PARAMETER SENSITIVITY ANALYSIS OF THE VENEER PEELING PROCESS BASED ON THE FINITE ELEMENT APPROACH

Lappeenranta-Lahti University of Technology LUT

Master's Program in Mechanical Engineering, Master Thesis

2022

Esa Kauppila

Supervisor: Roope Eskola

Examiners: Professor Aki Mikkola,

Associate professor Marko Matikainen

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
LUT School of Energy Systems
Mechanical Engineering

Esa Kauppila

PARAMETER SENSITIVITY ANALYSIS OF THE VENEER PEELING PROCESS BASED ON THE FINITE ELEMENT APPROACH

Master's thesis

2022

58 pages, 32 figures, 8 tables and 5 appendices

Supervisor: Roope Eskola

Examiner(s): Professor Aki Mikkola and Associate professor Marko Matikainen

Keywords: Finite Element Method, Cohesive Zone Modelling, Parameter sensitivity

In this master's thesis, the finite element model of a veneer peeling process had been analysed for parameter sensitivity. Parameter sensitivity analysis was done by using a one-at-a-time method and using SALib python library for sensitivity analysis. The model for the veneer peeling process was made by using ABAQUS FEA software.

The model consisted of a cutting knife and wood log. The wood log had a peeling layer which used cohesive elements to enable the separation of the peel during simulation of the peeling process. Analyses of the model were done by using a Quasi-static analysis method. The interesting parameters in the model were peeling speed, peel thickness, the position of the cutting knife tip relative to the centre of the wood log, and the rake angle of the cutting knife. Sensitivity of the parameter was defined by comparing the impact on the cutting knife's cutting force maximum and average values.

The results of the one-at-a-time method showed that the most important parameter for both maximum and average cutting force was the clearance angle of the cutting knife. The second important parameter was a tie between two parameters which were the peeling speed and thickness of the peel. Peeling speed was second most important for maximum cutting force and thickness of the peel was second most important for average cutting force. The least important parameter for both maximum and average was the position of the cutting knife tip relative to the centre of the wood log. SALib could not tell anything about the parameters' sensitivity, but this was probably due to an excessively small sample size, because the total amount of run was 36. This means that nine runs for different parameters ware analysed.

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
Energiajärjestelmä/LUT
Konetekniikka

Esa Kauppila

PARAMETRIHERKYYS ANALYYSI VIILUN SORVAUS PROSESSIN ELEMENTTIMETELMÄ MALLISTA

Konetekniikan diplomityö

2022

58 sivua, 32 kuvaa, 8 taulukkoa ja 5 liitettä

Ohjaaja: Roope Eskola

Tarkastajat: Professori Aki Mikkola ja tutkijaopettaja Marko Matikainen

Avainsanat: elementtimenetelmä, koheesio mallintaminen, parametriherkkyys

Tässä diplomityössä tutkittiin viilun sorvauksen elementtimenetelmämallin parametriherkkyyttä. Parametriherkkyyttä tutkittiin sekä varioimalla yksi parametri kerrallaan sekä käyttäen herkkyysanalyysiin soveltuvaa SALib Python-kirjastoa. Elementtimenetelmämalli tehtiin ja tutkittiin käyttäen ABAQUS-elementtimenetelmäohjelmistoa.

Viilun sorvausmalli koostuu terästä ja tukista. Tukissa on mallinnettu kerros, mikä leikataan pois analyysissä ja tämä kerros on mallinnettu käyttäen koheesio-elementtejä. Koheesio-elementtejä käytetään mahdollistamaan kerroksen erkaneminen tukista analyysin aikana. Analyysit tehtiin käyttäen kvasistaattista menetelmää. Tutkittavat parametrit olivat viilun nopeus, viilun paksuus, terän päästökulma ja terän kärjen sijainti suhteessa tukin keskiöön. Parametriherkkyys arvioitiin tarkastelemalla parametrien vaikutusta terään kohdistuvan leikkausvoiman maksimi- ja keskiarvoon.

Tuloksista nähtiin, että tärkein parametri, joka vaikutti sekä leikkausvoiman maksimiin että keskiarvoon oli terän päästökulma. Toiseksi tärkeimmät parametri olivat viilun nopeus ja viilun paksuus. Viilun nopeus oli toiseksi tärkein leikkausvoiman maksimissa ja viilun paksuus puolestaan keskiarvo leikkausvoimassa. Vähiten vaikuttava parametri maksimi ja kaskiarvo leikkausvoimalle oli terän kärjen sijainti suhteessa tukin keskiöön. Herkkyysanalyysi, joka tehtiin SALib:lla ei antanut tuloksia ollenkaan. Tämä johtui todennäköisesti parametrien liian pienestä variaatiomäärästä. Yhteensä tehtiin 36 analyysiä eri parametriarvoilla, mikä tarkoitti yhdeksää eri arvoa yhdelle parametrille. SALib:n analysaattori tarvitsisi riittävän suuren otannan, että se pystyisi kertomaan parametrien yhteyksistä tuloksiin.

