Thomas Ebner

A Proposed Failure Mechanism for Pulp Fibre-Fibre Joints

Master's Thesis

Graz University of Technology

Institute for Strength of Materials
Head: Univ.-Prof. Dr.-Ing. habil. Katrin Ellermann

Supervisor: Ass.-Prof. Dipl.-Ing. Dr.techn. Manfred H. Ulz

Graz, June 2015

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ______________	______________
Date	Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ______________	______________
Datum	Unterschrift

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Dedicated to my parents, sisters, girlfriend and all my friends who walked with me.

Special thanks to my supervisor Manfred Ulz.

Abstract

In the following study a FEM modelling framework of a pulp fibre-fibre joint was created. With this model it is possible to investigate the bonding area. The Institute for Paper, Pulp and Fibre Technology at Graz University of Technology set up a scientific test to determine the strength of the bond of two pulp fibres in paper. The boundary conditions and the applied force for the modelling framework are derived from this experiment. Three different failure modes in fracture mechanics are observed: Mode 1 (opening), Mode 2 (shear) and Mode 3 (twisting). A parameter study of the peak load at the edges of the fibres is carried out, in order to identify a failure mechanism. The peak stresses are not directly taken from the FEM models as these values are highly discretization-dependent. Instead the peak stresses are estimated from resultant forces and moments in an idealized geometry of the bonding region.

Zusammenfassung

In der vorliegenden Arbeit wurde ein finite Elemente Rechenmodell von einer Faser-Faserverbindung von Holzfasern erstellt, mithilfe dessen es möglich ist, die Spannungsverteilung in der Bruchfläche zu simulieren. Am Institut für Papier-, Zellstoff- und Fasertechnik der Technischen Universiät Graz wurde die Bindekraft von zwei Holzfasern mittels eines Versuches ermittelt. Die Randbedingungen wie Einspannung und Belastung für das Rechenmodell wurden von diesen Versuchsergebnissen abgeleitet. Es wurden drei verschiedene Versagensursachen betrachtet. Mode1 (Abhebung), Mode 2 (Scherung) und Mode 3 (Verdrehung). Um eine Versagenshypothese bei Papierfasern zu finden, wurde eine Parameterstudie durchgeführt. Bei dieser wurden die unterschiedlichen Spannungsspitzen in der Verbindungsfläche bei veränderter Geometrie und Belastung analysiert. Die maximale Spannung wurde nicht direkt am FEM Modell abgelesen, sondern über resultierende Schnittkräfte und Momente berechnet.

Contents

List of Figures

1 Introduction

1.1 Composition of the mechanical structure of wood

In the last years wood got more and more important for the economy, especially in the industrial manufacturing sector e. g. (paper industry). Because of its renewability, wood is very attractive as a material resource. Commonly woody plants are classified into two groups: Softwood and hardwood. Their difference is based on anatomical and chemical features. One major characteristic of wood is that it offers an unusual degree of variability based on the different growth conditions, environmental factors and in the variations in genetic stock. Bodig and Benjamin, 1993

Because of the complex internal structure of wood an anisotropic behaviour is given that refers to the several levels of material organisation. Particular types of anisotropy can be identified within a layer of the woody cell, in the cell wall as a whole, in the entire cell, in an aggregate of cells, and finally in an aggregation of several types of woody tissue. Bodig and Benjamin, 1993

The common wood fibre is hollow and designed as a rectangular or elliptical cross section. The average length of such a wood fibre is about 5 mm. A wood fibre consists of different layers and because of this quality they can be used as a composite. These cell wall layers are composed of different components and amounts of the three basic elements of wood: Cellulose, Hemicellulose and Lignin. In addition, these three basic components influence the characteristics and structure of wood by their contained amount.

1.1.1 Chemical composition of wood

In the following chapters cellulose, hemicellulose, lignin and the extractives will be described.

Cellulose

Cellulose is the primary content of the cell wall and can be found in almost every natural resource. In its chemical structure, cellulose is the simplest of the cell wall components. It consists of a linear polymer of glucose units. As shown in figure 1.1, biosynthetic polymerisation means that the basic polymeric unit of two combined glucose constituents is stretched to almost its maximum dimension. The range of glucose units in a cellulose molecule reaches from a few up to 15000. Bodig and Benjamin, 1993

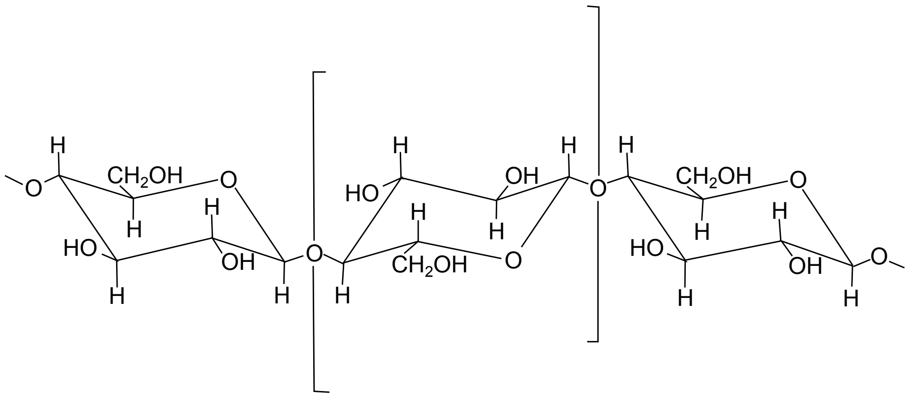


Figure 1.1: Portion of cellulose molecule

Hemicellulose

About 30 percent of a wood cell are made up by hemicellulose. With this percentage it constitutes the second biggest part of the total wood mass. It consists of two carbohydrate polymers, a xylose- and a mannose-containing polysaccharide. Basically hemicellulose is a modified form of cellulose. Hemicellulose is located in the cell walls in the form of individual molecules. In this way they are more closely associated with lignin and in an amorphous state, which is also caused by the presence of many side groups that prohibit a crystallisation of the molecules. Bodig and Benjamin, 1993

Lignin

Lignin is a three-dimensional polymer with apparently no ordered arrangement. It is the third component of the cell wall and it is a phenolic polymer, differing from the carbohydrates in its water repellency. Lignin is built up by the random free-radical polymerisation of three closely located phenolic substances. A characteristic of Lignin is, that it is completely amorphous and under normal conditions begins to soften at temperatures of 165-175 degree centigrade. Bodig and Benjamin, 1993

Extractives

Ectractives have no or just a small impact on the direct mechanical behaviour of wood. They are more likely to increase specific gravity and lower the equilibrium moisture content. Extractives influence the mechanical properties indirectly, they can control durability, colour, odour and taste. Bodig and Benjamin, 1993

1.1.2 Organisation of the cell wall

Cellulose molecules seldomly appear as individual entities. Within the cell wall they are more likely to build discrete bundles called elementary fibrils. As described in section 1.1, they can appear in the form of elliptical or rectangular cross sections of the order of 3.5 x 3.5nm, which is considered as an acceptable value. Each single elementary fibril consists of a parallel array of 50-80 cellulose molecules, which are aligned with the fibril axis. These elementary structural fibrils are often collected into larger units by means of hydrogen bonding between their respective surfaces. Microfibrils are the result of this construction in the dimension of 3.5 x 10nm, as shown in figure 1.2. Figure 1.2 also represents a schematic construction of microfibrils arranged in orderly form in a cell wall layer. One can note the spaces between the microfibrils which are available for deposition of different chemical substances and for the absorption of water. Bodig and Benjamin, 1993

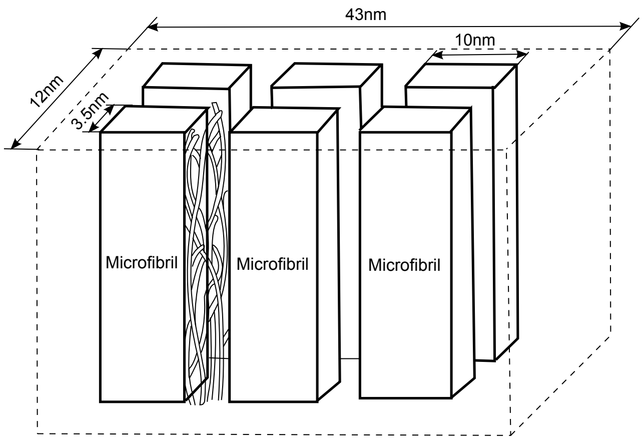


Figure 1.2: Distribution of chemical constituents across the cell wall

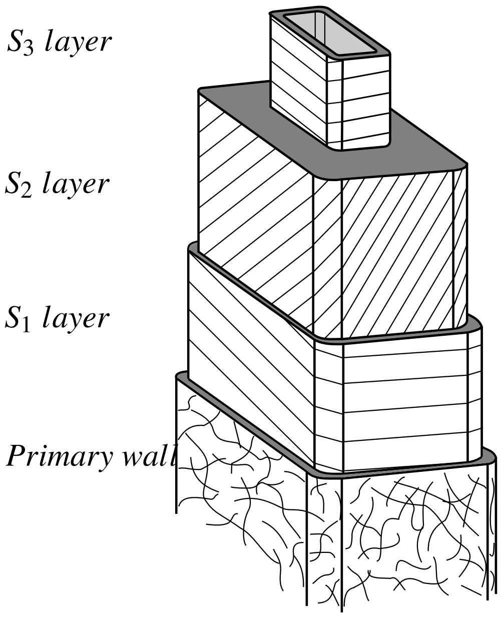


Figure 1.3: The layered structure of a single pulp fibre.

The organisation of cell wall layers is very complex. In common the cell wall of pulp fibres are built of four major layers as shown in figure 1.3. The primary wall is a thin, flexible and extensible layer of the cell wall. The next layer, the S1 layer, is very thin, usually no greater than 0,1 micrometres in thickness. Furthermore, the S2 layer, with a thickness about 2µm, is the thickest of the three layers and tends to dominate the behaviour of the cell. The last one is the S3 layer. All layers are composed of different degrees of cellulose, hemicellulose and lignin, which can be seen in figure 1.4.

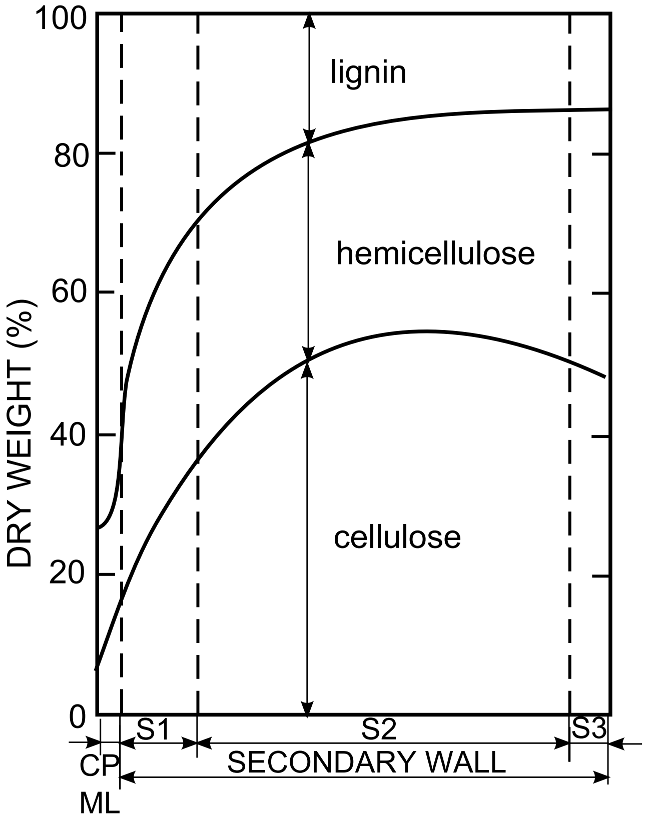


Figure 1.4: Distribution of chemical constituents across the cell wall

1.2 Production of paper

Paper-making is a complex process, which includes diverse manufacturing steps. The production of paper products from wood is dependent on two main technologies. These two fields are the pulp- and the paper technology. The pulp technology focuses on the liberation of fibres fixed in the wood or plant matrix, where the paper technology is the knowledge of how to unify the fibres into a find paper product. Ek, Gellerstedt, and Henriksson, 2009

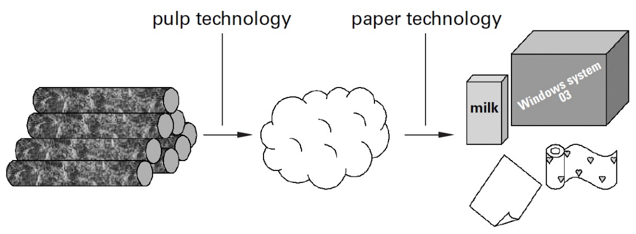


Figure 1.5: The making of paper products from wood includes two main technological fields. Ek, Gellerstedt, and Henriksson, 2009

1.2.1 Pulping Processes

The aim of the pulping process is to liberate the fibres from the wood matrix. This can be done in two different ways, either mechanically or chemically. Each of these methods has advantages and disadvantages. For example a mechanical process makes use of the whole wood material but, on the other hand this process demands a lot of electric energy. The advantage of a modern chemical pulp mill is that there is no need for external energy. The disadvantage is that only approximately half of the wood becomes pulp. The other half gets dissolved. In order to create an economical chemical pulping process, the existence of an efficient recovery system is of importance. Ek, Gellerstedt, and Henriksson, 2009

Mechanical Pulping

To obtain a mechanical pulp the fibres in the wood have to be released. This is done by grinding wood or wood chips. Depending on what mechanical pulping method is used, the pulp yield ranges between 90 and 100 %. The mechanical pulp consists of fibres and a large portion of smaller material called fines. These fines are made up by fragments from the fibre wall and broken fibres. They are very important for the excellent optical properties of mechanical pulps. A higher portion of long fibres guarantees the pulp strength. Figure 1.6 shows how groundwood pulp is produced. Round wood logs are pressed against a rotating cylinder which is made of sandstone.

The logs are fed parallel to the cylinder axis and by this process the fibres are scraped off. Ek, Gellerstedt, and Henriksson, 2009

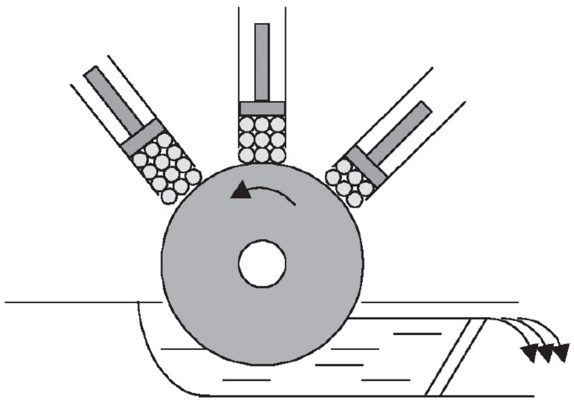


Figure 1.6: A schematic figure of a stone grinder. Ek, Gellerstedt, and Henriksson, 2009

Refiner pulp is another sort of mechanical pulp. In this case, wood chips are fed into the centre of two refining discs, as illustrated in figure 1.7. While one, or even both discs are rotating, the chips are reduced in chips and fibres are abraded off. In this process the discs are grooved. Near the centre the grooves are coarser and towards the perimeter the grooves are getting finer. The pulp is getting finer towards the edge of the disc. Ek, Gellerstedt, and Henriksson, 2009

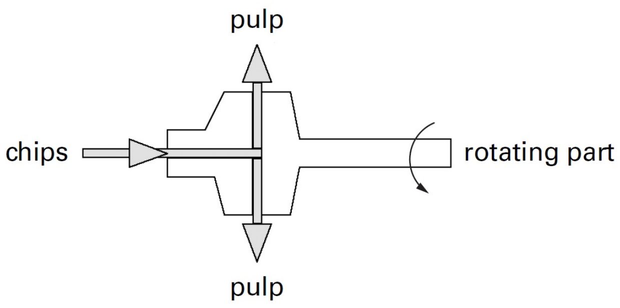


Figure 1.7: A chip refiner with one rotating disc. Ek, Gellerstedt, and Henriksson, 2009

Chemical Pulping

The fibres in woods are held together by lignin. By chemical pulping most of the lignin and the fibres are released. It has to be said that no chemical method is able to remove all of the lignin in the pulping stage. The process of delignification is terminated with still some lignin remaining in the pulp. The most often used method to remove lignin is the Kraft cooking. Sodium hydroxide and sodium sulphite are the used chemicals for this technique. If only sodium hydroxide is used it is called soda cooking. The sulphite cooking process uses sulphurous acid (H2SO3) and bisulphite ions (H2SO3-) to dissolve lignin. By performing chemical pulping better results are reached compared to the mechanical method. The chemical pulp fibres are more flexible and offer good strength properties. Ek, Gellerstedt, and Henriksson, 2009

Bleaching

Some paper products are in need of white paper. To get a better contrast between the paper and the print, a good print quality is required. Another reason is the cleanliness of the paper, which needs the technique of bleaching. Furthermore, bleaching delays the ageing process. Ek, Gellerstedt, and Henriksson, 2009

The bleaching process consists of several steps and by using different chemicals in each step. In the mechanical pulp dithionite and hydrogen peroxide are used. Chemical pulps are delignified with oxygen before entering the actual bleaching plant. Ek, Gellerstedt, and Henriksson, 2009

Papermaking

The process of papermaking requires a paper machine, which consolidates the liberated fibres from the wood to form again a web of paper. First of all, a very dilute slurry of fibres is sprayed on to a moving wire. This slurry can incorporate fillers, retention aid and wet strength additives. The wire is made up of an endless woven wire cloth with a mesh size that allows the water to be drained, but the fibres are retained on the wire. At this certain stage of papermaking the paper web enters the pressing section of the paper machine. Now the water is pressed out by squeezing the paper web between steel rolls. To ensure that a certain dryness of the paper can be given, the paper is dried in the drying section. The web is reeled at the end of

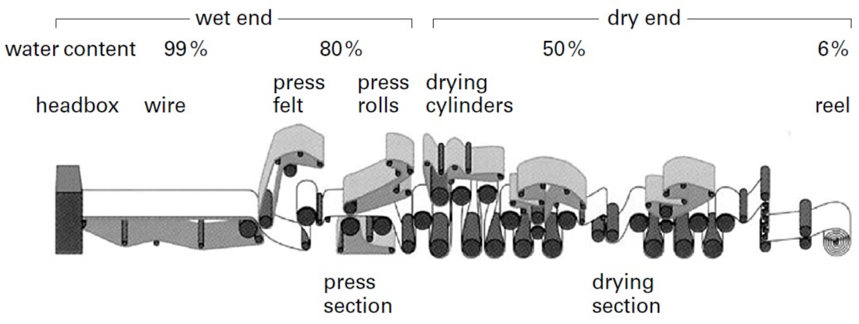


Figure 1.8: A simplified plot of a paper machine. Ek, Gellerstedt, and Henriksson, 2009

the paper machine. In figure 1.8 a simplified plot of a paper machine is given. Ek, Gellerstedt, and Henriksson, 2009

2 Continuum Mechanics

This chapter is written with reference to SIMULIA-DassaultSystémes, 2012

2.1 Deformation

Matter cannot appear or disappear. With this information we can assume that a material particle, initially located at some position $\vec{X}$ in space, will move to a new position $\vec{x}$. The history of the location of a particle can always be written as

$$\vec{x} = \vec{x}(\vec{X}, t) \tag{2.1}$$

$\vec{X}$ is the initial position of the material particle and $\vec{x}$ is the position during the deformation.

Now we consider two neighbouring material particles. If the initial configuration of these two particles is located at $\vec{X}$ and at $\vec{X} + d\vec{X}$, the current configuration must satisfy

$$d\vec{x} = \frac{\partial \vec{x}}{\partial \vec{X}} \cdot d\vec{X} \tag{2.2}$$

and the matrix

$$\mathbf{F} = \frac{\partial \vec{x}}{\partial \vec{X}} \tag{2.3}$$

is called the deformation gradient matrix. So we can write the equation 2.2 as

$$d\vec{x} = \mathbf{F} \cdot d\vec{X} \tag{2.4}$$

The motion of the vicinity of a material point distinguishes between the straining of the material and the rigid body motion. The material behaviour depends on the straining of the material. Now we are looking at an infinitesimal gauge length $d\vec{X}$. The initial and current lengths are measured by

$$dL^2 = d\vec{X}^T \cdot d\vec{X} \text{ and } dl^2 = d\vec{x}^T \cdot d\vec{x} \tag{2.5}$$

The "stretch ratio" of this gauge length is

$$\lambda = \frac{dl}{dL} = \sqrt{\frac{d\vec{x}^T \cdot d\vec{x}}{d\vec{X}^T \cdot d\vec{X}}} \tag{2.6}$$

If there is only a rigid body motion, the stretch ratio has the value one.

