.inp'
132 if len(XlinkTopNodesMin) != 0:
133     with fileinput.FileInput(file_name, inplace=True, backup='.bak') as f:
134         for line in f:
135             if line.startswith("*Elset, elset=MODELLEFT, instance=MINERAL-1"):
136                 write_next_line = True
137                 print(line, end='')
138                 continue
139             if write_next_line:
140                 print(line + "*Nset, nset=XlinkTopNodesMin, instance=MINERAL-1")
141                 write_next_line = False
142                 for node in XlinkTopNodesMin:
143                     line = str(node)
144                     if numbernode == 0:
145                         print(" " + line + ",", sep=" ", end=" ")
146                         numbernode += 1
147                         continue
148                     if numbernode != len(XlinkTopNodesMin) - 1:
149                         print(line + ",", sep=" ", end=" ")
150                         numbernode += 1
151                     else:
152                         print(line)
153
154             else:
155                 print(line, end='')
156
157
158 numbernode = 0
159 if len(XlinkTopNodesSmall) != 0:
160     with fileinput.FileInput(file_name, inplace=True, backup='.bak') as f:
161         for line in f:
162             if line.startswith("*Elset, elset=MODELLEFT, instance=MINERAL-1"):
163                 write_next_line = True
```

```
                print(line, end='')
                continue
            if write_next_line:
                print(line + "*Nset, nset=XlinkTopNodesSmall, instance=\"Collagen Small-1\"")
                write_next_line = False
                for node in XlinkTopNodesSmall:
                    line = str(node)
                    if numbernode == 0:
                        print(" " + line + ",", sep=" ", end=" ")
                        numbernode += 1
                        continue
                    if numbernode != len(XlinkTopNodesSmall) - 1:
                        print(line + ",", sep=" ", end=" ")
                        numbernode += 1
                    else:
                        print(line)

            else:
                print(line, end='')

numbernode = 0
if len(XlinkBottomNodes) != 0:
    with fileinput.FileInput(file_name, inplace=True, backup='.bak') as f:
        for line in f:
            if line.startswith("*Elset, elset=MODELLEFT, instance=MINERAL-1"):
                write_next_line = True
                print(line, end='')
                continue
            if write_next_line:
                print(line + "*Nset, nset=XlinkBottomNodes, instance=\"Collagen Full-1\"")
                write_next_line = False
                for node in XlinkBottomNodes:
                    line = str(node)
                    if numbernode == 0:
                        print(" " + line + ",", sep=" ", end=" ")
                        numbernode += 1
                        continue
                    if numbernode != len(XlinkBottomNodes) - 1:
                        print(line + ",", sep=" ", end=" ")
                        numbernode += 1
                    else:
```

```
                        print(line)
            else:
                print(line, end='')
```

E.5 Python Node Retrieval Script

E.5.1 Variable Dashpot Scheme

```
# Python script to write displacements for desired nodes into separate
# files. Each file contains the displacements in the x-direction from
# the last increment of every step.

# Import odb commands
from odbAccess import *
from abaqusConstants import *

# Import serialization commands
import pickle

# Import OS commands
import os

# Open odb file
import fnmatch

for file in os.listdir('.'):
    if fnmatch.fnmatch(file, '*.odb'):
        odb_file = file

odb = openOdb(odb_file)
#odb = openOdb('/home/mimendoz/NormalDSpacing.odb')

# Create folder for node displacement output (may not work on Windows)
if not os.path.exists('./node_displacement'):
        os.mkdir('./node_displacement')

# Create array with all steps and count the number of steps.
step_list = odb.steps.keys()
```

```
numSteps = len(step_list)
last_step = odb.steps.keys()[-1]


#node list for C2207
#node_list=[1207,92863,94680,95339,658613,663054,670800,670924]

#node list for C1809
node_list=[94030,1194,94991,1197,94143,91782,91781,647635]

# Output displacements for each node
for node_num in node_list:

    # Clear/create displacements array
    displacements = []

    # Write displacements from the last frame of every step to
    # separate files for each node
    for step in step_list:

        last_frame = odb.steps[step].frames[-1]

        # Add the value to displacement array
        displacements.append(last_frame.fieldOutputs['U'].values[- 1].data[0])

        # Wait to write data to file until last step
        if step == last_step:

            file_name = './node_displacement/node_' + str(node_num) + '_disp.txt'
            fid = open(file_name, 'wb')

            for index in range(0, len(displacements)):
                print>>fid, displacements[index], ","

            print 'Node ', node_num, ' complete'

            fid.close

odb.close
```

E.5.2 IMF Scheme

```
# Python script to write displacements for desired nodes into separate
# files. Each file contains the displacements in the x-direction from
# the last increment of every step.