Now we use the equation 2.4 to define the current infinitesimal gauge length.

$$d\vec{x}^T \cdot d\vec{x} = d\vec{X}^T \cdot \mathbf{F}^T \cdot \mathbf{F} \cdot d\vec{X} \tag{2.7}$$

$$\lambda^2 = \frac{d\vec{X}^T}{\sqrt{d\vec{X}^T \cdot d\vec{X}}} \cdot \mathbf{F}^T \cdot \mathbf{F} \cdot \frac{d\vec{X}}{\sqrt{d\vec{X}^T \cdot d\vec{X}}} = \vec{N}^T \cdot \mathbf{F}^T \cdot \mathbf{F} \cdot \vec{N} \tag{2.8}$$

$\vec{N}$ is a unit vector in direction of the gauge length $d\vec{x}$. The stretch ratio associated with any direction $\vec{N}$, at any material point defined by $\vec{X}$, is calculated as shown in equation 2.8.

2.2 Strain Measures

The stretch ratio λ as a measure of deformation is already known, but it is an unsatisfactory way of measuring deformation of steel. The elastic modulus of steel is about $200 \cdot 10^3$MPa and the yield stress about 200 MPa. So the deviation from zero only begins in the fourth significant digit. That is why we define strain as a function of the stretch ratio.

$$\varepsilon = f(\lambda) \tag{2.9}$$

$$\varepsilon = f(1) + (\lambda - 1)\frac{df}{d\lambda} + \frac{1}{2!}(\lambda - 1)^2 \frac{d^2 f}{d\lambda^2} + ... \tag{2.10}$$

We must have the following constraints:

$$f(1) = 0$$

$$\frac{df}{d\lambda(1)} = 1$$

$$\frac{df}{d\lambda} > 0 \text{ at } \lambda > 1$$

$f(1) = 0$ means the strain ε has the value zero, if the stretch ratio λ has the value one. $\frac{df}{d\lambda} = 1$ at $\lambda = 1$ is chosen to have the usual definition of strain as the "change in length per unit length" for small strains. The last constraint ensures, that each value of stretch has a unique corresponding value of strain.

Examples of commonly used strain measures:

$$\text{Nominal strain (Biot's strain): } f(\lambda) = \lambda - 1$$

This definition is used for uniaxial testing of stiff specimens.

$$\text{Logarithmic strain: } f(\lambda) = \ln \lambda$$

This strain measure is commonly used in metal plasticity.

$$\text{Green's strain : } f(\lambda) = \frac{1}{2}(\lambda^2 - 1)$$

This strain measure is used for problems involving large motions but only small strains.

Obviously many strain functions are possible and all of these satisfy the basic restrictions. The choice of the function depends on the user's convenience. There are two considerations. First, how easy the strain is able to be computed from the displacements, and second the appropriateness of the strain measure with respect to the particular constitutive model. The above examples do all refer to one-dimensional cases. In equation 2.11 the generalisation to a three-dimensional case is shown by the example of Green's strain.

$$\varepsilon^{\mathbf{G}} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}) \tag{2.11}$$

2.3 The multiplicative split of the deformation gradient

Many useful materials can carry only very small amounts of elastic strain in comparison to the plastic strain. Conventional structure steel is such a material. If we have elastic and plastic strain in a material particle, the elastic deformation reverses by unloading. The total deformation is elastic deformation multiplied by plastic deformation as shown in equation 2.12.

$$\mathbf{F} = \mathbf{F}^{el}\mathbf{F}^{pl} \tag{2.12}$$

2.4 Equilibrium and virtual work

The exact solution of each mechanical problem requires that both force and moment equilibrium are maintained at all times over any arbitrary volume of the body. An approximation of this equilibrium requirement is used in the displacement finite element method. In this section we write the exact equilibrium in the form of the virtual work statement. The exact equilibrium of force, as shown in equation 2.13 indicates that the sum of body force and surface traction has to be zero.

$$\int_S \vec{t} dS + \int_V \vec{f} dV = \vec{0} \tag{2.13}$$

In equation 2.13, t is the force per unit at any point on the surface S, and f is the body force per unit of current volume. At any point on surface S, $\vec{t}$ can be calculated from the Cauchy stress and the unit vector $\vec{n}$. This interrelationship is defined in equation 2.14.

$$\vec{t} = \vec{n}\sigma \tag{2.14}$$

If we use these two definitions, equations 2.13 and 2.14, we get

$$\int_S \vec{n}\sigma dS + \int_V \vec{f} dV = \vec{0}. \tag{2.15}$$

By using the Gauss's theorem we are able to write a surface integral as a volume integral. So we can write 2.15 as

$$\int_S \vec{n}\sigma dS = \frac{\partial}{\partial \vec{x}} \cdot \sigma dV. \tag{2.16}$$

Since the volume is arbitrary, this equation must hold at each point in the body. Thus, the integrals can be omitted.

$$\frac{\partial}{\partial \vec{x}} \cdot \sigma + \vec{f} = \vec{0} \tag{2.17}$$

The same considerations for the moment equilibrium found that the Cauchy stress matrix must be symmetric. 2.18 shows the general case of a moment equilibrium.

$$\int_S (\vec{x} \times \vec{t}) dS + \int_V (\vec{x} \times \vec{f}) dV = \vec{0} \tag{2.18}$$

Gauss's theorem applied to Equation 2.18 shows that the Cauchy stress matrix must be symmetric.

$$\sigma = \sigma^T \tag{2.19}$$

Up to this point no approximations were done. The basis of the displacement finite element method is replacing the equilibrium equation by an equivalent "weak form". This form is obtained by multiplying the pointwise differential equations by a "test function", and integrating as shown in equation 2.20. The "test function" is a "virtual" velocity field $\delta\vec{v}$.

$$\int_V \left[\frac{\partial}{\partial \vec{x}} \cdot \sigma + \vec{f} \right] \cdot \delta\vec{v} dV = 0 \tag{2.20}$$

After applying the chain rule and Gauss's theorem we get equation 2.21.

$$\int_S \vec{t} \cdot \delta\vec{v} dS + \int_V \vec{f} \cdot \delta\vec{v} dV = \int_V \sigma : \left(\frac{\partial \delta\vec{v}}{\partial \vec{x}} \right) dV \tag{2.21}$$

In the last term of equation 2.21 we find the virtual velocity gradient in the current configuration as shown in equation 2.22.

$$\left(\frac{\partial \delta\vec{v}}{\partial \vec{x}} \right) = \delta\mathbf{L} = \delta\mathbf{D} + \delta\mathbf{W} \tag{2.22}$$

The gradient is able to decompose into a symmetric and an antisymmetric part. $\delta\mathbf{D}$ is the symmetric part and $\delta\mathbf{W}$ is the anti symmetric part.

$\delta\mathbf{D} = \mathrm{sym}(\delta\mathbf{L}) = \frac{1}{2}(\delta\mathbf{L} + \delta\mathbf{L}^T)$ ... virtual strain rate

$\delta\mathbf{W} = \text{asym}(\delta\mathbf{L}) = \frac{1}{2}(\delta\mathbf{L} - \delta\mathbf{L}^T)$ ... virtual rate of spin

With these definitions:

$\sigma : \delta\mathbf{L} = \sigma : \delta\mathbf{D} + \sigma : \delta\mathbf{W}$

Since σ is symmetric,

$\sigma : \delta\mathbf{W} = \frac{1}{2}\sigma : \delta\mathbf{L} - \frac{1}{2}\sigma : \delta\mathbf{L}^T = \frac{1}{2}\sigma : \delta\mathbf{L} - \frac{1}{2}\sigma : \delta\mathbf{L} = 0$

Now it is possible to write the virtual work equation in the classical form as shown in equation 2.23.

$$\int_V \sigma : \delta\mathbf{D} dV = \int_S \delta\vec{v} \cdot \vec{t} dS + \int_V \delta\vec{v} \cdot \vec{f} dV \tag{2.23}$$

2.5 Stress Measures

The conjugate virtual strain rate (the rate of deformation) and the equilibrium in terms of Cauchy (“true”) stress are expressed in the virtual work statement as shown in equation 2.23. In this case “conjugate” means work conjugate. Work per current volume is defined by the product of stress and strain rate.

A solid material has a natural, elastic, reference state. To this reference state it returns upon unloading. A fully elastic material will always return to the original, unstressed state. A material like metal which yields, will be modified by the inelastic deformation.

The elasticity of the material is derivable from a thermodynamic potential. So there is a potential function for the elastic strain energy per unit of the natural reference volume for isothermal deformations. With this information the work rate per unit of volume in this elastic reference state is defined, as shown in equation 2.24.

$$dW^0 = \tau : d\varepsilon \tag{2.24}$$

In equation 2.24 ε is a particular choice of strain matrix based on the discussion in section 2.2. Furthermore τ is the stress matrix that is work conjugate to $d\varepsilon$.

The internal virtual work rate, as seen in equation 2.23, could be transformed as an integral over the natural reference volume as shown in equation 2.25.

$$\int_V \sigma : \mathbf{D} dV = \int_{V^0} J\sigma : \mathbf{D} dV^0 \tag{2.25}$$

J is the Jacobian of the elastic deformation (and is defined as dV/dV^0), which is between the natural reference and the current volume.

The stress measure is defined in equation 2.26 and it is named Kirchhoff stress.

$$\tau = J\sigma \tag{2.26}$$

$$\mathbf{D} = \mathrm{sym}\left(\frac{\partial v}{\partial x}\right) \tag{2.27}$$

Equation 2.27 shows the rate of deformation of the strain measure. The stress measure, equation 2.26, is work conjugated to the rate of deformation, equation 2.27.

If we use the standard Green's strain matrix, described in equation 2.11, $\varepsilon^G = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I})$, the rate of Green's strain is

$$\dot{\varepsilon}^G = \frac{1}{2}\left(\dot{\mathbf{F}}^T \cdot \mathbf{F} + \mathbf{F}^T \cdot \dot{\mathbf{F}}\right) \tag{2.28}$$

Using $\dot{\mathbf{F}} = \mathbf{F} \cdot \mathbf{L}$ so that

$\dot{\varepsilon}^G = \frac{1}{2}\mathbf{F}^T \cdot \left(\mathbf{L} + \mathbf{L}^T\right) \cdot \mathbf{F} = \mathbf{F}^T \cdot \mathbf{D} \cdot \mathbf{F}$

and, thus,

$$\mathbf{D} = \mathbf{F}^{-T} \cdot \dot{\varepsilon}^G \cdot \mathbf{F}^{-1} \tag{2.29}$$

The equation 2.29 can be used for the calculation of the work rate per unit reference volume

$$dW^0 = J\sigma : \left(\mathbf{F}^{-T} \cdot \dot{\varepsilon}^G \cdot \mathbf{F}^{-1}\right) = J\left(\mathbf{F}^{-1} \cdot \sigma \cdot \mathbf{F}^{-T}\right) : \dot{\varepsilon}^G \tag{2.30}$$

So we get the second Piola-Kirchhoff stress tensor, $\mathbf{S}$, equation 2.31, which is work conjugated to $\dot{\varepsilon}^G$, as shown in equation 2.28.

$$\mathbf{S} = J\mathbf{F}^{-1} \cdot \sigma \cdot \mathbf{F}^{-T} \tag{2.31}$$

2.6 Basic finite element equation

In this section the basic finite element equations are described. For a detailed description see SIMULIA-DassaultSystémes, 2012.

If the left-hand side of the virtual work equation 2.23 is replaced with the integral over the reference volume of the virtual work rate per reference volume defined by any conjugate pairing of stress and strain, the equation can be written in another form as shown in equation 2.32.

$$\int_{V^0} \tau^c : \delta\varepsilon dV^0 = \int_S \delta\vec{v} \cdot \vec{t} dS + \int_V \delta\vec{v} \cdot \vec{f} dV \tag{2.32}$$

This equation includes variables which are any conjugate pairing of material stress and strain measures. These variables are τ^c and ε.

$$\delta\vec{v} = \mathbf{N}_N \delta\vec{v}^N \tag{2.33}$$

$$\delta\varepsilon = \beta_N \delta\vec{v}^N \tag{2.34}$$

$\mathbf{N}_N$ is the interpolation function. It is used to interpolate the nodal points displacements within the elements and the virtual velocity field $\delta\vec{v}$. The strain-displacement-matrix β_N shows the relationship between the virtual velocity field and strains. So the equation 2.23 can be written as

$$\delta\vec{v}^N \left[\int_{V^0} \beta_N : \tau^c dV^0 - \int_S \mathbf{N}_N^T \cdot \vec{t} dS + \int_V \mathbf{N}_N^T \cdot \vec{f} dV \right] = 0 \tag{2.35}$$

Since the virtual variables are independent and arbitrary, we obtain the equation for the assumed displacement finite element analysis procedure.

$$\int_{V^0} \beta_N : \tau^c dV^0 = \int_S \mathbf{N}_N^T \cdot \vec{t} dS + \int_V \mathbf{N}_N^T \cdot \vec{f} dV \tag{2.36}$$

The Jacobian of the finite element equilibrium equations is needed for the Newton algorithm. Equation 2.37 shows a variation of equation 2.32, which is used to develop the Jacobian.

$$\begin{aligned} \int_{V^0} (d\tau^c : \delta\varepsilon + \tau^c : d\delta\varepsilon)\, dV^0 - \int_S d\vec{t}^T \cdot \delta\vec{v} dS - \int_S \vec{t}^T \cdot \delta\vec{v} dA_r \frac{1}{A_r} dS \\ - \int_V d\vec{f}^T \cdot \delta\vec{v} dV - \int_V \vec{f}^T \cdot \delta\vec{v} dJ \frac{1}{J} dV = 0 \end{aligned} \tag{2.37}$$

In equation 2.37 $d()$ represents the linear variation of the quantity with respect to the basic variables. As shown in equation 2.38, J is the volume change between the reference and the current volume.

$$J = \left| \frac{dV}{dV^0} \right| \tag{2.38}$$

A_r is the surface area ratio between the reference and the current configuration, as shown in equation 2.39.

$$A_r = \left| \frac{dS}{dS^0} \right| \tag{2.39}$$

We get the Jacobian matrix by restricting the above variations. Only variations in the nodal variables u^N are allowed. With these findings, the equation 2.37 is examined term by term.

The first term includes $d\tau^c$. We assume that the constitutive theory allows us to write

$$d\tau^c = \mathbf{H} : d\varepsilon + \mathbf{g}$$

H and **g** are defined in terms of the current state. Please see SIMULIA-DassaultSystémes, 2012 for a detailed information on the matrices **H** and **g** forming the material model. From the choice of the generalised strain measure and interpolation function,

$$\partial_N \varepsilon = \frac{\partial \varepsilon}{\partial u^N} = \beta_N$$

Based on the above constitutive assumption,

$$\partial_N \tau^c = \mathbf{H} : \beta_N$$

The second part of the first term in equation 2.37 is $\delta\varepsilon$. It is the first variation of ε with respect to the nodal variables,

$$\delta\varepsilon = \partial_M \varepsilon \delta u^M = \beta_M \delta u^M$$

Therefore we obtain the first term in the Jacobian Matrix,

$$\int_{V^0} \beta_M : \mathbf{H} : \beta_N dV^0$$

the usual "small-displacement stiffness matrix".

The second term in equation 2.37 is

$$\int_{V^0} \tau^c : d\delta\varepsilon dV^0.$$

It is rewritten as

$$\int_{V^0} \tau^c : \partial_N \delta\varepsilon dV^0,$$

which is

$$\int_{V^0} \tau^c : \partial_N \beta_M dV^0.$$

This term is the second term in the Jacobian matrix and is called the "initial stress matrix."

The next term in equation 2.37 is the external load rate. In general, these load vectors can be written as

$$\vec{t} = \vec{t}(\lambda, \vec{x}) \text{ and } \vec{f} = \vec{f}(\lambda, \vec{x}).$$

In this equation λ represents the externally prescribed loading parameters. The variation of the load vector with nodal variables can then be written symbolically as

$$\partial_N \vec{t} + \vec{t} \frac{1}{A_r} \partial_N A_r = \mathbf{Q}_N^S,$$

$$\partial_N \vec{f} + \vec{f} \frac{1}{J} \partial_N J = \mathbf{Q}_N^V,$$

and then writing

$$\delta \vec{v} = \mathbf{N}_M \delta v^M.$$

In this equation $\mathbf{N}_M$ is obtained directly from the interpolation functions. So the Jacobian terms of the last four terms of equation 2.37 can be written as

$$-\int_S \mathbf{N}_M^T \cdot \mathbf{Q}_N^S dS - \int_V \mathbf{N}_M^T \cdot \mathbf{Q}_N^V dV.$$

These terms are called the “load stiffness matrices.”

So we can write the complete Jacobian matrix, which is shown in equation 2.40.

$$\begin{aligned} \mathbf{K}_{MN} = \int_{V^0} \beta_M : \mathbf{H} : \beta_N dV^0 + \int_{V^0} \tau^c : \partial_N \beta_M dV^0 \\ - \int_S \mathbf{N}_M^T \cdot \mathbf{Q}_N^S dS - \int_V \mathbf{N}_M^T \cdot \mathbf{Q}_N^V dV \end{aligned} \tag{2.40}$$

2.7 Non-linear analysis

The generated finite element models in Abaqus are usually nonlinear. These models can involve from a few to thousands of variables. Similar to equation 2.23, a work equation can be written symbolically as

$$F^N(u^M) = 0 \tag{2.41}$$

In this equation F^N is the force component conjugate to the N^{th} variable in the problem. u^M is the value of the M^{th} variable. The task is to find the solutions for the u^M throughout the history of interest for the equation 2.41. In common for solving the nonlinear equation the Newton method is used, which is characterised by the following basic formalism: For example, we obtain an approximation u_i^M to the solution after an iteration i. So there is a difference between this solution and the exact solution to the discrete equilibrium equation. This difference is described as c_{i+1}^M. This means that

$$F^N(u_i^M + c_{i+1}^M) = 0$$

If the left side of this equation is written as a Taylor series about the approximate solution u_i^M, then we obtain equation 2.42

$$F^N(u_i^M) + \frac{\partial F^N}{\partial u^P}(u_i^M)c_{i+1}^P + \frac{\partial^2 F^N}{\partial u^P \partial u^Q}(u_i^M)c_{i+1}^P c_{i+1}^Q + ... = 0 \tag{2.42}$$

To get a linear system of equations all terms can be neglected except the first two. This neglection is possible, because the term c_{i+1}^P is very small. This linear system of equations can be written as equation 2.43.

$$K_i^{NP} c_{i+1}^P = -F_i^N \tag{2.43}$$

In this equation

$$K_i^{NP} = \frac{\partial F^N}{\partial u^P}(u_i^M)$$

is the Jacobian matrix and

$$F_i^N = F^N(u_i^M)\ .$$

The next approximation to the solution is then

$$u_{i+1}^M = u_i^M + c_{i+1}^M$$

and the iteration continues.

For further information the reader is referred to SIMULIA-DassaultSystémes, 2012

3 Summary of the appended paper

3.1 Title: A Proposed Failure Mechanism for Pulp Fiber-Fiber Joints

3.2 Abstract

Due to stress concentration at the edges fiber-fiber bonds under load are known to fail gradually, inwards from the edges. In this paper we propose a failure mechanism for fiber-fiber joints under load, based on the peak stresses occurring at the bond edges. We have modeled the mechanical testing of individual fiber-fiber joints under different modes of loading using a FEM modeling framework. The model is based on results of fiber-fiber joint strength tests designed to induce each of the three failure modes in fracture mechanics: Mode 1 (opening), Mode 2 (shear) and Mode 3 (twisting). We carried out a parameter study of the peak load at the edges of the fibers in order to identify a failure mechanism. The peak stresses were not directly taken from the FEM models as these values are highly discretization-dependent. Instead the peak stresses were estimated from resultant forces and moments in the bond and an idealized geometry of the bonding region. The literature has up to now focused on shear load as a failure mechanism for fiber fiber bonds. However, our findings indicate that pulp fiber joints are sensitive to normal stresses and insensitive to shear stresses. Hence, we suggest to utilize failure criteria related to normal stress in future work.

3.3 Contribution to paper

- development of input script in Abaqus to model a fibre - fibre bond

- performance of all simulations
- evaluation and interpretation of numerical results in cooperation with M. Ulz
- formulation of submitted text to the journal Cellulose in cooperation with U. Hirn and M. Ulz

Cellulose

A Proposed Failure Mechanism for Pulp Fiber-Fiber Joints

--Manuscript Draft--

Manuscript Number:	
Full Title:	A Proposed Failure Mechanism for Pulp Fiber-Fiber Joints
Article Type:	Original Research
Keywords:	failure criteria; interfiber joint strength; fiber-fiber bond; shear and normal stress
Corresponding Author:	Manfred H Ulz Graz University of Technology Graz, AUSTRIA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Graz University of Technology
Corresponding Author's Secondary Institution:	
First Author:	Thomas Ebner
First Author Secondary Information:	
Order of Authors:	Thomas Ebner
	Ulrich Hirn
	Wolfgang J Fischer
	Franz J Schmied
	Robert Schennach
	Manfred H Ulz
Order of Authors Secondary Information:	
Abstract:	Due to stress concentration at the edges fiber-fiber bonds under load are known to fail gradually, inwards from the edges. In this paper we propose a failure mechanism for fiber-fiber joints under load, based on the peak stresses occurring at the bond edges. We have modeled the mechanical testing of individual fiber-fiber joints under different modes of loading using a FEM modeling framework. The model is based on results of fiber-fiber joint strength tests designed to induce each of the three failure modes in fracture mechanics: Mode 1 (opening), Mode 2 (shear) and Mode 3 (twisting). We carried out a parameter study of the peak load at the edges of the fibers in order to identify a failure mechanism. The peak stresses were not directly taken from the FEM models as these values are highly discretization-dependent. Instead the peak stresses were estimated from resultant forces and moments in the bond and an idealized geometry of the bonding region. The literature has up to now focused on shear load as a failure mechanism for fiber-fiber bonds. However, our findings indicate that pulp fiber joints are sensitive to normal stresses and insensitive to shear stresses. Hence, we suggest to utilize failure criteria related to normal stress in future work.
Suggested Reviewers:	Tetsu Uesaka Mid Sweden University Tetsu.Uesaka@miun.se expert in modeling pulp fibers
	Mikael Magnusson Innventia mikael.magnusson@innventia.com expert in modeling pulp fibers;former staff member of the Department of Solid

	Mechanics, KTH Royal Institute of Technology, Sweden (see references)
	Sören Östlund KTH Royal Institute of Technology soren@kth.se expert in modeling pulp fibers

Cellulose manuscript No.
(will be inserted by the editor)

A Proposed Failure Mechanism for Pulp Fiber-Fiber Joints

Thomas Ebner · Ulrich Hirn · Wolfgang J. Fischer · Franz J. Schmied · Robert Schennach · Manfred H. Ulz

Received: date / Accepted: date

Abstract Due to stress concentration at the edges fiber-fiber bonds under load are known to fail gradually, inwards from the edges. In this paper we propose a failure mechanism for fiber-fiber joints under load, based on the peak stresses occurring at the bond edges.

We have modeled the mechanical testing of individual fiber-fiber joints under different modes of loading using a FEM modeling framework. The model is based on results of fiber-fiber joint strength tests designed to induce each of the three failure modes in fracture mechanics: Mode 1 (opening), Mode 2 (shear) and Mode 3 (twisting). We carried out a parameter study of the peak load at the edges of the fibers in order to identify a failure mechanism. The peak stresses were not directly taken from the FEM models as these values are highly discretization-dependent. Instead the peak stresses were estimated from resultant forces and moments in the bond and an idealized geometry of the bonding region.

The literature has up to now focused on shear load as a failure mechanism for fiber-fiber bonds. However, our findings indicate that pulp fiber joints are sensitive to normal stresses and insensitive to shear stresses. Hence, we suggest to utilize failure criteria related to normal stress in future work.

Keywords failure criteria · interfiber joint strength · fiber-fiber bond · shear and normal stress

T. Ebner · M. H. Ulz
Institute for Strength of Materials, Graz University of Technology, 8010 Graz, Austria
E-mail: manfred.ulz@tugraz.at

U. Hirn · W. J. Fischer
Institute for Paper, Pulp and Fiber Technology, Graz University of Technology, 8010 Graz, Austria

F. J. Schmied
Mondi, R&D Paper Europe & International, Theresienthalstrae 50, 3363 Ulmerfeld-Hausmening, Austria

R. Schennach
Institute of Solid State Physics, Graz University of Technology, 8010 Graz, Austria

T. Ebner · U. Hirn · W. J. Fischer · F. J. Schmied · R. Schennach
CD-Laboratory for Surface Chemical and Physical Fundamentals of Paper Strength, Graz University of Technology, 8010 Graz, Austria

1 Introduction

The bonding strength between pulp fibers in paper is one of the key parameters determining the strength of the paper. It is not possible to measure fiber-fiber bond strength reliably from paper sheets because paper strength also depends on other factors like e.g. fiber length, fiber tensile strength, paper density and straining during drying of the sheet. Therefore, fiber-fiber bond strength is usually investigated by measuring the bond strength of individual fiber-fiber joints, compare Saketi and Kallio (2011), Fischer et al. (2012), Magnusson et al. (2013b), Schmied et al. (2012), Schniewind et al. (1964). It might be intuitive to think that the strength (i.e. the breaking load in N) of a fiber-fiber joint is composed of a *specific bond strength* (bonding force per unit area, N/m^2) times the bonded area (in m^2) of the fiber-fiber joint. This, however, is not the case. Stress concentrations occur at the edges of the bonding area (Button (1979), Uesaka (1984), Page (2002)) which leads to a progressive failure of the fiber-fiber bonds. This progressive failure has also been observed in fiber-fiber joint testing, where sudden drops in loading force indicate local failure of the bond. Please refer to Uesaka (1984), Magnusson et al. (2013c) and specifically to Schmied et al. (2013).

There is considerable evidence that failure in paper also occurs due to progressive failure of fiber-fiber bonds. Nordman et al. (1952) found that the light scattering coefficient of paper increases upon straining. The increase in light scattering can be attributed to new surface area created in the paper due to the separation of previously bonded fiber regions (Page, 2002). Investigations of fiber-fiber bonds in paper using polarized light microscopy have shown that the bonds indeed fail progressively, under dynamic load (Page et al., 1962) as well as under constant load, i.e. creep testing (DeMaio et al., 2006).

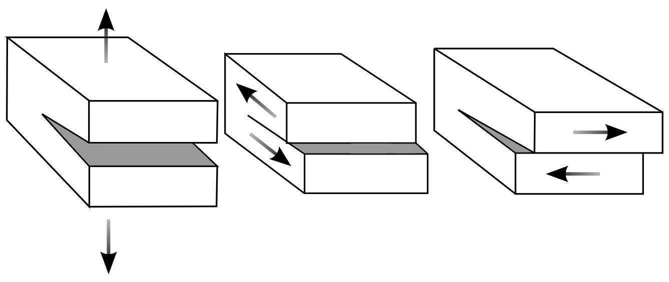
It is the aim of this work to propose a key mechanism of fiber-fiber bond failure based on the peak stresses occurring at the edges of the bonds. Progressive failure is always initiated by the peak stresses in the structure. Therefore failure theories give a criterion for yield or fracture in the material by providing a scalar representation of a multiaxial state of stress, i.e. the normal- and shear stress are combined into a single value (Brinson and Brinson, 2008, Pruitt and Chakravartula, 2011). It is important to understand that in many respects the behavior of pulp fibers is fundamentally different from classical engineering materials: typically, pulp fibers possess a sophisticated hierarchical micro-structure (Bodig and Benjamin, 1993). Therefore, classical failure theories may not directly apply. Collagen, like pulp fibers, is a viscoelastic, fibril based biomaterial. It is well researched, because of its relevance regarding defects and surgery of blood vessels. Still, no conclusive failure mechanism has been worked out for this material, although several different failure mechanisms have been discussed, compare Gasser (2011) and also Wang et al. (1997). Recently a comprehensive Finite Element Method (FEM) framework to model the behavior of fiber-fiber joints during mechanical testing was presented by Magnusson et al. (2013c). The work focused on resultant forces and moments in the bonding regions and did not consider local stress concentrations. Based on that they discussed a failure criterion according to which the bonds are more sensitive to shear load than to normal load. For further work they recommended incorporation of local stress variations, e.g. by cohesive zone modeling. In a recent review (Da Silva and Campilho, 2012) on cohesive zone modeling, several different failure models are discussed for fiber-based composites, the results there also do not permit a general recommendation for the case of pulp fibers.

In this work we will propose a key mechanism of pulp fiber-fiber bond failure based on the analysis of peak stresses inferred from FEM models of fiber-fiber bond mechanical testing. We have conducted three different types of fiber-fiber bond strength measurements, each one designed to predominantly load the fiber-fiber joint in one of the three fracture modes, compare Figure 1. The parameters for the FEM models are taken from the experiments and the literature, compare Magnusson et al. (2013c). Several parameters that represent the characteristic features of the pair of bonded fibers are defined. These are fiber thickness t, fiber width w, fiber fibril angle ψ and crossing angle ϕ of the fiber-fiber joint. These parameters are varied in physically meaningful ranges in a parameter study for three different types of loading, which correspond to mode 1, mode 2 and mode 3 type of fracture modes. The applied loading in the numerical model is taken from the corresponding experimental results at rupture. For each parameter set and type of loading, the arising resultant shear and normal forces as well as the resultant peeling, twisting and tearing moments in the bond region are obtained with the help of a FEM model in Abaqus (2012). These resultant forces and moments allow to compute estimated normal and shear stress distributions in the interfiber joint based on a simplified model of the fiber-fiber joint geometry. Based on that, we can identify peak values for normal and shear stress for each parameter set.

The paper is organized as follows. Section 2 describes the experiments on fiber-fiber joints and provides the experimentally obtained parameters for the FEM model. Section 3 shows the FEM discretization and the computation of the estimated normal and shear stress distributions in the bonding region. Section 4 presents the obtained peak stresses for the three types of loading. Our results show a surprising behavior: while the obtained peak values for normal stress are within the same range for the three types of loading, the peak values for shear stress are clearly in different ranges. This suggests that normal stress plays an important role in the failure of pulp fiber-fiber bonds.

2 Experimental evaluation

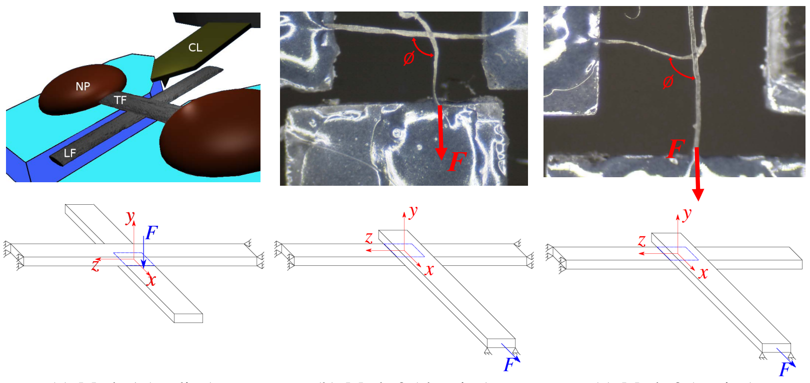
In fracture mechanics three different modes of progressive failure are known, Figure 1. Cracks may propagate in the plane perpendicular to normal stress (Mode 1, peeling), in the plane with shear stresses with the crack line perpendicular to the stresses (Mode 2, shearing) or in the plane with shear stress with the crack line parallel to the shear stress (Mode 3, tearing). In single fiber testing we have performed experiments to specifically address these different failure modes.



(a) Mode 1 (peeling) (b) Mode 2 (shearing) (c) Mode 3 (tearing)

Fig. 1 Illustration of the three modes of fracture mechanics.

For this work the experimental setup is shown in Figure 2. The setup in Figure 2(a) creates predominantly Mode 1 load, the setup in Figure 2(b) creates predominantly Mode 2 load and the setup in Figure 2(c) creates predominantly Mode 3 load. The details for the experimental procedure for Mode 1 are described by Schmied et al. (2012) and for Modes 2 and 3 by Fischer et al. (2012). Please note that the configurations shown in Figure 2 do *not* create pure loadings according to Mode 1, 2 and 3. Due to the curved geometry of the fibers, fiber twisting during the experiment and the tilting of the fibril angle to the fiber axis there is a large amount of peeling-, twisting- and tearing load on the bonding region in all three experiments. For a detailed analysis of this question please refer to Magnusson et al. (2013c,a).



(a) Mode 1 (peeling) (b) Mode 2 (shearing) (c) Mode 3 (tearing)

Fig. 2 The experimental setup for the three modes.

The geometry of the specimens were captured by micrographs. Furthermore, the applied force at rupture was measured. As the individual pairs of bonded fibers differ strongly, mean values were reported. The mean fiber width is found to be 32.00µm and the mean fiber thickness equals 7.45µm. The experimentally obtained mean force in Mode 1 equals 0.33mN, see Schmied et al. (2013). The mean force for Mode 2 is 6.45mN and for Mode 3 it is 1.06mN, see Fischer et al. (2012).

3 Structural analysis using a finite element model

3.1 Geometric discretization, material behavior and loading

The cell wall of pulp fibers consists of four major layers. These are the primary wall and three secondary layers (S_1, S_2, S_3) as shown in Figure 3. All layers are composed of cellulose, hemicellulose and lignin in varying composition (Bodig and Benjamin, 1993). Furthermore, each secondary layer shows a micro-fibril wrapped helically along the fiber at a specific angle. The fiber's cell wall is made up to 80-85% of the S_2 layer (Page, 1969a) and it is commonly assumed in the literature that this layer has the highest influence on the fiber's mechanical behavior (*e.g.* Magnusson and Östlund (2013)). Therefore, the pulp fiber will be modeled by the S_2 layer only.

The objective of this work is to develop a numerical model of a pair of bonded fibers, which keeps the principled characteristics, but neglects superfluous details. Each real pair of

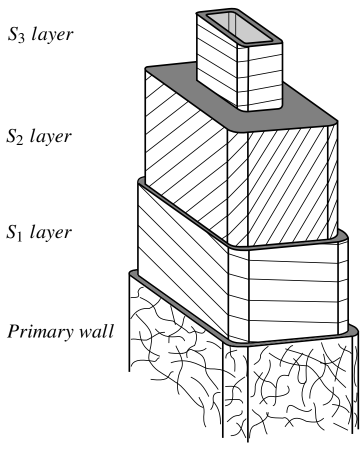


Fig. 3 The layered structure of a single pulp fiber.

bonded fibers is unique. It will differ from any other pair in terms of geometry and material properties. Therefore, a system that is reduced to a minimal set of parameters is chosen to study the distinct influence of the model parameters. The fiber-fiber cross is modeled by two straight beams, which is tantamount to neglecting the curvature and the twist along the fiber direction (Seth, 2006). The model parameters are chosen to be the width w, the thickness t and the fibril angle ψ of the fibers, Figure 4. Furthermore, the crossing angle ϕ of both fibers is investigated, Figure 2(b,c).

The fibers are considered as fully collapsed volumetric bodies. The cross section of the idealized fiber model is given in Figure 4. Each fiber consists of two parts with the micro-fibril pointing in opposite directions in each part. If the upper part shows an angle of $\psi = 30°$, then the lower part has $-30°$. The micro-fibril angle is expected to be constant along the fiber length. Furthermore, the length of the fibers is taken to be 1mm, in close agreement with the previously described experiments. In all performed computations the loaded fiber is positioned right in the middle of the fixed fiber.

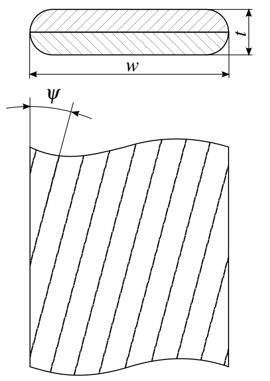


Fig. 4 Cross section and geometry of the idealized fiber structure.

The material behavior of the fiber (modeled by the S_2 layer only) is chosen to be transversely isotropic in the model. This material law considers the effect of the micro structure of the fiber. The micro-fibrils act as a reinforcement in the matrix of lignin and hemicellulose. The axis of transverse isotropy is aligned with the direction of the micro-fibril. The material constants as used in the simulation are shown in Table 1. The modulus of elasticity $E_1 = 30$GPa is chosen as an average of the data given in Magnusson and Östlund (2013). It has to be mentioned that the material properties of the S_2 layer are subject to wide variations (Groom et al., 1995, Neagu et al., 2004, Page et al., 1977). Furthermore, the fibers will show viscoelastic material behavior in reality. As there is no material data available for this behavior, we assume the fiber to behave according to the previously described anisotropic elastic model.

Coefficient	E_1	$E_2 = E_3$	$G_{12} = G_{13}$	G_{23}	$\nu_{12} = \nu_{13}$	ν_{23}
Value	E_1	$\frac{E_1}{11}$	$\frac{E_1}{23}$	$\frac{E_2}{2(1+\nu_{23})}$	0.022	0.39

Table 1 Material constants of the cell wall. Taken from Magnusson and Östlund (2013).

Three different modes of loading were tested according to the experiments described in Section 2. The three models of the various modes, their boundary conditions and the direction of the applied force can be seen in Figure 2. In mode 2 and 3 the load is applied in x-direction. If the crossing angle ϕ is different to 90° and thereby the axis of the loaded fiber is not aligned to the x-direction, then the force is still applied in x-direction. In mode 1 the applied force points into the negative y-direction. We assume the load to rupture the bonding region to be much smaller than the load to plastically deform (or even rupture) the fiber (Burgert et al., 2003). Hence, the bonding region is the predetermined breaking point of the structure.

3.2 Finite element discretization

The commercial FEM software Abaqus (2012, version 6.11-2) and its scripting interface in Python is used to perform the simulations (non-linear quasi-static FEM model). The pair of bonded fibers is discretized using a mesh consisting of 8-noded hexahedral elements with reduced integration (C3D8R in the Abaqus element library). A mesh size dependency check is performed and the elements' size is chosen to render the deviation in the results of Section 5 to be practically insignificant.

The FEM model assumes the contact area to be fully bonded, which is unlikely for real bonded fibers. Regions close to the edge of the bonding region or even in the interior of the bonding region may not be molecularly bonded (Page, 1960, Kappel et al., 2009). It is discussed in Torgnysdotter et al. (2007a,b) that the degree of contact is of great importance for the maximum stress in the bonding region. Recent results suggest that usually there is a high degree of bonding between fiber surfaces (Hirn et al., 2013), thus for simplification we neglect possible flaws in the bonding. Furthermore, we assume that the contact zone does not change before rupture. As a result, the two surfaces of the fibers in contact are tied to each other by a surface-to-surface contact discretization (tie constraints in Abaqus). An example of the meshed pair of bonded fibers is given in Figure 5.

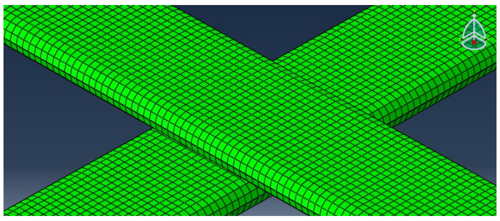


Fig. 5 Finite element model of the fiber-fiber bond.