# Import odb commands
from odbAccess import *
from abaqusConstants import *

# Import serialization commands
import pickle

# Import OS commands
import os

# Open odb file
import fnmatch

for file in os.listdir('.'):
    if fnmatch.fnmatch(file, '*.odb'):
        odb_file = file

odb = openOdb(odb_file)

# Create folder for node displacement output (may not work on Windows)
if not os.path.exists('./node_displacement_TGM'):
        os.mkdir('./node_displacement_TGM')

# Create array with all steps and count the number of steps.
step_list = odb.steps.keys()
numSteps = len(step_list)
last_step = odb.steps.keys()[-1]

#Establish the Terminus node sets as regions to specify in FieldOutput
SetFull = odb.rootAssembly.instances['Collagen Full-1'].nodeSets['TERMINUSFULL']
SetHalf = odb.rootAssembly.instances['Collagen Half-1'].nodeSets['TERMINUSHALF']
SetSmall = odb.rootAssembly.instances['Collagen Small-1'].nodeSets['TERMINUSSMALL']

#Assigns node set info as odboutput data to a variable
```

```
SetFullNodes = SetFull.nodes
SetHalfNodes = SetHalf.nodes
SetSmallNodes = SetSmall.nodes

#How many nodes are in each set/How long should our subsequent loops run?
FullNodesLength = len(SetFullNodes)
HalfNodesLength = len(SetHalfNodes)
SmallNodesLength = len(SetSmallNodes)

#Iterate and write UX displacements to files...need better control flow
for node in range(FullNodesLength):

        SetFullNodesR = SetFullNodes[node]

        displacements = []
        for step in step_list:

                last_frame = odb.steps[step].frames[-1]

                displacements.append(last_frame.fieldOutputs['U'].getSubset(region=SetFullNodesR).values[-1].data[0])

                if step == last_step:
                        file_name = './node_displacement_TGM/Col_Full_Node_' + str(node) + '_disp.txt'
                        fid = open(file_name, 'wb')

                        for index in range(0, len(displacements)):
                                print>>fid, displacements[index]

                        fid.close

for node in range(HalfNodesLength):

        SetHalfNodesR = SetHalfNodes[node]

        displacements = []
        for step in step_list:

                last_frame = odb.steps[step].frames[-1]

                displacements.append(last_frame.fieldOutputs['U'].getSubset(region=SetHalfNodesR).values[-1].data[0])

```

```
                if step == last_step:
                        file_name = './node_displacement_TGM/Col_Half_Node_' + str(node) + '_disp.txt'
                        fid = open(file_name, 'wb')

                        for index in range(0, len(displacements)):
                                print>>fid, displacements[index]

                        fid.close

for node in range(SmallNodesLength):

        SetSmallNodesR = SetSmallNodes[node]

        displacements = []
        for step in step_list:

                last_frame = odb.steps[step].frames[-1]

                displacements.append(last_frame.fieldOutputs['U'].getSubset(region=SetSmallNodesR).values[-1].data[0])

                if step == last_step:
                        file_name = './node_displacement_TGM/Col_Small_Node_' + str(node) +
                        ↪  '_disp.txt'
                        fid = open(file_name, 'wb')

                        for index in range(0, len(displacements)):
                                print>>fid, displacements[index]

                        fid.close

odb.close
```

E.6 Matlab Post Processing Scripts

```
%TangentDelta.m

%% MatLab code to aqcuire data from Abaqus files
%% and determine the tangent delta for each analysis
clear all
```