3.3 Resultant forces and moments in the bonding region

The applied loading causes resultant reaction forces and moments in the bonding region (compare with the similar treatment in Magnusson and Östlund (2013)). These are described in a local coordinate system, the origin of which is defined at the centroid of the interface region. As already shown in Figure 2, the y-axis is defined by the outward unit normal, z is defined in direction of the fixed fiber, and x is orthogonal to the previous two directions. The resultant reaction forces and moments in coordinate directions are computed from the nodal forces (NFORC in Abaqus). The quantities N_i, Q_{xi} and Q_{zi} are the nodal forces at node i (for n nodes in the bonding region) in y-. x- and z-direction, respectively. The resultant reaction forces are computed as

$$N = \sum_{i=1}^{n} N_i, \qquad Q_x = \sum_{i=1}^{n} Q_{xi}, \qquad Q_z = \sum_{i=1}^{n} Q_{zi}. \tag{1}$$

The resultant moments in the local coordinate system are found as

$$M_x = \sum_{i=1}^{n} -z_i \cdot N_i, \qquad M_z = \sum_{i=1}^{n} x_i \cdot N_i, \qquad M_y = \sum_{i=1}^{n} z_i \cdot Q_{xi} - x_i \cdot Q_{zi}. \tag{2}$$

The quantities x_i and z_i are the perpendicular distances of the nodal forces to the origin of the coordinate system. Figure 6 gives a visualization of the resultant forces and moments in the bonding region.

4 Resultant stresses in the bonding region

Although it may appear straightforward, the peak stresses extracted directly from the FEM model of the fiber-fiber joints need to be treated with care, for a detailed discussion on this topic please refer to Da Silva and Campilho (2012). The peak stresses are typically found close to the stress discontinuities of the model, i.e. sharp corners or interfaces with different material properties. In our case this is where the rounded edge of one fiber touches the surface of the other fiber, compare Figure 5. The magnitude of the peak stresses in the FEM model strongly depend on how well the stress field is modeled around these discontinuities, specifically it is very sensitive to both the mesh size used and the considered geometrical details in the model. In particular, the latter cannot be appropriately met in any simplified fiber-fiber model. Therefore, we refrain from extracting the peak stresses directly from the model.

Instead we apply the resulting forces and moments (as described in the previous section) to estimate the peak stresses using an idealized model of the bonding region. We are

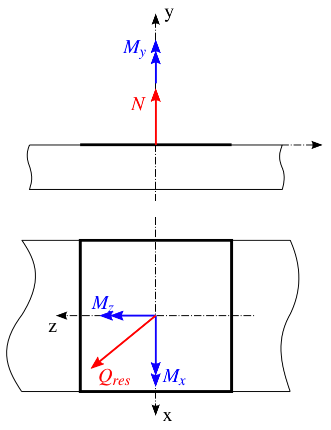


Fig. 6 Resultant forces and moments in the bonding region. $Q_{res} = \sqrt{{Q_x}^2 + {Q_z}^2}$.

simplifying the actual stress situation in fiber-fiber joints by neglecting local unbonded regions and irregularities in the fiber geometries. We are aware that these simplifications lead to deviations from the reality in terms of absolute stresses. It is, however not our goal to correctly model the absolute values of the peak stresses or fit the experimental results to the FEM model. Instead we want to extract the *general behavior* of the peak stresses and the *relation between shear- and normal stresses*. We obtain this generalization on the one hand by simplifying the geometry of the model and on the other hand by varying the parameters for fiber-fiber bond configurations in a wide range, compare Table 2. Nevertheless, we would like to point out that the simplification only relates to the rectangular geometry of the bonding zone and the negligence of edge effects creating stress discontinuities, the calculation of the stresses follows standard procedures in mechanics. In conclusion the presented approach computes idealized stress distributions (constant for tensile and shear loading, linear for bending and torsion) and obtains a single estimated peak value for the normal stress and a single estimated peak value for the shear stress for each pair of fibers. This allows for an easy comparison of very different geometrical settings.

The interface region between the fibers in a joint is defined by the area A of the bonding region, the second area moment of inertia I for bending and the polar section modulus W_p for torsion. As can be seen in Figure 4 the length of the bonding region has the value $w-t$. We find for two orthogonal fibers (crossing angle $\phi = 90°$)

$$A = (w-t)^2, \qquad I = \frac{(w-t)^3 \cdot (w-t)}{12}, \qquad W_p = 0.208 \cdot (w-t)^3. \tag{3}$$

The presented formula for W_p is only valid for a square section area (Grote and Feldhusen, 2011). If the crossing angle ϕ is different to 90°, the bonding region A changes to a rhomboid. For this case the area A and the second area moments of inertia I_1, I_2 are analytically and the torsion constant W_p is numerically computed for principal axes.

The estimated normal stress distribution according to the resultant normal force is constant

$$\sigma_N = \frac{N}{A}. \tag{4}$$

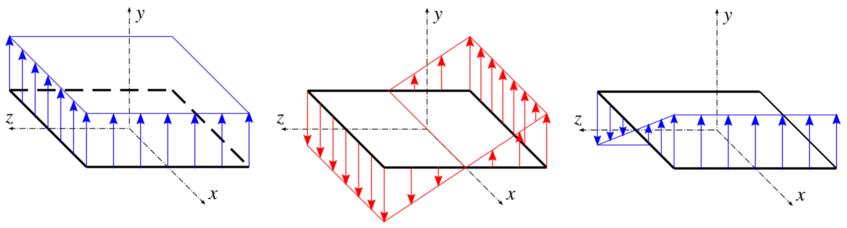
Next, the contribution to the normal stress due to bending for orthogonal fibers is computed as

$$\sigma_B = \frac{M_z}{I} \cdot x - \frac{M_x}{I} \cdot z. \tag{5}$$

If the crossing angle ϕ is different to 90°, σ_B is set up in principal axes. The total normal stress distribution is given as

$$\sigma_{res} = \sigma_N + \sigma_B \tag{6}$$

215 and is visualized in Figure 7. The maximum of the total normal stress may be obtained by computing its value at the corresponding corner of the bonding region.



(a) normal stress (b) bending stress about the x-axis (c) bending stress about the z-axis

Fig. 7 Components of normal stress.

216 The estimated shear stress distribution according to the resultant shear forces is assumed to be constant over the bonding region

$$\tau_{Qres} = \frac{1}{A}\sqrt{Q_x^2 + Q_z^2}. \tag{7}$$

Furthermore, the maximum shear stress due to torsion is computed as

$$\tau_T = \frac{M_y}{W_p}. \tag{8}$$

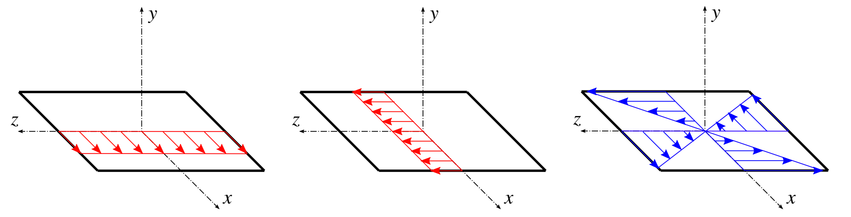
If the crossing angle ϕ is different to 90°, τ_T is numerically computed. The maximum value of the total shear stress is found as

$$\tau_{res} = \tau_{Qres} + \tau_T \tag{9}$$

217 and is visualized in Figure 8.

	t [µm]	w [µm]	ψ [°]	ϕ [°]
range	4.65 - 10.25	25.20 - 45.60	0 - 45	60 - 120
increments	0.70	3.40	5	5

Table 2 Ranges of modified parameters.



(a) shear stress in x-direction (b) shear stress in z-direction (c) torsional shear stress about the y-axis

Fig. 8 Components of shear stress.

5 Results, Discussion and Conclusions

The obtained peak values for normal and shear stresses for each parameter set and type of loading are collected and presented in Figure 9. The range of the varied parameters are listed in Table 2. The applied load is taken according to Section 2. Furthermore, a dependency check on the applied load, Young's modulus and Poisson's ratio (compare Table 1) is performed by varying a single quantity and keeping the remaining parameters unchanged. The results were essentially equivalent to Figure 9, thus we refrained from reproducing them here.

Our results show that the obtained peak values for normal stress are within the same range for all three types of loading, while the peak values for shear stress are found in different ranges (compare Figure 9). Please note that the fiber bond testing setups specifically designed to apply shear forces to the fiber-fiber bond (Mode 2 and 3 in figure 2, M2 and M3 in figure 9) have the same - or even higher peak normal stresses - like the Mode 1 configuration. Our findings thus lead to a new interpretation of the single fiber-fiber testing experiments described in the literature. The intuitive explanation that the different setups lead to different failure modes is not plausible anymore, instead we suggest that in all three types of experiments the bonds fail due to normal stresses, i.e. Mode 1 failure.

Material failure strongly depends on whether the material microstructure renders it ductile, brittle, or semi-brittle (Bartenev and Zuyev, 1968, Collins, 1981, Pruitt and Chakravartula, 2011). While ductile materials yield before failure, brittle materials will instantly fracture. A semi-brittle system shows a small amount of plastic deformation prior to failure. Metals are commonly considered as ductile (Tresca or von Mises failure criterion), ceramics as brittle (normal stress failure criterion) and polymers or tissues range somewhere in between. The mechanical behavior of macromolecular structures is known to depend on many variables in a complex manner: chain chemistry, configuration and length; molecular weight distribution; strain rate; local loading conditions; etc. (Pruitt and Chakravartula, 2011). However, our findings suggest that the fiber-fiber joint is more likely to be sensitive to normal loading than shear loading. Furthermore, this allows us to conclude that the failure criterion of fiber-fiber joints should be related to normal stress.

Finally, the ideas presented in this paper have the potential to shift the understanding of how the fiber-fiber bonds in paper are failing. The fibers in paper under tensile load are subjected to shear stress, because they are aligned predominantly in the paper plane. That has intuitively lead to the idea that the shear stresses are responsible for the paper failure. Also the most common theory on paper tensile strength, the equation of Page (1969b), employs shear stress as the key mechanism for fiber-fiber bond strength. As a consequence,

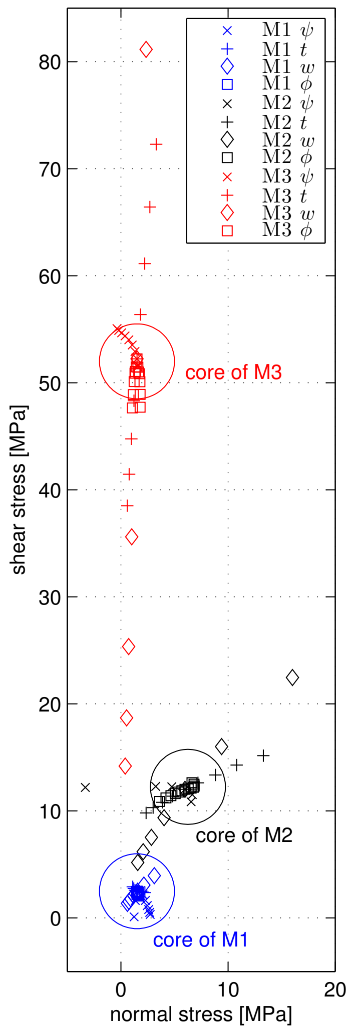


Fig. 9 Peak values of normal and shear stress of parameter study (M1 ... mode 1, M2 ... mode 2, M3 ... mode 3). For each mode a "core region" of the estimated peak stresses can be identified in the plane of normal stress and shear stress.

usually shear load is regarded to be the tensile failure mechanism in paper, also compare Page (2002). The present results, however, suggest that Mode 1 failure may be predominant in fiber-fiber bonds which is a new perspective on the mechanical failure of paper under tensile load.

Conflict of Interest Statement

The authors declare that they have no conflict of interest.

References

Abaqus FEA. 2012. *Abaqus/CAE User's Manual*. Dassault Systémes.

Bartenev GM and Zuyev YS. 1968. *Strength and failure of visco-elastic materials*. (Oxford: Pergamon Press Ltd.).

Bodig J and Benjamin AJ. 1993. *Mechanics of wood and wood composites*. (Florida: Krieger Publishing Company).

Brinson HF and Brinson LC. 2008. *Polymer engineering science and viscoelasticity: an introduction*. (New York: Springer).

Burgert I, Frühmann K, Keckes J, Fratzl P, and Stanzl-Tschegg SE. 2003. Microtensile testing of wood fibers combined with video extensometry for efficient strain detection. *Holzforschung*, 57:661–664.

Button AF. *Fiber-fiber bond strength: a study of a linear elastic model structure*. PhD thesis, IPST, Georgia Institute of Technology, 1979.

Collins JA. 1981. *Failure of materials in mechanical design: analysis, prediction, prevention*. (New York: John Wiley & Sons, Inc.).

Da Silva LFM and Campilho RDSG. 2012. *Advances in numerical modelling of adhesive joints*. SpringerBriefs in Applied Sciences and Technology. (Berlin: Springer).

DeMaio A, Lowe R, Patterson T, and Ragauskas A. 2006. Direct observations of bonding influence on the tensile creep behavior of paper. *Nordic Pulp & Paper Research Journal*, 21(3):297–302.

Fischer W, Hirn U, Bauer W, and Schennach R. 2012. Testing of individual fiber-fiber joints under biaxial load and simultaneous analysis of deformation. *Nordic Pulp & Paper Research Journal*, 27:237–244.

Gasser C. 2011. An irreversible constitutive model for fibrous soft biological tissue: A 3-d microfiber approach with demonstrative application to abdominal aortic aneurysms. *Acta Biomaterialia*, 7(6):2457–2466.

Groom LH, Shaler SM, and Mott L. Characterizing micro and macromechanical properties of single wood fibres. In *International Paper Physics Conference, Niagara-on-the-Lake*, pages 11–14, 1995.

Grote KH and Feldhusen J. 2011. *Dubbel*. (Berlin: Springer Verlag), 23 edition.

Hirn U, Schennach R, Ganser C, Magnusson M, Teichert C, and Östlund C. *Trans. of the 15th Fundamental Research Symposium, Cambridge*, chapter The area in molecular contact in fiber-fiber bonds, pages 201–226. 2013. ISBN 978-0-9926163-0-4.

Kappel L, Hirn U, Bauer W, and Schennach R. 2009. A novel method for the determination of bonded area of individual fiber-fiber bonds. *Nordic Pulp & Paper Research Journal*, 24:199–244.

Magnusson MS and Östlund S. 2013. Numerical evaluation of interfibre joint strength measurements in terms of three-dimensional resultant forces and moments. *Cellulose*, 20:1691–1710.

Magnusson MS, Fischer JW, Östlund S, and Hirn U. Interfibre joint strength under peeling, shearing and tearing types of loading. In *Trans. of the 15th Fundamental Research Symposium, Cambridge*, pages 103–124, 2013a.

Magnusson MS, Zhang X, and Östlund S. 2013b. Experimental evaluation of the interfibre joint strength of papermaking fibres in terms of manufacturing parameters and in two different loading directions. *Experimental Mechanics*. DOI 10.1007/s11340-013-9757-y.

Magnusson MS, Zhang X, and Östlund S. 2013c. Experimental evaluation of the interfibre joint strength of papermaking fibres in terms of manufacturing parameters and in two different loading directions. *Experimental Mechanics*, 53:1621–1634.

Neagu RC, Gamstedt EK, and Lindström M. 2004. Influence of wood-fibre hygroexpansion on the dimensional instability of fibre mats and composites. *Composites Part A*, 36:772–788.

Nordman L, Gustafsson C, and Gustafsson G. 1952. On the strength of bondings in paper. *Paperi ja Puu*, 3:47.

Page DH. 1960. Fibre-to-fibre bonds part 1 - a method for their direct observation. *Paper Technology*, 1:407–411.

Page DH. 1969a. A method for determining the fibrillar angle in wood tracheids. *Journal of Microscopy*, 90:137–143.

Page DH. 1969b. A theory for the tensile strength of paper. *Tappi J.*, 52(4):674–681.

Page DH. 2002. The meaning of nordman bond strength. *Nordic Pulp & Paper Research Journal*, 17(1):39–44.

Page DH, Tydeman PA, and Hunt M. A study of fibre-to-fibre bonding by direct observation. In *Fund. Res. Symp 1961, The formation and structure of paper*, volume 1, pages 171–194. 1962.

Page DH, El-Hosseiny F, Winkler F, and Lancaster APS. 1977. Elastic modulus of single wood pulp fibres. *Tappi Journal*, 60:114–117.

Pruitt LA and Chakravartula AM. 2011. *Mechanics of biomaterials: fundamental principles for implant design*. (Cambridge: Cambridge University Press).

Saketi P and Kallio P. Microrobotic platform for making, manipulating and breaking individual paper fiber bonds. In *IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pages 1–6, 2011.

Schmied F, Teichert C, Kappel L, Hirn U, and Schennach R. 2012. Joint strength measurements of individual fiber-fiber bonds an atomic force microscopy based method. *Review of scientific instruments*, 83:073902–1 – 073902–8. http://dx.doi.org/10.1063/1.4731010.

Schmied F, Teichert C, Kappel L, Hirn U, Bauer W, and Schennach R. 2013. What holds paper together: Nanometre scale exploration of bonding between paper fibres. *Scientific Reports*, 3:2432 1–6. http://dx.doi.org/10.1038/srep02432.

Schniewind AP, Nemeth LJ, and Brink DL. 1964. Fiber and Pulp Properties - I. Shear Strength of single Fiber Crossings. *Tappi J.*, **47**:244–248.

Seth RS. 2006. The importance of fibre straightness for pulp strength. *Pulp & Paper Canada*, 107:34–42.

Torgnysdotter A, Kulachenko A, Gradin P, and Wågberg L. 2007a. Fiber/fiber crosses: Finite element modeling and comparison with experiment. *Journal of Composite Materials*, 41: 1603–1618.

Torgnysdotter A, Kulachenko A, Gradin P, and Wågberg L. 2007b. The link between the fiber contact zone and the physical properties of paper: A way to control paper properties. *Journal of Composite Materials*, 41:1619–1633.

Uesaka T. *Handbook of physical testing of paper*, chapter Determination of fiber-fiber bond properties, pages 379–402. CRC Press, 1984.

Wang JL, Parnianpour M, Shirazi-Adl A, and Engin A E. 1997. Failure criterion of collagen fiber: viscoelastic behavior simulated by using load control data. *Theoretical and Applied Fracture Mechanics*, 27:1–12.

4 Appendix

The code was written by the author. Based on preliminary work done by Peter Oswald and the book Puri, 2011

4.1 Source code for modelling fibre-fibre joints

```
#
#       parameters [mm bzw. TPa bzw. mikroN]
#



#       geometry adhesive part

AdhesiveThickness=0.00001    # model behavior of adhesive joints
                             # Results in article ‘‘A Proposed Failure
                             # Mechanism for Pulp Fiber-Fiber Joints‘‘are
                             # calculated without cohesive elements

#       geometry fixed fiber

FixedFiberWidth=0.032
FixedFiberThickness=0.00745/2
FixedFiberLength=1.0
FixedFiberFibrilAngle=10

#       geometry strained fiber

StrainedFiberWidth=0.032
StrainedFiberThickness=0.00745/2
StrainedFiberLength=1.0
StrainedFiberOverhang=StrainedFiberLength/2
StrainedFiberFibrilAngle=10

#       connection

AngelFiber=90                      #encolsing angle of fibers
DistanceFiber=FixedFiberLength/2 #position of the strained fiber

#       Mesh
```

4 Appendix

```
MeshSize=0.01
MinMeshSize=0.1
deviationFactor=0.1

#       Fiber Material Properties

FiberElasticModuli1=30000000000
FiberElasticModuli2=FiberElasticModuli1/11
FiberElasticModuli3=FiberElasticModuli1/11

FiberPoissonsRatio1=0.022
FiberPoissonsRatio2=0.022
FiberPoissonsRatio3=0.39

FiberShearModuli1=FiberElasticModuli1/23
FiberShearModuli2=FiberElasticModuli1/23
FiberShearModuli3=FiberElasticModuli2/(2*(1+FiberPoissonsRatio3))

#       Adhesive Material Properties

GTC=7
deltaratio=0.0005
deltafail=0.00005*MeshSize
Keff=2*GTC/(deltaratio*deltafail*deltafail)

QuadeDamage=50
DamageEnergie=300

#       Force

R_Force = 0.0005          # defines the force application area
R_Force2 = 0.0025
Center_Distance = (FixedFiberWidth/2 + 0.003)/sin(AngelFiber*pi/180)

Force=0.000330 * 1000000
AngelForce=AngelFiber     #AngelForce is 90 or AngelFiber



FiberJob_name = 'FiberJob_FA120_MODE1_V_L2'    # Abaqus Job name

#----------------------------------------------------------------------------

from abaqus import *
from abaqusConstants import *
session.viewports['Viewport:␣1'].makeCurrent()
session.viewports['Viewport:␣1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
from driverUtils import executeOnCaeStartup
executeOnCaeStartup()
Mdb()
```

```
mdb.models.changeKey(fromName='Model-1', toName='Fiber-Fiber␣Joint')
fiberModel = mdb.models['Fiber-Fiber␣Joint']

#
#       define Material Properties
#
# Adhesive Material:
adhesiveMaterial = fiberModel.Material(name='AdhesiveMaterial')
adhesiveMaterial.Elastic(type=TRACTION, table=((Keff, Keff, Keff), ))
adhesiveMaterial.QuadeDamageInitiation(table=((QuadeDamage, QuadeDamage,
    QuadeDamage), ))
adhesiveMaterial.quadeDamageInitiation.DamageEvolution(
    type=ENERGY, modeMixRatio=TRACTION, table=((DamageEnergie, ), ))
# Fiber Composite Material:
fiberMaterial = fiberModel.Material(name='FiberCompositeMaterial')
fiberMaterial.Elastic(type=ENGINEERING_CONSTANTS, table=((
    FiberElasticModuli1, FiberElasticModuli2, FiberElasticModuli3,
    FiberPoissonsRatio1, FiberPoissonsRatio2, FiberPoissonsRatio3,
    FiberShearModuli1, FiberShearModuli2, FiberShearModuli3),))

# Fiber Composite Material Isotrop:
fiberMaterial_iso = fiberModel.Material(name='FiberCompositeMaterial_iso')
fiberMaterial_iso.Elastic(table=((FiberElasticModuli1, 0.3), ))


#
#       create "Adhesive Part"
#

AdhesiveSketch = fiberModel.ConstrainedSketch(name='AdhesiveSketch', sheetSize
    =100)

g, v, d, c = AdhesiveSketch.geometry, AdhesiveSketch.vertices, AdhesiveSketch.
    dimensions, AdhesiveSketch.constraints
AdhesiveSketch.setPrimaryObject(option=STANDALONE)

AdhesiveSketch.Line(point1=(0.0, 0.0), point2=(0.0, FixedFiberWidth-2*
    FixedFiberThickness))
AdhesiveSketch.Line(point1=(0, FixedFiberWidth-2*FixedFiberThickness), point2
    =(StrainedFiberWidth-2*StrainedFiberThickness, FixedFiberWidth-2*
    FixedFiberThickness))
AdhesiveSketch.HorizontalConstraint(entity=g.findAt(((StrainedFiberWidth-2*
    StrainedFiberThickness)/2, FixedFiberWidth-2*FixedFiberThickness)),
    addUndoState=False)
AdhesiveSketch.Line(point1=(StrainedFiberWidth-2*StrainedFiberThickness,
    FixedFiberWidth-2*FixedFiberThickness), point2=(StrainedFiberWidth-2*
    StrainedFiberThickness, 0.0))
AdhesiveSketch.Line(point1=(StrainedFiberWidth-2*StrainedFiberThickness, 0.0),
     point2=(0.0, 0.0))
AdhesiveSketch.HorizontalConstraint(entity=g.findAt(((StrainedFiberWidth-2*
    StrainedFiberThickness)/2, 0.0)), addUndoState=False)
AdhesiveSketch.ParallelConstraint(entity1=g.findAt((0, (FixedFiberWidth-2*
    FixedFiberThickness)/2)), entity2=g.findAt((StrainedFiberWidth-2*
    StrainedFiberThickness, (FixedFiberWidth-2*FixedFiberThickness)/2)))
```

```
AdhesiveSketch.FixedConstraint(entity=v.findAt((0.0, 0.0)))
AdhesiveSketch.DistanceDimension(entity1=g.findAt((0.0, (FixedFiberWidth-2*
    FixedFiberThickness)/2)),
                                 entity2=g.findAt((StrainedFiberWidth-2*
                                     StrainedFiberThickness, (FixedFiberWidth
                                     -2*FixedFiberThickness)/2)),
                                 textPoint=(0.02, 0.03), value=
                                     StrainedFiberWidth-2*
                                     StrainedFiberThickness)
AdhesiveSketch.DistanceDimension(entity1=g.findAt(((StrainedFiberWidth-2*
    StrainedFiberThickness)/2, FixedFiberWidth-2*FixedFiberThickness)),
                                 entity2=g.findAt(((StrainedFiberWidth-2*
                                     StrainedFiberThickness)/2, 0.0)),
                                 textPoint=(0.05, 0.01),value=FixedFiberWidth
                                     -2*FixedFiberThickness)


AdhesiveSketch.AngularDimension(line1=g.findAt((0.0, (FixedFiberWidth-2*
    FixedFiberThickness)/2)),
                                line2=g.findAt(((StrainedFiberWidth-2*
                                    StrainedFiberThickness)/2, 0.0)),
                                textPoint=(0.05, 0.05), value=AngelFiber)


AdhesivePart = fiberModel.Part(name='Adhesive', dimensionality=THREE_D, type=
    DEFORMABLE_BODY)
AdhesivePart.BaseSolidExtrude(sketch=AdhesiveSketch, depth=AdhwsiveThickness)


v1 = AdhesivePart.vertices
d1 = AdhesivePart.datums
f1=AdhesivePart.faces

AdhesivePart.DatumPointByMidPoint(point1=v1[0], point2=v1[5])
AdhesivePart.DatumPointByOffset(point=d1[2], vector=(0, -(FixedFiberWidth-2*
    FixedFiberThickness)/2, 0.05))
AdhesivePart.DatumPointByOffset(point=d1[2], vector=(0, -(FixedFiberWidth-2*
    FixedFiberThickness)/2, -0.05))
AdhesivePart.DatumPlaneByThreePoints(point1=d1[3], point2=d1[2], point3=d1[4])
AdhesivePart.DatumAxisByTwoPoint(point1=d1[4], point2=d1[3])
AdhesivePart.DatumPlaneByRotation(plane=f1.findAt(coordinates=((
    StrainedFiberWidth-2*StrainedFiberThickness)/2, 0,
        AdhwsiveThickness/2)), axis=d1[6], angle=-(90-AngelFiber))
AdhesivePart.DatumPlaneByOffset(plane=d1[7], point=d1[2])



#
#       create one half of the fixed Fiber
#

fixedFiberSketch = fiberModel.ConstrainedSketch(name='fixedFiberSketch',
    sheetSize=100)
```

```
fixedFiberSketch.rectangle(point1=(0, 0), point2=(FixedFiberWidth,
    FixedFiberThickness))

fixedFiberPart = fiberModel.Part(name='fixed␣Fiber␣Half', dimensionality=
    THREE_D,
    type=DEFORMABLE_BODY)

fixedFiberPart.BaseSolidExtrude(sketch=fixedFiberSketch, depth=
    FixedFiberLength)

fixedFiberPart.Round(radius=FixedFiberThickness, edgeList=(fixedFiberPart.
    edges[1], fixedFiberPart.edges[5]))

#
#       partition the "fixed Fiber Half" into 3 sections for meshing
#

c = fixedFiberPart.cells

pickedCells = c.getSequenceFromMask(mask=('[#1␣]', ), )
e, v1, d1 = fixedFiberPart.edges, fixedFiberPart.vertices, fixedFiberPart.
    datums
fixedFiberPart.PartitionCellByPlanePointNormal(point=v1[3], normal=e[4], cells
    =pickedCells)

pickedCells = c.getSequenceFromMask(mask=('[#2␣]', ), )
e1, v2, d2 = fixedFiberPart.edges, fixedFiberPart.vertices, fixedFiberPart.
    datums
fixedFiberPart.PartitionCellByPlanePointNormal(point=v2[7], normal=e1[4],
    cells=pickedCells)


#
#       create one half of the strained Fiber
#

strainedFiberSketch = fiberModel.ConstrainedSketch(name='strainedFiberSketch',
     sheetSize=100)

strainedFiberSketch.rectangle(point1=(0, 0), point2=(StrainedFiberWidth,
    StrainedFiberThickness))

strainedFiberPart = fiberModel.Part(name='strained␣Fiber␣Half', dimensionality
    =THREE_D,
    type=DEFORMABLE_BODY)

strainedFiberPart.BaseSolidExtrude(sketch=strainedFiberSketch, depth=
    StrainedFiberLength)

strainedFiberPart.Round(radius=StrainedFiberThickness, edgeList=(
    strainedFiberPart.edges[1], strainedFiberPart.edges[5]))

strainedFiberPart.DatumPointByCoordinate(coords=(StrainedFiberWidth/2,
    StrainedFiberThickness/2, 0.0))
```

```
strainedFiberPart.DatumPointByCoordinate(coords=(StrainedFiberWidth-
    StrainedFiberThickness/2, StrainedFiberThickness/2, 0.0))
strainedFiberPart.DatumPointByCoordinate(coords=(StrainedFiberThickness/2,
    StrainedFiberThickness/2, 0.0))


#
#       partition the "strained Fiber Half" into 3 sections for meshing
#

c = strainedFiberPart.cells

pickedCells = c.getSequenceFromMask(mask=('[#1␣]', ), )
e, v1, d1 = strainedFiberPart.edges, strainedFiberPart.vertices,
    strainedFiberPart.datums
strainedFiberPart.PartitionCellByPlanePointNormal(point=v1[3], normal=e[4],
    cells=pickedCells)

pickedCells = c.getSequenceFromMask(mask=('[#2␣]', ), )
e1, v2, d2 = strainedFiberPart.edges, strainedFiberPart.vertices,
    strainedFiberPart.datums
strainedFiberPart.PartitionCellByPlanePointNormal(point=v2[7], normal=e1[4],
    cells=pickedCells)

#
#       defining Sections
#

fiberModel.CohesiveSection(name='AdhesiveSection',
    material='AdhesiveMaterial', response=TRACTION_SEPARATION,
    outOfPlaneThickness=None)

fiberModel.HomogeneousSolidSection(name='FiberCompositeSection',
    material='FiberCompositeMaterial')

#
#       assign section to parts
#

adhesive_region = (AdhesivePart.cells,)
AdhesivePart.SectionAssignment(region=adhesive_region, sectionName='
    AdhesiveSection', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='')


fixedFiber_region = (fixedFiberPart.cells,)
fixedFiberPart.SectionAssignment(region=fixedFiber_region, sectionName='
    FiberCompositeSection', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='')


strainedFiber_region = (strainedFiberPart.cells, )
strainedFiberPart.SectionAssignment(region=strainedFiber_region, sectionName='
    FiberCompositeSection', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='')
```

```
#
#       assign datumCSYS to 'fixed Fiber Half'
#

v1 = fixedFiberPart.vertices
fixedFiberPart.DatumCsysByThreePoints(origin=v1[4], point1=v1[7], point2=v1
    [1],
    name='RectangularCSYS', coordSysType=CARTESIAN)


#
#       assign datumCSYS to 'strained Fiber Half'
#


v1 = strainedFiberPart.vertices
strainedFiberPart.DatumCsysByThreePoints(origin=v1[4], point1=v1[7], point2=v1
    [1],
    name='RectangularCSYS', coordSysType=CARTESIAN)


#
#       assign material orientation to "fixed Fiber Half"
#

c = fixedFiberPart.cells
cells = c.getSequenceFromMask(mask=('[#7␣]', ), )
region = regionToolset.Region(cells=cells)
orientation = fixedFiberPart.datums[6]
fixedFiberPart.MaterialOrientation(
    region=region, orientationType=SYSTEM, localCsys=orientation, fieldName=''
        ,
    axis=AXIS_3, additionalRotationType=ROTATION_ANGLE, angle=
        FixedFiberFibrilAngle,
    additionalRotationField='', stackDirection=STACK_3)


#
#       assign material orientation to "strained Fiber Half"
#

c = strainedFiberPart.cells
cells = c.getSequenceFromMask(mask=('[#7␣]', ), )
region = regionToolset.Region(cells=cells)
orientation = strainedFiberPart.datums[9]
strainedFiberPart.MaterialOrientation(
    region=region, orientationType=SYSTEM, localCsys=orientation, fieldName=''
        ,
    axis=AXIS_3, additionalRotationType=ROTATION_ANGLE, angle=
        StrainedFiberFibrilAngle,
    additionalRotationField='', stackDirection=STACK_3)
```

```
#
#       Create the Assembly
#

fiberAssembly = fiberModel.rootAssembly



#
#       Add fixed Fiber to the Assembly
#


fiberAssembly.DatumCsysByDefault(CARTESIAN)

fiberAssembly.Instance(name='fixed␣Fiber␣Half-1', part=fixedFiberPart,
    dependent=OFF)
fiberAssembly.Instance(name='fixed␣Fiber␣Half-2', part=fixedFiberPart,
    dependent=OFF)

#
#       create Surface for Tie Constrained on fixed Fibers
#

a = fiberAssembly

s1 = fiberAssembly.instances['fixed␣Fiber␣Half-1'].faces
side1Faces1 = s1.findAt(((FixedFiberThickness/2, 0.0, FixedFiberLength/2), ),
    ((FixedFiberWidth/2, 0.0,
    FixedFiberLength/2), ), ((FixedFiberWidth-FixedFiberThickness/2, 0.0,
        FixedFiberLength/2), ))
a.Surface(side1Faces=side1Faces1, name='Surf-1')

s1 = a.instances['fixed␣Fiber␣Half-2'].faces
side1Faces1 = s1.findAt(((FixedFiberThickness/2, 0.0, FixedFiberLength/2), ),
    ((FixedFiberWidth/2, 0.0,
    FixedFiberLength/2), ), ((FixedFiberWidth-FixedFiberThickness/2, 0.0,
        FixedFiberLength/2), ))
a.Surface(side1Faces=side1Faces1, name='Surf-2')


#
#       create constraints to assamble fixed fiber
#

a1 = fiberAssembly
f1 = a1.instances['fixed␣Fiber␣Half-2'].faces
f2 = a1.instances['fixed␣Fiber␣Half-1'].faces
a1.FaceToFace(movablePlane=f1[1], fixedPlane=f2[1], flip=OFF, clearance=0.0)
a1.FaceToFace(movablePlane=f1[3], fixedPlane=f2[3], flip=ON, clearance=0.0)
e1 = a1.instances['fixed␣Fiber␣Half-2'].edges
e2 = a1.instances['fixed␣Fiber␣Half-1'].edges
a1.EdgeToEdge(movableAxis=e1[0], fixedAxis=e2[10], flip=OFF)
```

```
#
#        Add Adhesive to the Assembly
#

a1 = fiberAssembly
a1.Instance(name='Adhesive-1', part=AdhesivePart, dependent=OFF)
p1 = a1.instances['Adhesive-1']


#
#        create constraints to assamble Adhesive
#

a1 = fiberAssembly
f1 = a1.instances['fixed␣Fiber␣Half-1'].faces
f2 = a1.instances['Adhesive-1'].faces
d1 = a1.instances['Adhesive-1'].datums

a1.FaceToFace(movablePlane=f2[4], fixedPlane=f1[9], flip=ON, clearance=0)
a1.FaceToFace(movablePlane=f2[1], fixedPlane=f1[0], flip=ON, clearance=-(
    FixedFiberWidth-2*FixedFiberThickness))
a1.FaceToFace(movablePlane=d1[5], fixedPlane=f1[1], flip=OFF, clearance=-
    DistanceFiber)


#
#        Add strained Fiber to the Assembly
#

a = fiberAssembly
session.viewports['Viewport:␣1'].setValues(displayedObject=a)
a.DatumCsysByDefault(CARTESIAN)
p=strainedFiberPart
a.Instance(name='strained␣Fiber␣Half-1', part=strainedFiberPart, dependent=OFF
    )
a.Instance(name='strained␣Fiber␣Half-2', part=strainedFiberPart, dependent=OFF
    )



#
#        create Surface for Tie Constrained on strained Fibers
#


s1 = a.instances['strained␣Fiber␣Half-1'].faces
side1Faces1 = s1.findAt(((StrainedFiberThickness/2, 0.0, StrainedFiberLength
    /2), ), ((StrainedFiberWidth/2, 0,
    StrainedFiberLength), ), ((StrainedFiberWidth-StrainedFiberThickness/2,
        0.0, StrainedFiberLength/2), ))
a.Surface(side1Faces=side1Faces1, name='Surf-7')

s1 = a.instances['strained␣Fiber␣Half-2'].faces
```

```
side1Faces1 = s1.findAt(((StrainedFiberThickness/2, 0.0, StrainedFiberLength
    /2), ), ((StrainedFiberWidth/2, 0,
    StrainedFiberLength), ), ((StrainedFiberWidth-StrainedFiberThickness/2,
        0.0, StrainedFiberLength/2), ))
a.Surface(side1Faces=side1Faces1, name='Surf-8')


#
#       create constraints to assamble strained Fiber
#

a = fiberAssembly
f1 = a.instances['strained␣Fiber␣Half-1'].faces
f2 = a.instances['Adhesive-1'].faces
f3 = a.instances['fixed␣Fiber␣Half-1'].faces
d2 = a.instances['Adhesive-1'].datums
a.FaceToFace(movablePlane=f1[1], fixedPlane=d2[8], flip=OFF, clearance=
    StrainedFiberOverhang)
a.FaceToFace(movablePlane=f1[4], fixedPlane=f2[2], flip=ON, clearance=0)
a.FaceToFace(movablePlane=f1[9], fixedPlane=f3[9], flip=ON, clearance=0)

f1 = a.instances['strained␣Fiber␣Half-2'].faces
f2 = a.instances['strained␣Fiber␣Half-1'].faces
a.FaceToFace(movablePlane=f1[1], fixedPlane=f2[1], flip=OFF, clearance=0.0)
a.FaceToFace(movablePlane=f1[3], fixedPlane=f2[3], flip=ON, clearance=0.0)
e1 = a.instances['strained␣Fiber␣Half-2'].edges
e2 = a.instances['strained␣Fiber␣Half-1'].edges
a.EdgeToEdge(movableAxis=e1[0], fixedAxis=e2[10], flip=OFF)


#
#       partion the the strained fiber for adding the load
#
a = mdb.models['Fiber-Fiber␣Joint'].rootAssembly
f11 = a.instances['strained␣Fiber␣Half-1'].faces
e11 = a.instances['strained␣Fiber␣Half-1'].edges
t = a.MakeSketchTransform(sketchPlane=f11.findAt(coordinates=(FixedFiberWidth
    /2,
        FixedFiberThickness, DistanceFiber)), sketchUpEdge=e11.findAt(
            coordinates=(FixedFiberWidth/2,
        FixedFiberThickness, DistanceFiber + (StrainedFiberWidth/2 -
            StrainedFiberThickness)/sin(AngelFiber*pi/180))), sketchPlaneSide=
            SIDE1, origin=(FixedFiberWidth/2, FixedFiberThickness,
        DistanceFiber))