```
close all

%cd node_displacement_TGM
%% Save data from Abaqus displacement files to column vectors
filename = 'node_1194_disp.txt';
A1 = importdata(filename);

filename = 'node_1197_disp.txt';
A2 = importdata(filename);

filename = 'node_91781_disp.txt';
A3 = importdata(filename);

filename = 'node_91782_disp.txt';
A4 = importdata(filename);

filename = 'node_94030_disp.txt';
A5 = importdata(filename);

filename = 'node_94143_disp.txt';
A6 = importdata(filename);

filename = 'node_94991_disp.txt';
A7 = importdata(filename);

filename = 'node_647635_disp.txt';
A8 = importdata(filename);

%% Combine all node displacement column vectors into a single array
Disp_Data = cat(2,A1,A2,A3,A4,A5,A6,A7,A8);
%Removes the comma delimiter from Disp_Data:
%Disp_Data = regexprep(Disp_Data,'(\d+)(\s),','£1£2');
%Disp_Data = cellfun(@str2num, Disp_Data);


%% Determine an average displacement from all the nodes and save to a
%% single column vector
Disp_Data = transpose(Disp_Data);
Ave_Disp = mean(Disp_Data);
Ave_Disp = transpose(Ave_Disp);

```

```
%% Calculate the average strain behavior based on total length
%Update p == length_F or Total Length of Model
p = 6.7407926;
Data = Ave_Disp/p; % Divide by the periodic length to get strain

%% Remove data from first 10 cycles
Data_10_cycles = Data(201:length(Data));

%% Initialize frequency, time, and initial amplitude
% Select a frequency
%f = 1; % 1 Hz frequency
%f = 2; % 2 Hz frequency
%f = 3; % 3 Hz frequency
%f = 5; % 5 Hz frequency
%f = 7; % 7 Hz frequency
%f = 9; % 9 Hz frequency
f = 12; % 12 Hz frequency
%f = 15; % 15 Hz frequency

%t = 1/(20.*f):1/(20.*f):20/f; % Time for entire data

t = 10/f + 1/(20.*f):1/(20.*f):20/f; % Time for last 10 cycles
t = t(:); % Transpose time to match Data vector

t2 = 0:1/(20.*f):1/f;

initial_amp = 1.35e-4; %From Mendoza Appendix, does not seem to affect Tan D, just R^2

%% Function file that accepts curve parameters as inputs and then outputs
%% fitting error
%Starting = rand(1,3);
Starting = rand(1,2);
options = optimset('Display','iter', 'TolX', 1e-5, 'TolFun', 1e-5);
%options = optimset('Display','iter');
%Estimates = fminsearch(@CurveFit,Starting,options,t,Data,f,initial_amp);
% Curve fit for entire data

[Estimates FunctionValue] = fminsearch(@CurveFit,Starting,options,t,Data_10_cycles,f,initial_amp);
% Curve fit for last 10 cycles

%% Calculate curve fit equation and coefficient of determination
```

```
strain = Estimates(1)*sin(2.*pi.*f.*t - Estimates(2)) + initial_amp;
[r2 rmse] = rsquare(Data_10_cycles,strain); % r^2 value for last 10 cycles

%strain = Estimates(1)*sin(2.*pi.*f*t - Estimates(2)) + initial_amp;
%[r2 rmse] = rsquare(Data,strain); % r^2 value for entire data

%% Normalized stress and strain history for first cycle
norm_stress = sin(2.*pi.*f.*t2);
norm_strain = sin(2.*pi.*f.*t2 - Estimates(2));

%% Plot the fitted curve over the raw data
fig1 = figure;
plot(t,Data_10_cycles,'*') % Plot last 10 cycles
%plot(t,Data,'*') % Plot entire data
hold on
plot(t,strain,'r')
xlabel('Time (seconds)','FontSize',16)
ylabel('Strain (unitless)','FontSize',16)
title('Tangent Delta Calculation','FontSize',16)
str = {'R-squared',num2str(r2),'Tangent Delta',num2str(Estimates(2))};
annotation('textbox',[.7,.12,.2,.15],'String',str);
set(fig1,'Position',[1 540 500 400])
%% Plot normalized stress and strain for 1 cycle on a separate figure
fig2 = figure;
plot(t2,norm_stress,'--r')
hold on
plot(t2,norm_strain,'k')
set(fig2,'Position',[1 1 500 400])

format short e;
disp(Estimates(2))
clipboard('copy',Estimates(2));
```