s1 = mdb.models['Fiber-Fiber␣Joint'].ConstrainedSketch(name='__profile__',
        sheetSize=2.0, gridSpacing=0.05, transform=t)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=SUPERIMPOSE)

a.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)

s1.CircleByCenterPerimeter(center=(0 , -Center_Distance),
        point1=(R_Force, -Center_Distance ))
```

```
s1.CircleByCenterPerimeter(center=(0 , -Center_Distance),
        point1=(R_Force2, -Center_Distance ))

s1.CircleByCenterPerimeter(center=(0 , -Center_Distance),
        point1=(R_Force+(R_Force2-R_Force)/3, -Center_Distance ))

s1.CircleByCenterPerimeter(center=(0 , -Center_Distance),
        point1=(R_Force+(R_Force2-R_Force)*2/3, -Center_Distance ))




pickedFaces = f11.findAt(((FixedFiberWidth/2, FixedFiberThickness,
    DistanceFiber), ))
a.PartitionFaceBySketch(sketchUpEdge=e11.findAt(coordinates=(FixedFiberWidth
    /2, FixedFiberThickness,  DistanceFiber + (StrainedFiberWidth/2 -
    StrainedFiberThickness)/sin(AngelFiber*pi/180))),
                        faces=pickedFaces, sketch=s1)
s1.unsetPrimaryObject()
del mdb.models['Fiber-Fiber␣Joint'].sketches['__profile__']


c1 = a.instances['strained␣Fiber␣Half-1'].cells
e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedCells = c1.findAt(((FixedFiberWidth/2, FixedFiberThickness,
    DistanceFiber), ))
pickedEdges =(e1.findAt(coordinates=(FixedFiberWidth/2 + (Center_Distance +
    R_Force)*sin(AngelFiber*pi/180), FixedFiberThickness, DistanceFiber - (
    Center_Distance + R_Force)*cos(AngelFiber*pi/180))), )

e2 = a.instances['fixed␣Fiber␣Half-2'].edges
a.PartitionCellByExtrudeEdge(line=e2.findAt(coordinates=(FixedFiberThickness,
    -FixedFiberThickness*3/4, 0.0)), cells=pickedCells, edges=pickedEdges,
    sense=FORWARD)

pickedCells1 = c1.findAt(((FixedFiberWidth/2, FixedFiberThickness,
    DistanceFiber), ))
pickedEdges1 =(e1.findAt(coordinates=(FixedFiberWidth/2 + (Center_Distance +
    R_Force+(R_Force2-R_Force)/3)*sin(AngelFiber*pi/180), FixedFiberThickness,
     DistanceFiber - (Center_Distance + R_Force+(R_Force2-R_Force)/3)*cos(
    AngelFiber*pi/180))), )

e2 = a.instances['fixed␣Fiber␣Half-2'].edges
a.PartitionCellByExtrudeEdge(line=e2.findAt(coordinates=(FixedFiberThickness,
    -FixedFiberThickness*3/4, 0.0)), cells=pickedCells1, edges=pickedEdges1,
    sense=FORWARD)

pickedCells1 = c1.findAt(((FixedFiberWidth/2, FixedFiberThickness,
    DistanceFiber), ))
pickedEdges1 =(e1.findAt(coordinates=(FixedFiberWidth/2 + (Center_Distance +
    R_Force+(R_Force2-R_Force)*2/3)*sin(AngelFiber*pi/180),
    FixedFiberThickness, DistanceFiber - (Center_Distance + R_Force+(R_Force2-
    R_Force)*2/3)*cos(AngelFiber*pi/180))), )
```

```
e2 = a.instances['fixed␣Fiber␣Half-2'].edges
a.PartitionCellByExtrudeEdge(line=e2.findAt(coordinates=(FixedFiberThickness,
    -FixedFiberThickness*3/4, 0.0)), cells=pickedCells1, edges=pickedEdges1,
    sense=FORWARD)




pickedCells1 = c1.findAt(((FixedFiberWidth/2, FixedFiberThickness,
    DistanceFiber), ))
pickedEdges1 =(e1.findAt(coordinates=(FixedFiberWidth/2 + (Center_Distance +
    R_Force+(R_Force2-R_Force)*3/3)*sin(AngelFiber*pi/180),
    FixedFiberThickness, DistanceFiber - (Center_Distance + R_Force+(R_Force2-
    R_Force)*3/3)*cos(AngelFiber*pi/180))), )

e2 = a.instances['fixed␣Fiber␣Half-2'].edges
a.PartitionCellByExtrudeEdge(line=e2.findAt(coordinates=(FixedFiberThickness,
    -FixedFiberThickness*3/4, 0.0)), cells=pickedCells1, edges=pickedEdges1,
    sense=FORWARD)



#
#       creating the Step in which the load is applayed
#

fiberModel.StaticStep   (name='FiberLoad', previous='Initial',
                         timePeriod=1.0, initialInc=0.1,
                         description='Load␣the␣front␣surface␣of␣the␣Fiber')


fiberModel.fieldOutputRequests['F-Output-1'].setValues(variables=('S', 'PE', '
    PEEQ', 'PEMAG', 'LE', 'U', 'RF', 'CF', 'SF', 'NFORC', 'TF', 'CSTRESS', '
    CDISP', 'COORD'))


#
#       create Partions for the seed
#

#       partion the fixed Fiber

a = fiberAssembly
c1 = a.instances['fixed␣Fiber␣Half-1'].cells
cells1 = c1.findAt(((FixedFiberThickness/2, FixedFiberThickness/2, 0), ), ((
    FixedFiberWidth/2, FixedFiberThickness/2, 0), ), ((FixedFiberWidth-
    FixedFiberThickness/2, FixedFiberThickness/2, 0), ))

c2 = a.instances['fixed␣Fiber␣Half-2'].cells
cells2 = c2.findAt(((FixedFiberThickness/2, -FixedFiberThickness/2, 0.0), ),
    ((FixedFiberWidth/2, -FixedFiberThickness/2, 0), ),
                   ((FixedFiberWidth-FixedFiberThickness/2, -
                       FixedFiberThickness/2, 0), ))

pickedCells = cells1+cells2
```

```
f11 = a.instances['Adhesive-1'].faces
a.PartitionCellByExtendFace(extendFace=f11.findAt(coordinates=(FixedFiberWidth
    /2,FixedFiberThickness + AdhwsiveThickness/2, DistanceFiber-(
    StrainedFiberWidth/2-StrainedFiberThickness)/sin(AngelFiber*pi/180))),
    cells=pickedCells)

c1 = a.instances['fixed␣Fiber␣Half-1'].cells
cells1 = c1.findAt(((FixedFiberThickness/2, FixedFiberThickness/2,
    FixedFiberLength), ), ((FixedFiberWidth/2, FixedFiberThickness/2,
    FixedFiberLength), ), ((FixedFiberWidth-FixedFiberThickness/2,
    FixedFiberThickness/2, FixedFiberLength), ))

c2 = a.instances['fixed␣Fiber␣Half-2'].cells
cells2 = c2.findAt(((FixedFiberThickness/2, -FixedFiberThickness/2,
    FixedFiberLength), ), ((FixedFiberWidth/2, -FixedFiberThickness/2,
    FixedFiberLength), ), ((FixedFiberWidth-FixedFiberThickness/2, -
    FixedFiberThickness/2, FixedFiberLength), ))

pickedCells = cells1+cells2
f11 = a.instances['Adhesive-1'].faces
a.PartitionCellByExtendFace(extendFace=f11.findAt(coordinates=(FixedFiberWidth
    /2,FixedFiberThickness + AdhwsiveThickness/2, DistanceFiber+(
    StrainedFiberWidth/2-StrainedFiberThickness)/sin(AngelFiber*pi/180))),
    cells=pickedCells)

#       partion the strained Fiber

c1 = a.instances['strained␣Fiber␣Half-1'].cells
cells1 = c1.findAt(((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
    StrainedFiberThickness/2, FixedFiberLength/2), ),
                   ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                       StrainedFiberThickness/2, FixedFiberLength/2+(
                       StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                       AngelFiber*pi/180))), ),
                   ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                       StrainedFiberThickness/2, FixedFiberLength/2-(
                       StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                       AngelFiber*pi/180))), ))
c2 = a.instances['strained␣Fiber␣Half-2'].cells
cells2 = c2.findAt(((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
    StrainedFiberThickness*3/2, FixedFiberLength/2), ),
                   ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                       StrainedFiberThickness*3/2, FixedFiberLength/2+(
                       StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                       AngelFiber*pi/180))), ),
                   ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                       StrainedFiberThickness*3/2, FixedFiberLength/2-(
                       StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                       AngelFiber*pi/180))), ))

pickedCells = cells1+cells2
f11 = a.instances['Adhesive-1'].faces
a.PartitionCellByExtendFace(extendFace=f11.findAt(coordinates=(
    FixedFiberThickness, FixedFiberThickness+AdhwsiveThickness/2,
    FixedFiberLength/2+FixedFiberWidth/2*cos(AngelFiber*pi/180))), cells=
    pickedCells)

c1 = a.instances['strained␣Fiber␣Half-1'].cells
cells1 = c1.findAt(((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
    StrainedFiberThickness/2, FixedFiberLength/2), ),
                   ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                       StrainedFiberThickness/2, FixedFiberLength/2+(
                       StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                       AngelFiber*pi/180))), ),
                   ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                       StrainedFiberThickness/2, FixedFiberLength/2-(
                       StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                       AngelFiber*pi/180))), ))
c2 = a.instances['strained␣Fiber␣Half-2'].cells
cells2 = c2.findAt(((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
    StrainedFiberThickness*3/2, FixedFiberLength/2), ),
                   ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                       StrainedFiberThickness*3/2, FixedFiberLength/2+(
                       StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                       AngelFiber*pi/180))), ),
                   ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                       StrainedFiberThickness*3/2, FixedFiberLength/2-(
                       StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                       AngelFiber*pi/180))), ))

pickedCells = cells1+cells2
f11 = a.instances['Adhesive-1'].faces
a.PartitionCellByExtendFace(extendFace=f11.findAt(coordinates=(FixedFiberWidth
    -FixedFiberThickness, FixedFiberThickness+AdhwsiveThickness/2,
    FixedFiberLength/2-FixedFiberWidth/2*cos(AngelFiber*pi/180))), cells=
    pickedCells)

#
#       create Surface for Tie Constrained in the connection


s1 = a.instances['fixed␣Fiber␣Half-1'].faces
side1Faces1 = s1.findAt(((FixedFiberWidth/2, FixedFiberThickness,
    DistanceFiber), ))
a.Surface(side1Faces=side1Faces1, name='Surf-3')

s1 = a.instances['strained␣Fiber␣Half-1'].faces
side1Faces1 = s1.findAt(((FixedFiberWidth/2, FixedFiberThickness,
    DistanceFiber), ))
a.Surface(side1Faces=side1Faces1, name='Surf-6')



#
#       Mesh the Assembly
#

a = fiberAssembly
```

```
partInstances =(a.instances['fixed␣Fiber␣Half-1'], a.instances['fixed␣Fiber␣
    Half-2'], )
a.seedPartInstance(regions=partInstances, size=MeshSize, deviationFactor=
    deviationFactor,
                   minSizeFactor=MinMeshSize)



e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedEdges = e1.findAt(((FixedFiberWidth/2 + (Center_Distance + R_Force2)*sin
    (AngelFiber*pi/180), FixedFiberThickness, DistanceFiber - (Center_Distance
     + R_Force2)*cos(AngelFiber*pi/180)), ))
a.seedEdgeByNumber(edges=pickedEdges, number=8, constraint=FINER)

e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedEdges = e1.findAt(((FixedFiberWidth/2 + (Center_Distance + R_Force+(
    R_Force2-R_Force)*2/3)*sin(AngelFiber*pi/180), FixedFiberThickness,
    DistanceFiber - (Center_Distance + R_Force+(R_Force2-R_Force)*2/3)*cos(
    AngelFiber*pi/180)), ))
a.seedEdgeByNumber(edges=pickedEdges, number=8, constraint=FINER)

e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedEdges = e1.findAt(((FixedFiberWidth/2 + (Center_Distance + R_Force+(
    R_Force2-R_Force)*1/3)*sin(AngelFiber*pi/180), FixedFiberThickness,
    DistanceFiber - (Center_Distance + R_Force+(R_Force2-R_Force)*1/3)*cos(
    AngelFiber*pi/180)), ))
a.seedEdgeByNumber(edges=pickedEdges, number=8, constraint=FINER)

e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedEdges = e1.findAt(((FixedFiberWidth/2 + (Center_Distance + R_Force)*sin(
    AngelFiber*pi/180), FixedFiberThickness, DistanceFiber - (Center_Distance
    + R_Force)*cos(AngelFiber*pi/180)), ))
a.seedEdgeByNumber(edges=pickedEdges, number=8, constraint=FINER)


e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedEdges = e1.findAt(((FixedFiberWidth/2 + (Center_Distance + R_Force2)*sin
    (AngelFiber*pi/180), FixedFiberThickness+StrainedFiberThickness,
    DistanceFiber - (Center_Distance + R_Force2)*cos(AngelFiber*pi/180)), ))
a.seedEdgeByNumber(edges=pickedEdges, number=8, constraint=FINER)

e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedEdges = e1.findAt(((FixedFiberWidth/2 + (Center_Distance + R_Force+(
    R_Force2-R_Force)*2/3)*sin(AngelFiber*pi/180), FixedFiberThickness+
    StrainedFiberThickness, DistanceFiber - (Center_Distance + R_Force+(
    R_Force2-R_Force)*2/3)*cos(AngelFiber*pi/180)), ))
a.seedEdgeByNumber(edges=pickedEdges, number=8, constraint=FINER)

e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedEdges = e1.findAt(((FixedFiberWidth/2 + (Center_Distance + R_Force+(
    R_Force2-R_Force)*1/3)*sin(AngelFiber*pi/180), FixedFiberThickness+
    StrainedFiberThickness, DistanceFiber - (Center_Distance + R_Force+(
    R_Force2-R_Force)*1/3)*cos(AngelFiber*pi/180)), ))
a.seedEdgeByNumber(edges=pickedEdges, number=8, constraint=FINER)
```

```
e1 = a.instances['strained␣Fiber␣Half-1'].edges
pickedEdges = e1.findAt(((FixedFiberWidth/2 + (Center_Distance + R_Force)*sin(
    AngelFiber*pi/180), FixedFiberThickness+StrainedFiberThickness,
    DistanceFiber - (Center_Distance + R_Force)*cos(AngelFiber*pi/180)), ))
a.seedEdgeByNumber(edges=pickedEdges, number=8, constraint=FINER)



partInstances =(a.instances['fixed␣Fiber␣Half-1'], a.instances['fixed␣Fiber␣
    Half-2'], )
a.generateMesh(regions=partInstances)


partInstances =(a.instances['strained␣Fiber␣Half-2'], a.instances['strained␣
    Fiber␣Half-1'], )
a.seedPartInstance(regions=partInstances, size=MeshSize, deviationFactor=
    deviationFactor,
        minSizeFactor=MinMeshSize)


partInstances =(a.instances['strained␣Fiber␣Half-1'], a.instances['strained␣
    Fiber␣Half-2'], )
a.generateMesh(regions=partInstances)




#       Meshtype fixed fiber

c1 = a.instances['fixed␣Fiber␣Half-1'].cells
cells1 = c1.findAt(((FixedFiberThickness/2, FixedFiberThickness/2, 0.0), ), ((
    FixedFiberWidth/2, FixedFiberThickness/2, 0.0), ), ((FixedFiberWidth-
    FixedFiberThickness/2, FixedFiberThickness/2, 0.0), ),
                   ((FixedFiberThickness/2, FixedFiberThickness/2,
                       DistanceFiber), ), ((FixedFiberWidth/2,
                       FixedFiberThickness/2, DistanceFiber), ), ((
                       FixedFiberWidth-FixedFiberThickness/2,
                       FixedFiberThickness/2, DistanceFiber), ),
                   ((FixedFiberWidth/2, FixedFiberThickness/2,
                       FixedFiberLength), ), ((FixedFiberWidth/2,
                       FixedFiberThickness/2, FixedFiberLength), ), ((
                       FixedFiberWidth/2, FixedFiberThickness/2,
                       FixedFiberLength), ))

c2 = a.instances['fixed␣Fiber␣Half-2'].cells
cells2 = c2.findAt(((FixedFiberThickness/2, -FixedFiberThickness/2, 0.0), ),
    ((FixedFiberWidth/2, -FixedFiberThickness/2, 0.0), ), ((FixedFiberWidth-
    FixedFiberThickness/2, -FixedFiberThickness/2, 0.0), ),
                   ((FixedFiberThickness/2, -FixedFiberThickness/2,
                       DistanceFiber), ), ((FixedFiberWidth/2, -
                       FixedFiberThickness/2, DistanceFiber), ), ((
                       FixedFiberWidth-FixedFiberThickness/2, -
                       FixedFiberThickness/2, DistanceFiber), ),
                    ((FixedFiberWidth/2, -FixedFiberThickness/2,
                        FixedFiberLength), ), ((FixedFiberWidth/2, -
                        FixedFiberThickness/2, FixedFiberLength), ), ((
                        FixedFiberWidth/2, -FixedFiberThickness/2,
                        FixedFiberLength), ))

pickedRegions =((cells1+cells2), )

elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=C3D8R, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D8R, elemLibrary=STANDARD)

a.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
    elemType3))



#       Meshtype strained fiber

c1 = a.instances['strained␣Fiber␣Half-1'].cells
cells1 = c1.findAt(((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
    StrainedFiberThickness/2, DistanceFiber+(StrainedFiberWidth-
    StrainedFiberThickness)/(2*sin(AngelFiber*pi/180))), ),
                    ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                        StrainedFiberThickness/2, DistanceFiber), ),
                    ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                        StrainedFiberThickness/2, DistanceFiber-(
                        StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                        AngelFiber*pi/180))), ),
                    ((FixedFiberWidth/2-(StrainedFiberOverhang)*sin(AngelFiber*
                        pi/180), FixedFiberThickness+AdhwsiveThickness+
                        StrainedFiberThickness/2, DistanceFiber+(
                        StrainedFiberOverhang)*cos(AngelFiber*pi/180)), ),
                    ((FixedFiberWidth/2-(StrainedFiberOverhang)*sin(AngelFiber*
                        pi/180)-(StrainedFiberWidth/2-StrainedFiberThickness/2)
                        *cos(AngelFiber*pi/180), FixedFiberThickness+
                        AdhwsiveThickness+StrainedFiberThickness/2,
                        DistanceFiber+(StrainedFiberOverhang)*cos(AngelFiber*pi
                        /180)-(StrainedFiberWidth-StrainedFiberThickness)/2*sin
                        (AngelFiber*pi/180)), ),
                    ((FixedFiberWidth/2-(StrainedFiberOverhang)*sin(AngelFiber*
                        pi/180)+(StrainedFiberWidth/2-StrainedFiberThickness/2)
                        *cos(AngelFiber*pi/180), FixedFiberThickness+
                        AdhwsiveThickness+StrainedFiberThickness/2,
                        DistanceFiber+(StrainedFiberOverhang)*cos(AngelFiber*pi
                        /180)+(StrainedFiberWidth-StrainedFiberThickness)/2*sin
                        (AngelFiber*pi/180)), ),
                    ((FixedFiberWidth/2+sin(AngelFiber*pi/180)*(
                        StrainedFiberLength-StrainedFiberOverhang),
                        FixedFiberThickness+AdhwsiveThickness+
                        StrainedFiberThickness/2, DistanceFiber-cos(AngelFiber*
                        pi/180)*(StrainedFiberLength-StrainedFiberOverhang)), )
                        ,
                    ((FixedFiberWidth/2+sin(AngelFiber*pi/180)*(
                        StrainedFiberLength-StrainedFiberOverhang)+cos(
                            AngelFiber*pi/180)*((StrainedFiberWidth-
                            StrainedFiberThickness)/2), FixedFiberThickness+
                            AdhwsiveThickness+StrainedFiberThickness/2,
                            DistanceFiber-cos(AngelFiber*pi/180)*(
                            StrainedFiberLength-StrainedFiberOverhang)+sin(
                            AngelFiber*pi/180)*((StrainedFiberWidth-
                            StrainedFiberThickness)/2)), ),
                        ((FixedFiberWidth/2+sin(AngelFiber*pi/180)*(
                            StrainedFiberLength-StrainedFiberOverhang)-cos(
                            AngelFiber*pi/180)*((StrainedFiberWidth-
                            StrainedFiberThickness)/2), FixedFiberThickness+
                            AdhwsiveThickness+StrainedFiberThickness/2,
                            DistanceFiber-cos(AngelFiber*pi/180)*(
                            StrainedFiberLength-StrainedFiberOverhang)-sin(
                            AngelFiber*pi/180)*((StrainedFiberWidth-
                            StrainedFiberThickness)/2)), ))


c2 = a.instances['strained␣Fiber␣Half-2'].cells
cells2 = c2.findAt(((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
    StrainedFiberThickness*3/2, DistanceFiber+(StrainedFiberWidth-
    StrainedFiberThickness)/(2*sin(AngelFiber*pi/180))), ),
                        ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                            StrainedFiberThickness*3/2, DistanceFiber), ),
                        ((FixedFiberWidth/2, FixedFiberThickness+AdhwsiveThickness+
                            StrainedFiberThickness*3/2, DistanceFiber-(
                            StrainedFiberWidth-StrainedFiberThickness)/(2*sin(
                            AngelFiber*pi/180))), ),
                        ((FixedFiberWidth/2-(StrainedFiberOverhang)*sin(AngelFiber*
                            pi/180), FixedFiberThickness+AdhwsiveThickness+
                            StrainedFiberThickness*3/2, DistanceFiber+(
                            StrainedFiberOverhang)*cos(AngelFiber*pi/180)), ),
                        ((FixedFiberWidth/2-(StrainedFiberOverhang)*sin(AngelFiber*
                            pi/180)-(StrainedFiberWidth/2-StrainedFiberThickness/2)
                            *cos(AngelFiber*pi/180), FixedFiberThickness+
                            AdhwsiveThickness+StrainedFiberThickness*3/2,
                            DistanceFiber+(StrainedFiberOverhang)*cos(AngelFiber*pi
                            /180)-(StrainedFiberWidth-StrainedFiberThickness)/2*sin
                            (AngelFiber*pi/180)), ),
                        ((FixedFiberWidth/2-(StrainedFiberOverhang)*sin(AngelFiber*
                            pi/180)+(StrainedFiberWidth/2-StrainedFiberThickness/2)
                            *cos(AngelFiber*pi/180), FixedFiberThickness+
                            AdhwsiveThickness+StrainedFiberThickness*3/2,
                            DistanceFiber+(StrainedFiberOverhang)*cos(AngelFiber*pi
                            /180)+(StrainedFiberWidth-StrainedFiberThickness)/2*sin
                            (AngelFiber*pi/180)), ),
                        ((FixedFiberWidth/2+sin(AngelFiber*pi/180)*(
                            StrainedFiberLength-StrainedFiberOverhang),
                            FixedFiberThickness+AdhwsiveThickness+
                            StrainedFiberThickness*3/2, DistanceFiber-cos(
                            AngelFiber*pi/180)*(StrainedFiberLength-
                            StrainedFiberOverhang)), ),
                        ((FixedFiberWidth/2+sin(AngelFiber*pi/180)*(
                            StrainedFiberLength-StrainedFiberOverhang)+cos(
                            AngelFiber*pi/180)*((StrainedFiberWidth-
                        StrainedFiberThickness)/2), FixedFiberThickness+
                        AdhwsiveThickness+StrainedFiberThickness*3/2,
                        DistanceFiber-cos(AngelFiber*pi/180)*(
                        StrainedFiberLength-StrainedFiberOverhang)+sin(
                        AngelFiber*pi/180)*((StrainedFiberWidth-
                        StrainedFiberThickness)/2)), ),
                    ((FixedFiberWidth/2+sin(AngelFiber*pi/180)*(
                        StrainedFiberLength-StrainedFiberOverhang)-cos(
                        AngelFiber*pi/180)*((StrainedFiberWidth-
                        StrainedFiberThickness)/2), FixedFiberThickness+
                        AdhwsiveThickness+StrainedFiberThickness*3/2,
                        DistanceFiber-cos(AngelFiber*pi/180)*(
                        StrainedFiberLength-StrainedFiberOverhang)-sin(
                        AngelFiber*pi/180)*((StrainedFiberWidth-
                        StrainedFiberThickness)/2)), ))


pickedRegions =((cells1+cells2), )

elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=C3D8R, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D8R, elemLibrary=STANDARD)

a.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
    elemType3))



#
#       create Tie Constrains between Fiber Halfs
#

a = fiberAssembly
region1=a.surfaces['Surf-1']
region2=a.surfaces['Surf-2']
fiberModel.Tie(name='Constraint␣fixed␣Fiber-Fiber',
    master=region1, slave=region2, constraintEnforcement=SURFACE_TO_SURFACE,
        positionToleranceMethod=COMPUTED, adjust=ON,
    tieRotations=ON, thickness=ON)


region1=a.surfaces['Surf-7']
region2=a.surfaces['Surf-8']
mdb.models['Fiber-Fiber␣Joint'].Tie(name='Constraint␣strained␣Fiber-Fiber',
    master=region1, slave=region2,constraintEnforcement=SURFACE_TO_SURFACE,
        positionToleranceMethod=COMPUTED, adjust=ON,
    tieRotations=ON, thickness=ON)

#
#       create Tie Constrains between fixed Fiber and strained Fiber
#

a = fiberAssembly
region1=a.surfaces['Surf-3']
```

```
region2=a.surfaces['Surf-6']
fiberModel.Tie(name='Constraint␣fixed␣Fiber␣-␣strained␣Fiber',
    master=region1, slave=region2,constraintEnforcement=SURFACE_TO_SURFACE,
        positionToleranceMethod=COMPUTED, adjust=ON,
   tieRotations=ON, thickness=ON)




#
#       Create a Node Set
#
nodes1 = a.surfaces['Surf-3'].nodes
a.Set(nodes=nodes1, name='NodeSet_fixed')

nodes2 = a.surfaces['Surf-6'].nodes
a.Set(nodes=nodes2, name='NodeSet_strained')

#
#       Create a Element Set
#
elm1 = a.surfaces['Surf-3'].elements
a.Set(elements=elm1, name='ElmSet_fixed')

elm2 = a.surfaces['Surf-6'].elements
a.Set(elements=elm2, name='ElmSet_strained')


#       create Boundary Conditions
#
#       first side

a = fiberAssembly

f1 = a.instances['fixed␣Fiber␣Half-1'].faces

fixed_end_face1_half1_pt1_x = FixedFiberThickness*4/5
fixed_end_face1_half1_pt1_y = FixedFiberThickness/2
fixed_end_face1_half1_pt1_z = FixedFiberLength
fixed_end_face1_half1_pt1 = (fixed_end_face1_half1_pt1_x,
    fixed_end_face1_half1_pt1_y,fixed_end_face1_half1_pt1_z)

fixed_end_face1_half1_pt2_x = FixedFiberWidth-FixedFiberThickness*4/5
fixed_end_face1_half1_pt2_y = FixedFiberThickness/2
fixed_end_face1_half1_pt2_z = FixedFiberLength
fixed_end_face1_half1_pt2 = (fixed_end_face1_half1_pt2_x,
    fixed_end_face1_half1_pt2_y,fixed_end_face1_half1_pt2_z)

fixed_end_face1_half1_pt3_x = FixedFiberWidth/2
fixed_end_face1_half1_pt3_y = FixedFiberThickness/2
fixed_end_face1_half1_pt3_z = FixedFiberLength
fixed_end_face1_half1_pt3 = (fixed_end_face1_half1_pt3_x,
    fixed_end_face1_half1_pt3_y,fixed_end_face1_half1_pt3_z)
```

```
faces1 = f1.findAt(((fixed_end_face1_half1_pt1), ), ((
    fixed_end_face1_half1_pt2), ), ((fixed_end_face1_half1_pt3), ))


f2 = a.instances['fixed␣Fiber␣Half-2'].faces

fixed_end_face1_half2_pt1_x = FixedFiberThickness*4/5
fixed_end_face1_half2_pt1_y = -FixedFiberThickness/2
fixed_end_face1_half2_pt1_z = FixedFiberLength
fixed_end_face1_half2_pt1 = (fixed_end_face1_half2_pt1_x,
    fixed_end_face1_half2_pt1_y,fixed_end_face1_half2_pt1_z)

fixed_end_face1_half2_pt2_x = FixedFiberWidth-FixedFiberThickness*4/5
fixed_end_face1_half2_pt2_y = -FixedFiberThickness/2
fixed_end_face1_half2_pt2_z = FixedFiberLength
fixed_end_face1_half2_pt2 = (fixed_end_face1_half2_pt2_x,
    fixed_end_face1_half2_pt2_y,fixed_end_face1_half2_pt2_z)

fixed_end_face1_half2_pt3_x = FixedFiberWidth/2
fixed_end_face1_half2_pt3_y = -FixedFiberThickness/2
fixed_end_face1_half2_pt3_z = FixedFiberLength
fixed_end_face1_half2_pt3 = (fixed_end_face1_half2_pt3_x,
    fixed_end_face1_half2_pt3_y,fixed_end_face1_half2_pt3_z)

faces2 = f2.findAt(((fixed_end_face1_half2_pt1), ), ((
    fixed_end_face1_half2_pt2), ), ((fixed_end_face1_half2_pt3), ))

region = regionToolset.Region(faces=faces1+faces2)
fiberModel.EncastreBC(name='BC-1', createStepName='Initial', region=region)

#
#       create Surface for Reaktionforces
#

a = fiberAssembly


a.Surface(side1Faces=faces1+faces2, name='Surf_rf1')


#       second side

fixed_end_face2_half1_pt1_x = FixedFiberThickness*4/5
fixed_end_face2_half1_pt1_y = FixedFiberThickness/2
fixed_end_face2_half1_pt1_z = 0
fixed_end_face2_half1_pt1 = (fixed_end_face2_half1_pt1_x,
    fixed_end_face2_half1_pt1_y,fixed_end_face2_half1_pt1_z)

fixed_end_face2_half1_pt2_x = FixedFiberWidth-FixedFiberThickness*4/5
fixed_end_face2_half1_pt2_y = FixedFiberThickness/2
fixed_end_face2_half1_pt2_z = 0
fixed_end_face2_half1_pt2 = (fixed_end_face2_half1_pt2_x,
    fixed_end_face2_half1_pt2_y,fixed_end_face2_half1_pt2_z)

fixed_end_face2_half1_pt3_x = FixedFiberWidth/2
```

```
fixed_end_face2_half1_pt3_y = FixedFiberThickness/2
fixed_end_face2_half1_pt3_z = 0
fixed_end_face2_half1_pt3 = (fixed_end_face2_half1_pt3_x,
    fixed_end_face2_half1_pt3_y,fixed_end_face2_half1_pt3_z)

faces1 = f1.findAt(((fixed_end_face2_half1_pt1), ), ((
    fixed_end_face2_half1_pt2), ), ((fixed_end_face2_half1_pt3), ))

fixed_end_face2_half2_pt1_x = FixedFiberThickness*4/5
fixed_end_face2_half2_pt1_y = -FixedFiberThickness/2
fixed_end_face2_half2_pt1_z = 0
fixed_end_face2_half2_pt1 = (fixed_end_face2_half2_pt1_x,
    fixed_end_face2_half2_pt1_y,fixed_end_face2_half2_pt1_z)

fixed_end_face2_half2_pt2_x = FixedFiberWidth-FixedFiberThickness*4/5
fixed_end_face2_half2_pt2_y = -FixedFiberThickness/2
fixed_end_face2_half2_pt2_z = 0
fixed_end_face2_half2_pt2 = (fixed_end_face2_half2_pt2_x,
    fixed_end_face2_half2_pt2_y,fixed_end_face2_half2_pt2_z)

fixed_end_face2_half2_pt3_x = FixedFiberWidth/2
fixed_end_face2_half2_pt3_y = -FixedFiberThickness/2
fixed_end_face2_half2_pt3_z = 0
fixed_end_face2_half2_pt3 = (fixed_end_face2_half2_pt3_x,
    fixed_end_face2_half2_pt3_y,fixed_end_face2_half2_pt3_z)

faces2 = f2.findAt(((fixed_end_face2_half2_pt1), ), ((
    fixed_end_face2_half2_pt2), ), ((fixed_end_face2_half2_pt3), ))

region = regionToolset.Region(faces=faces1+faces2)
mdb.models['Fiber-Fiber␣Joint'].EncastreBC(name='BC-2', createStepName='
    Initial', region=region)


#
#       create Surface for Reaktionforces
#

a = fiberAssembly

a.Surface(side1Faces=faces1+faces2, name='Surf_rf2')




#
#       create Boundary Conditions for load application line
#


f1 = a.instances['strained␣Fiber␣Half-1'].faces


BC_pt1_x_half1 = FixedFiberWidth/2+sin(AngelFiber*pi/180)*(StrainedFiberLength
    -StrainedFiberOverhang)
```

```
BC_pt1_y_half1 = FixedFiberThickness+StrainedFiberThickness/2
BC_pt1_z_half1 = DistanceFiber-cos(AngelFiber*pi/180)*(StrainedFiberLength-
    StrainedFiberOverhang)

BC_pt2_x_half1 = FixedFiberWidth/2+sin(AngelFiber*pi/180)*(StrainedFiberLength
    -StrainedFiberOverhang)+cos(AngelFiber*pi/180)*((StrainedFiberWidth-
    StrainedFiberThickness)/2)
BC_pt2_y_half1 = FixedFiberThickness+StrainedFiberThickness/2
BC_pt2_z_half1 = DistanceFiber-cos(AngelFiber*pi/180)*(StrainedFiberLength-
    StrainedFiberOverhang)+sin(AngelFiber*pi/180)*((StrainedFiberWidth-
    StrainedFiberThickness)/2)

BC_pt3_x_half1 = FixedFiberWidth/2+sin(AngelFiber*pi/180)*(StrainedFiberLength
    -StrainedFiberOverhang)-cos(AngelFiber*pi/180)*((StrainedFiberWidth-
    StrainedFiberThickness)/2)
BC_pt3_y_half1 = FixedFiberThickness+StrainedFiberThickness/2
BC_pt3_z_half1 = DistanceFiber-cos(AngelFiber*pi/180)*(StrainedFiberLength-
    StrainedFiberOverhang)-sin(AngelFiber*pi/180)*((StrainedFiberWidth-
    StrainedFiberThickness)/2)


faces1 = f1.findAt(((BC_pt1_x_half1, BC_pt1_y_half1, BC_pt1_z_half1), ), ((
    BC_pt2_x_half1, BC_pt2_y_half1, BC_pt2_z_half1), ), ((BC_pt3_x_half1,
    BC_pt3_y_half1, BC_pt3_z_half1), ))


f2 = a.instances['strained␣Fiber␣Half-2'].faces


BC_pt1_x_half2 = FixedFiberWidth/2+sin(AngelFiber*pi/180)*(StrainedFiberLength
    -StrainedFiberOverhang)
BC_pt1_y_half2 = FixedFiberThickness+StrainedFiberThickness*3/2
BC_pt1_z_half2 = DistanceFiber-cos(AngelFiber*pi/180)*(StrainedFiberLength-
    StrainedFiberOverhang)

BC_pt2_x_half2 = FixedFiberWidth/2+sin(AngelFiber*pi/180)*(StrainedFiberLength
    -StrainedFiberOverhang)+cos(AngelFiber*pi/180)*((StrainedFiberWidth-
    StrainedFiberThickness)/2)
BC_pt2_y_half2 = FixedFiberThickness+StrainedFiberThickness*3/2
BC_pt2_z_half2 = DistanceFiber-cos(AngelFiber*pi/180)*(StrainedFiberLength-
    StrainedFiberOverhang)+sin(AngelFiber*pi/180)*((StrainedFiberWidth-
    StrainedFiberThickness)/2)

BC_pt3_x_half2 = FixedFiberWidth/2+sin(AngelFiber*pi/180)*(StrainedFiberLength
    -StrainedFiberOverhang)-cos(AngelFiber*pi/180)*((StrainedFiberWidth-
    StrainedFiberThickness)/2)
BC_pt3_y_half2 = FixedFiberThickness+StrainedFiberThickness*3/2
BC_pt3_z_half2 = DistanceFiber-cos(AngelFiber*pi/180)*(StrainedFiberLength-
    StrainedFiberOverhang)-sin(AngelFiber*pi/180)*((StrainedFiberWidth-
    StrainedFiberThickness)/2)


faces2 = f2.findAt(((BC_pt1_x_half2, BC_pt1_y_half2, BC_pt1_z_half2), ), ((
    BC_pt2_x_half2, BC_pt2_y_half2, BC_pt2_z_half2), ), ((BC_pt3_x_half2,
    BC_pt3_y_half2, BC_pt3_z_half2), ))
```

```
region_Load = regionToolset.Region(faces=faces1+faces2)
fiberModel.DisplacementBC(name='BC-Load', createStepName='FiberLoad', region=
    region_Load, u1=0.0, u2=UNSET, u3=0.0, ur1=UNSET, ur2=UNSET, ur3=UNSET,
    amplitude=UNSET, fixed=OFF, distributionType=UNIFORM, fieldName='',
    localCsys=None)

#       create Surface for Reaktionforces

a = fiberAssembly

a.Surface(side1Faces=faces1+faces2, name='Surf_rfL')


#
#       Create a Node Set for reactionforce
#
nodes1 = a.surfaces['Surf_rf1'].nodes
nodes2 = a.surfaces['Surf_rf2'].nodes
nodes3 = a.surfaces['Surf_rfL'].nodes
a.Set(nodes=nodes1, name='NodeSet_rf')
a.Set(nodes=nodes3, name='NodeSet_rfL')


#
#       Create a CSYS in the connection area
#

d1 = a.datums
a.DatumCsysByOffset(datumCoordSys=d1[16], name='Datum␣csys-3', coordSysType=
    CARTESIAN, vector=(FixedFiberWidth/2, FixedFiberThickness, DistanceFiber))




#
#       create Load
#

a = fiberAssembly


s1 = a.instances['strained␣Fiber␣Half-1'].faces

load_face_half1_pt1_x = FixedFiberWidth/2 + (Center_Distance)*sin(AngelFiber*
    pi/180)
load_face_half1_pt1_y = FixedFiberThickness
load_face_half1_pt1_z = DistanceFiber - (Center_Distance)*cos(AngelFiber*pi
    /180)
load_face_half1_pt1 = (load_face_half1_pt1_x,load_face_half1_pt1_y,
    load_face_half1_pt1_z)
```

```
side1Faces1 = s1.findAt(((load_face_half1_pt1), )  )



region = regionToolset.Region(side1Faces=side1Faces1)


fiberModel.Pressure(name='Load-1',
    createStepName='FiberLoad', region=region, distributionType=TOTAL_FORCE,
    field='', magnitude=Force, amplitude=UNSET)



#
#       create a Job
#

mdb.Job(name=FiberJob_name, model='Fiber-Fiber␣Joint', type=ANALYSIS,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, description='Fiber-
        Fiber␣Adhesive␣Joint',
    parallelizationMethodExplicit=DOMAIN, multiprocessingMode=DEFAULT,
    numDomains=1, userSubroutine='', numCpus=1,
    echoPrint=OFF, modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)


mdb.jobs[FiberJob_name].submit(consistencyChecking=OFF)

mdb.jobs[FiberJob_name].waitForCompletion()

print 'end'
```

4.2 Source code for analysing the results

```
import getopt, sys
import string, fpformat
from odbAccess import *
from visualization  import *
import odbAccess
import odb
import os
import time
#
import pythoncom, win32com.client, win32api
import win32com.client

from win32com.client import constants
from win32com.client import Dispatch



# open a EXCEL file
excel = win32com.client.Dispatch("Excel.Application")
```

```
book = excel.Workbooks.Add()
sheet = book.Worksheets(1)

sheet.Range("A1").Value = "Output␣data␣for␣the␣odb␣files"
sheet.Range("A3").Value = "Sectionforce␣fixed␣fiber␣␣␣␣Case␣4:␣variabel␣
    Fiberthickness"
sheet.Range("A5").Value = "Fiber␣Width"
sheet.Range("B5").Value = "Ny"
sheet.Range("C5").Value = "Mz"
sheet.Range("D5").Value = "Mx"
sheet.Range("E5").Value = "Qx"
sheet.Range("F5").Value = "Qz"
sheet.Range("G5").Value = "My"
sheet.Range("H5").Value = "Qres"
sheet.Range("I5").Value = "Mres"


sheet.Range("K3").Value = "Sectionforce␣strained␣fiber␣␣␣␣Case␣4:␣variabel␣
    Fiberthickness"
sheet.Range("K5").Value = "Fiber␣Width"
sheet.Range("L5").Value = "Ny"
sheet.Range("M5").Value = "Mz"
sheet.Range("N5").Value = "Mx"
sheet.Range("O5").Value = "Qx"
sheet.Range("P5").Value = "Qz"
sheet.Range("Q5").Value = "My"
sheet.Range("R5").Value = "Qres"
sheet.Range("S5").Value = "Mres"




FixedFiberWidth=0.0320
FixedFiberThickness= 0.003725
FixedFiberLength=1.0
DistanceFiber=FixedFiberLength/2


#  different odb files for different geometries
odbPath_Thickness = ['FiberJob_FTh0,00465_MODE1_V.odb', 'FiberJob_FTh0,00535
    _MODE1_V.odb', 'FiberJob_FTh0,00605_MODE1_V.odb', 'FiberJob_FTh0,00675
    _MODE1_V.odb', 'FiberJob_FTh0,00745_MODE1_V.odb', 'FiberJob_FTh0,00815
    _MODE1_V.odb', 'FiberJob_FTh0,00885_MODE1_V.odb', 'FiberJob_FTh0,00955
    _MODE1_V.odb', 'FiberJob_FTh0,01025_MODE1_V.odb']
fiberThickness = [0.00465, 0.00535, 0.00605, 0.00675, 0.00745, 0.00815,
    0.00885, 0.00955, 0.01025]




jobcount = 0      # counter for opening odb files
row = 6


############################################################
```

```
#######################################################
## different Fiberthickness
#######################################################



for odbPath in odbPath_Thickness:


# Try to open odb, otherwise throw exception
    try:
        fiberOdb = openOdb(path=odbPath, readOnly=True)
    except IOError, value:
        print 'Error:', value


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


    # ------------------------------------------------------------------------
    # Define variables for the current frame
    # ------------------------------------------------------------------------

    currentFrame = fiberOdb.steps['FiberLoad'].frames[-1]


    # ------------------------------------------------------------------------
    # Define variables for the Assembly
    # ------------------------------------------------------------------------

    assembly = fiberOdb.rootAssembly


    # ------------------------------------------------------------------------
    # Define variables for stresses and displacements
    # ------------------------------------------------------------------------

    #Create CSYS
    assembly.DatumCsysByThreePoints(name='Transform␣CSYS', coordSysType=
        CARTESIAN, origin=(0, 0, 0), point1=(1.0, 0, 0), point2=(0, 1.0, 0))

    odb_stresses = fiberOdb.steps['FiberLoad'].frames[-1].fieldOutputs['S']
    odb_displacements = fiberOdb.steps['FiberLoad'].frames[-1].fieldOutputs['U
        ']
    odb_reactionforce = fiberOdb.steps['FiberLoad'].frames[-1].fieldOutputs['
        RF']
    odb_sectionforce_x = fiberOdb.steps['FiberLoad'].frames[-1].fieldOutputs['
        NFORC1']
    odb_sectionforce_y = fiberOdb.steps['FiberLoad'].frames[-1].fieldOutputs['
        NFORC2']
    odb_sectionforce_z = fiberOdb.steps['FiberLoad'].frames[-1].fieldOutputs['
        NFORC3']
    odb_coord = fiberOdb.steps['FiberLoad'].frames[-1].fieldOutputs['COORD']
```

```
# Transformation for Stresses
CSYS = assembly.datumCsyses['Transform␣CSYS']
odb_stresses_trans = odb_stresses.getTransformedField(datumCsys=CSYS)



# Create a variable that refers to the desired node set
myNset_fixed = assembly.nodeSets['NODESET_FIXED']
myNset_strained = assembly.nodeSets['NODESET_STRAINED']
RFNset = assembly.nodeSets['NODESET_RF']


# Create a variable that refers to the displacement of
# this node set
myDisplacement  = odb_displacements.getSubset(region=myNset_fixed)
myStress        = odb_stresses_trans.getSubset(position=ELEMENT_NODAL,
    region=myNset_fixed)
myRF            = odb_reactionforce.getSubset(region=RFNset)
myCoord_fixed   = odb_coord.getSubset(region=myNset_fixed)
mySF_x_fixed    = odb_sectionforce_x.getSubset(region=myNset_fixed)
mySF_y_fixed    = odb_sectionforce_y.getSubset(region=myNset_fixed)
mySF_z_fixed    = odb_sectionforce_z.getSubset(region=myNset_fixed)
myCoord_strained= odb_coord.getSubset(region=myNset_strained)
mySF_x_strained = odb_sectionforce_x.getSubset(region=myNset_strained)
mySF_y_strained = odb_sectionforce_y.getSubset(region=myNset_strained)
mySF_z_strained = odb_sectionforce_z.getSubset(region=myNset_strained)



# write to txt file
filename = "c:/SIMULIA/Abaqus/Temp/TXT/" + odbPath[0:-4]+'_fixed-strained.
    txt'
print "Writing␣to␣file:␣%s" % filename
file = open(filename, 'w')
file.write("%␣Output␣data␣for␣the" +  odbPath)
file.write("%\n\n")

# write coords fom the fixed fiber side to file
single_line = "Fixed␣Fiber␣Side\n"
file.writelines(single_line)
single_line = "##############################\n"
file.writelines(single_line)
single_line = "coords␣for␣nodes␣in␣node␣set␣" + myNset_fixed.name + "\n"
file.writelines(single_line)
single_line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format('number', '
    x_coord' , 'y_coord', 'z_coord')+"\n"
file.writelines(single_line)
file.writelines('
    ------------------------------------------------------------'+"\n")

coord_fixed=[[], [], [], []]
coord_fixed_trans=[[], [], [], []]

for v in myCoord_fixed.values:
        coord_fixed[0].append(v.nodeLabel)
        coord_fixed[1].append(v.data[0])
        coord_fixed[2].append(v.data[1])
        coord_fixed[3].append(v.data[2])

        coord_fixed_trans[0].append(v.nodeLabel)
        coord_fixed_trans[1].append(v.data[0]-FixedFiberWidth/2)
        coord_fixed_trans[2].append(v.data[1]-fiberThickness[jobcount]/2)
        coord_fixed_trans[3].append(v.data[2]-DistanceFiber)


for i in range(len(coord_fixed[0])):

    Line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format(coord_fixed
        [0][i], coord_fixed[1][i], coord_fixed[2][i], coord_fixed[3][i],)+
        "\n"

    file.writelines(Line)

file.write("\n\n")


single_line = "transformed␣coords␣for␣nodes␣in␣node␣set␣" + myNset_fixed.
    name + "\n"
file.writelines(single_line)
single_line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format('number', '
    x_coord' , 'y_coord', 'z_coord')+"\n"
file.writelines(single_line)
file.writelines('
    ------------------------------------------------------------'+"\n")



for i in range(len(coord_fixed_trans[0])):

    Line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format(
        coord_fixed_trans[0][i], coord_fixed_trans[1][i],
        coord_fixed_trans[2][i], coord_fixed_trans[3][i],)+"\n"

    file.writelines(Line)

file.write("\n\n")




# write sectionforce from the fixed fiber side to file
single_line = "sectionforce␣for␣nodes␣in␣node␣set␣" + myNset_fixed.name +
    "\n"
file.writelines(single_line)
single_line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format('number', '
    x_direction' , 'y_direction', 'z_direction')+"\n"
file.writelines(single_line)
file.writelines('
    ------------------------------------------------------------'+"\n")
```

```
Sectionforces_fixed= [[], [], [], []]


averagedSF_x = 0
SF_x = 0
nodeLabel=0
count=0
for n in myNset_fixed.nodes[0]: # doing calculations only at selected
    nodes
        nodeNum=n.label

        for v in range(len(mySF_x_fixed.values)):
            if nodeNum == mySF_x_fixed.values[v].nodeLabel:
                SF_x = SF_x + mySF_x_fixed.values[v].data


                count=count+1
                nodeLabel=mySF_x_fixed.values[v].nodeLabel


        averagedSF_x= SF_x

        Sectionforces_fixed[0].append(nodeLabel)
        Sectionforces_fixed[1].append(averagedSF_x)


        averagedSF_x = 0
        SF_x = 0
        nodeLabel=0
        count=0

averagedSF_y = 0
SF_y = 0
nodeLabel=0
count=0
for n in myNset_fixed.nodes[0]: # doing calculations only at selected
    nodes
        nodeNum=n.label

        for v in range(len(mySF_y_fixed.values)):
            if nodeNum == mySF_y_fixed.values[v].nodeLabel:
                SF_y = SF_y + mySF_y_fixed.values[v].data


                count=count+1
                nodeLabel=mySF_y_fixed.values[v].nodeLabel


        averagedSF_y= SF_y



        Sectionforces_fixed[2].append(averagedSF_y)
        averagedSF_y = 0
        SF_y = 0
        nodeLabel=0
        count=0



    averagedSF_z = 0
    SF_z = 0
    nodeLabel=0
    count=0

    for n in myNset_fixed.nodes[0]: # doing calculations only at selected
        nodes
        nodeNum=n.label

        for v in range(len(mySF_z_fixed.values)):
            if nodeNum == mySF_z_fixed.values[v].nodeLabel:
                SF_z = SF_z + mySF_z_fixed.values[v].data


                count=count+1
                nodeLabel=mySF_z_fixed.values[v].nodeLabel

        averagedSF_z= SF_z
        Sectionforces_fixed[3].append(averagedSF_z)


        averagedSF_z = 0
        SF_z = 0
        nodeLabel=0
        count=0

    for i in range(len(Sectionforces_fixed[0])):

        Line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format(
            Sectionforces_fixed[0][i], Sectionforces_fixed[1][i],
            Sectionforces_fixed[2][i], Sectionforces_fixed[3][i],)+"\n"

        file.writelines(Line)

    file.write("\n\n")



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

    nodeLabel = 0
    SF_X = 0
    SF_Y = 0
    SF_Z = 0
    SM_Z = 0
    SM_X = 0
    SM_Y = 0
    for i in range(len(Sectionforces_fixed[0])):

        nodeLabel = Sectionforces_fixed[0][i]
        SF_X = SF_X + Sectionforces_fixed[1][i]
        SF_Y = SF_Y + Sectionforces_fixed[2][i]
        SF_Z = SF_Z + Sectionforces_fixed[3][i]

        SM_Z = SM_Z + Sectionforces_fixed[2][i]*coord_fixed_trans[1][i]
        SM_X = SM_X - Sectionforces_fixed[2][i]*coord_fixed_trans[3][i]
        SM_Y = SM_Y + (Sectionforces_fixed[1][i]*coord_fixed_trans[3][i] -
            Sectionforces_fixed[3][i]*coord_fixed_trans[1][i])



    file.writelines('SF_y␣=␣' + str(SF_Y) + "\n")
    file.writelines('SM_z␣=␣' + str(SM_Z) + "\n")
    file.writelines('SM_x␣=␣' + str(SM_X) + "\n")
    file.write("\n")
    file.writelines('SF_x␣=␣' + str(SF_X) + "\n")
    file.writelines('SF_z␣=␣' + str(SF_Z) + "\n")
    file.writelines('SM_y␣=␣' + str(SM_Y) + "\n")



    file.write("\n\n")



    row1 = row

    #print to a excell file
    sheet.Cells(row,1).Value = fiberThickness[jobcount]
    sheet.Cells(row,2).Value = SF_Y
    sheet.Cells(row,3).Value = SM_Z
    sheet.Cells(row,4).Value = SM_X
    sheet.Cells(row,5).Value = SF_X
    sheet.Cells(row,6).Value = SF_Z
    sheet.Cells(row,7).Value = SM_Y
    sheet.Cells(row,8).Value = (SF_X**2 + SF_Z**2)**(1/2)
    sheet.Cells(row,9).Value = (SM_X**2 + SM_Z**2)**(1/2)

    row = row1

    nodeLabel = 0
    SF_X = 0
    SF_Y = 0
    SF_Z = 0
    SM_Z = 0
    SM_X = 0
    SM_Y = 0

########################################################################
```

```
##########################################################################


 # write coords fom the strained fiber side to file
    single_line = "Strained␣Fiber␣Side\n"
    file.writelines(single_line)
    single_line = "##############################\n"
    file.writelines(single_line)
    single_line = "coords␣for␣nodes␣in␣node␣set␣" + myNset_strained.name + "\n
        "
    file.writelines(single_line)
    single_line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format('number', '
        x_coord' , 'y_coord', 'z_coord')+"\n"
    file.writelines(single_line)
    file.writelines('
        ------------------------------------------------------------'+"\n")

    coord_strained=[[], [], [], []]
    coord_strained_trans=[[], [], [], []]

    for v in myCoord_strained.values:
            coord_strained[0].append(v.nodeLabel)
            coord_strained[1].append(v.data[0])
            coord_strained[2].append(v.data[1])
            coord_strained[3].append(v.data[2])

            coord_strained_trans[0].append(v.nodeLabel)
            coord_strained_trans[1].append(v.data[0]-FixedFiberWidth/2)
            coord_strained_trans[2].append(v.data[1]-fiberThickness[jobcount
                ]/2)
            coord_strained_trans[3].append(v.data[2]-DistanceFiber)


    for i in range(len(coord_strained[0])):

        Line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format(coord_strained
            [0][i], coord_strained[1][i], coord_strained[2][i], coord_strained
            [3][i],)+"\n"

        file.writelines(Line)

    file.write("\n\n")

    single_line = "transfomed␣coords␣for␣nodes␣in␣node␣set␣" + myNset_strained
        .name + "\n"
    file.writelines(single_line)
    single_line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format('number', '
        x_coord' , 'y_coord', 'z_coord')+"\n"
    file.writelines(single_line)
    file.writelines('
        ------------------------------------------------------------'+"\n")



    for i in range(len(coord_strained_trans[0])):
        Line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format(
            coord_strained_trans[0][i], coord_strained_trans[1][i],
            coord_strained_trans[2][i], coord_strained_trans[3][i],)+"\n"

        file.writelines(Line)

    file.write("\n\n")






    # write sectionforce from the strained fiber side to file
    single_line = "sectionforce␣for␣nodes␣in␣node␣set␣" + myNset_strained.name
         + "\n"
    file.writelines(single_line)
    single_line = '{0:5}␣␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format('number', '
        x_direction' , 'y_direction', 'z_direction')+"\n"
    file.writelines(single_line)
    file.writelines('
        ------------------------------------------------------------'+"\n")


    Sectionforces_strained= [[], [], [], []]


    averagedSF_x = 0
    SF_x = 0
    nodeLabel=0
    count=0
    for n in myNset_strained.nodes[0]: # doing calculations only at selected
        nodes
            nodeNum=n.label

            for v in range(len(mySF_x_strained.values)):
                if nodeNum == mySF_x_strained.values[v].nodeLabel:
                    SF_x = SF_x + mySF_x_strained.values[v].data


                    count=count+1
                    nodeLabel=mySF_x_strained.values[v].nodeLabel


            averagedSF_x= SF_x

            Sectionforces_strained[0].append(nodeLabel)
            Sectionforces_strained[1].append(averagedSF_x)


            averagedSF_x = 0
            SF_x = 0
            nodeLabel=0
            count=0
```

```
averagedSF_y = 0
SF_y = 0
nodeLabel=0
count=0
for n in myNset_strained.nodes[0]: # doing calculations only at selected
    nodes
        nodeNum=n.label

        for v in range(len(mySF_y_strained.values)):
            if nodeNum == mySF_y_strained.values[v].nodeLabel:
                SF_y = SF_y + mySF_y_strained.values[v].data


                count=count+1
                nodeLabel=mySF_y_strained.values[v].nodeLabel


        averagedSF_y= SF_y



        Sectionforces_strained[2].append(averagedSF_y)


        averagedSF_y = 0
        SF_y = 0
        nodeLabel=0
        count=0



averagedSF_z = 0
SF_z = 0
nodeLabel=0
count=0
for n in myNset_strained.nodes[0]: # doing calculations only at selected
    nodes
        nodeNum=n.label

        for v in range(len(mySF_z_strained.values)):
            if nodeNum == mySF_z_strained.values[v].nodeLabel:
                SF_z = SF_z + mySF_z_strained.values[v].data


                count=count+1
                nodeLabel=mySF_z_strained.values[v].nodeLabel

        averagedSF_z= SF_z
        Sectionforces_strained[3].append(averagedSF_z)


        averagedSF_z = 0
        SF_z = 0
        nodeLabel=0
```

```
            count=0

    for i in range(len(Sectionforces_strained[0])):

        Line = '{0:5}␣␣␣␣␣␣␣␣{1:10}␣␣␣␣{2:10}␣␣␣␣{3:10}'.format(
            Sectionforces_strained[0][i], Sectionforces_strained[1][i],
            Sectionforces_strained[2][i], Sectionforces_strained[3][i],)+"\n"

        file.writelines(Line)

    file.write("\n\n")



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


    nodeLabel = 0
    SF_X = 0
    SF_Y = 0
    SF_Z = 0
    SM_Z = 0
    SM_X = 0
    SM_Y = 0


    for i in range(len(Sectionforces_strained[0])):

        nodeLabel = Sectionforces_strained[0][i]
        SF_X = SF_X + Sectionforces_strained[1][i]
        SF_Y = SF_Y + Sectionforces_strained[2][i]
        SF_Z = SF_Z + Sectionforces_strained[3][i]

        SM_Z = SM_Z + Sectionforces_strained[2][i]*coord_strained_trans[1][i]
        SM_X = SM_X - Sectionforces_strained[2][i]*coord_strained_trans[3][i]
        SM_Y = SM_Y + (Sectionforces_strained[1][i]*coord_strained_trans[3][i]
             - Sectionforces_strained[3][i]*coord_strained_trans[1][i])

    file.writelines('SF_y␣=␣' + str(SF_Y) + "\n")
    file.writelines('SM_z␣=␣' + str(SM_Z) + "\n")
    file.writelines('SM_x␣=␣' + str(SM_X) + "\n")
    file.write("\n")
    file.writelines('SF_x␣=␣' + str(SF_X) + "\n")
    file.writelines('SF_z␣=␣' + str(SF_Z) + "\n")
    file.writelines('SM_y␣=␣' + str(SM_Y) + "\n")

    #print to a excell file
    sheet.Cells(row,11).Value = fiberThickness[jobcount]
    sheet.Cells(row,12).Value = SF_Y
    sheet.Cells(row,13).Value = SM_Z
    sheet.Cells(row,14).Value = SM_X
    sheet.Cells(row,15).Value = SF_X
    sheet.Cells(row,16).Value = SF_Z
    sheet.Cells(row,17).Value = SM_Y
sheet.Cells(row,18).Value = (SF_X**2 + SF_Z**2)**(1/2)
sheet.Cells(row,19).Value = (SM_X**2 + SM_Z**2)**(1/2)

nodeLabel = 0
SF_X = 0
SF_Y = 0
SF_Z = 0
SM_Z = 0
SM_X = 0
SM_Y = 0

file.write("\n\n")

#######################
#Reactionforces       #
#######################
single_line = "reactionforces␣for␣nodeset" + RFNset.name + "\n"
file.writelines(single_line)


row = row+1
nodecount = 0
count=0

RF_x = 0
RF_y = 0
RF_z = 0




RF_x_list=[]
RF_y_list=[]
RF_z_list=[]



nodeLabel=0


for v in myRF.values:

    RF_x=v.data[0]
    RF_y=v.data[1]
    RF_z=v.data[2]

    nodeLabel=v.nodeLabel



    RF_x_list.append(RF_x)
    RF_y_list.append(RF_y)
    RF_z_list.append(RF_z)
```

```
        nodecount=nodecount+1

        RF_x = 0
        RF_y = 0
        RF_z = 0





    RFx_sum = sum(RF_x_list)
    RFy_sum = sum(RF_y_list)
    RFz_sum = sum(RF_z_list)


    file.write("\n\n")


    file.writelines('reactionforce␣x-direction␣=␣' + str(RFx_sum) + "\n")
    file.writelines('reactionforce␣y-direction␣=␣' + str(RFy_sum) + "\n")
    file.writelines('reactionforce␣z-direction␣=␣' + str(RFz_sum) + "\n")


    file.write("\n\n")


    file.close()




    jobcount = jobcount + 1

    print odbPath
    print nodecount

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


#save excel file

book.SaveAs("fiber_analysis_fixed-strained_fTh.xlsx") # or .xls depending on
    version

sheet = None
book = None
excel.Quit()
excel = None
```

Bibliography [*]

Bodig, J and A J Benjamin (1993). *Mechanics of wood and wood composites*. (Florida: Krieger Publishing Company) (cit. on pp. 1–3).

Ek, M, G Gellerstedt, and G Henriksson (2009). *Pulping Chemistry and Technology*. (Berlin: Walter de Gruyter GmbH & Co. KG) (cit. on pp. 6–9).

Puri, G (2011). *Python Scripts for Abaqus: Learn by Example*. (self-publishing company). ISBN: 978-0-615-52050-6 (cit. on p. 43).

SIMULIA-DassaultSystémes (2012). *Abaqus/CAE User's Manual* (cit. on pp. 11, 18, 19, 23).

[*]For further references, the author kindly refers to the bibliography of the appended paper in this thesis.