ACKNOWLEDGEMENTS

I would like to thank Aki Mikkola, Marko Matikainen and Roope Eskola for the help that they give me to write the thesis and the feedback they provided. Also, I would like to thank Babak Bozorgmehri for all the help he provided modelling and the running simulations for the thesis.

SYMBOLS AND ABBREVIATIONS

Roman Characters

C Damping matrix

F External force matrix

$\boldsymbol{f}$ External force vector

K Stiffness matrix

$\mathbf{K_g}$ Global Stiffness matrix

$\mathbf{K_1}$, $\mathbf{K_2}$ and $\mathbf{K_3}$ Local stiffness matrixes 1,2 and 3

M Mass matrix

r_i Inner radius of the wood log

$\boldsymbol{u}$ Displacement vector

$\dot{\boldsymbol{u}}$ Velocity

$\ddot{\boldsymbol{u}}$ Acceleration

Abbreviations

CZM Cohesive zone modelling

FEM Finite element method

LVL Laminated Veneer Lumber

OPG Optimal Peeling Geometry

XFEM Extended Finite element method

Table of contents

1 Introduction

Veneers are valuable forest products which can be used in multiple products. Veneers are made from the different species of hardwood and softwood. Both hardwood and softwood have different uses for products. Veneers can be used as cover for a more attractive outer face for example for doors and tables. Veneer covers can be used also as architectural design features. Often high-quality hardwood veneer is used for covering to achieve visually pleasing result. Veneer can be made into laminated veneer lumber (LVL). LVL is used in construction projects such as new buildings, repairs, and renovations. LVL is often made out softwood veneers which have been glued together. LVL uses thicker veneers than covering. Another use for softwood veneer is manufacture into plywood. This plywood can be used as inner structure, for example for floor panels covered with hardwood veneer. (Metsä wood, 2022; Smith and Hansen 2001)

A veneer production process consists of multiple phases. These phases are log handling, veneer peeling, veneer drying and patching etc. (Raute, 2021). In this work, the veneer peeling process is examined more closely. From the peeling process, the focus is on the veneer lathe. The main function of the veneer lathe is to cut peel out of a wood log. This is done by rotating the wood log with the desired angular velocity and using a stationary cutting knife to cut the peel of the log. The wood log can be rotated with or without spindles, depending on the machine. The number of spindles depends on the machine, but there are often two. The spindles are driven by electric motors. The lathe has space to drop the remaining log core when the minimum diameter has been achieved. A lathe also often has support rolls for the log. The main function of the support rolls is to ensure that log geometry stays as a straight cylindrical during the peeling process. A roller nose bar on top of the cutting knife guides the peel to exit the lathe to the next machine. There are also spindleless lathes where log rotation is done by support rolls. Spindleless lathes can handle smaller diameter logs than spindled lathes. Both Spindleless and spindled lathes drop the rest of the log when the minimum log diameter is achieved in peeling. There are hybrid lathes on the market that combine both solutions for log rotation. With hybrid technology the best parts of each technology can be achieved. An example of the veneer lathe is Raute's veneer lathe R5. Raute veneer lathe R5 is shown in Figure 1.

Figure 1. Raute Veneer lathe R5. (Raute R5 lathe picture, 2021)

Raute veneer lathe R5 consists of a cutting knife, double spindles and fixed nose bar. A knife gap and pitch angle of the lathe can be controlled digitally. The Lathe has automated lubrication for critical areas. Raute veneer lathe R5 can handle log diameters from 130 mm to 600 mm. The lathe can peel at speed of up to 300 m/min, which is 5 m/s as a tangential velocity of the peel exiting from the lathe. The lathe drops a leftover core when the minimal diameter of 55 mm has been reached. The R5 lathe uses Raute's optimal peeling geometry (OPG), which ensures the quality of veneer is a constant trough out the peeling process. For this, the lathe support rolls also act as knife pressure and peels vibration controllers. OPG also, adjust a knife angle when a log gets smaller during peeling. (Raute Veneer Lathe R5, 2021; Raute Veneer peeling, 2021)

An example of the hybrid veneer peeling lathe is the Raute veneer lathe R7-Hybrid. R7-Hybrid handle peeling with and without spindles. This hybrid solution enables extraction of the smallest possible core from the wood log. R7-Hybrid Raute veneer lathe can handle log diameters from 130 mm to 600 mm. The peeling speed of the lathe is the same as in R5 Raute veneer lathe. The minimum core diameter for the R7-Hybrid is 25 mm. The R7-Hybrid has two spindles and a powered roller bar to rotate wood logs a during peeling. The R7-Hybrid has OPG to ensure maximum quality for the veneer. The lathe has an automated

knife change which makes maintenance easier and work safer. (Raute Veneer Lathe R7-Hybrid, 2021; Raute Veneer Peeling Line R7-Hybrid, 2021)

The veneer lathe is only one part of the veneer production line. There are multiple parts in the line before and after the veneer lathe. Figure 2 shows an example of the veneer peeling line from Raute.

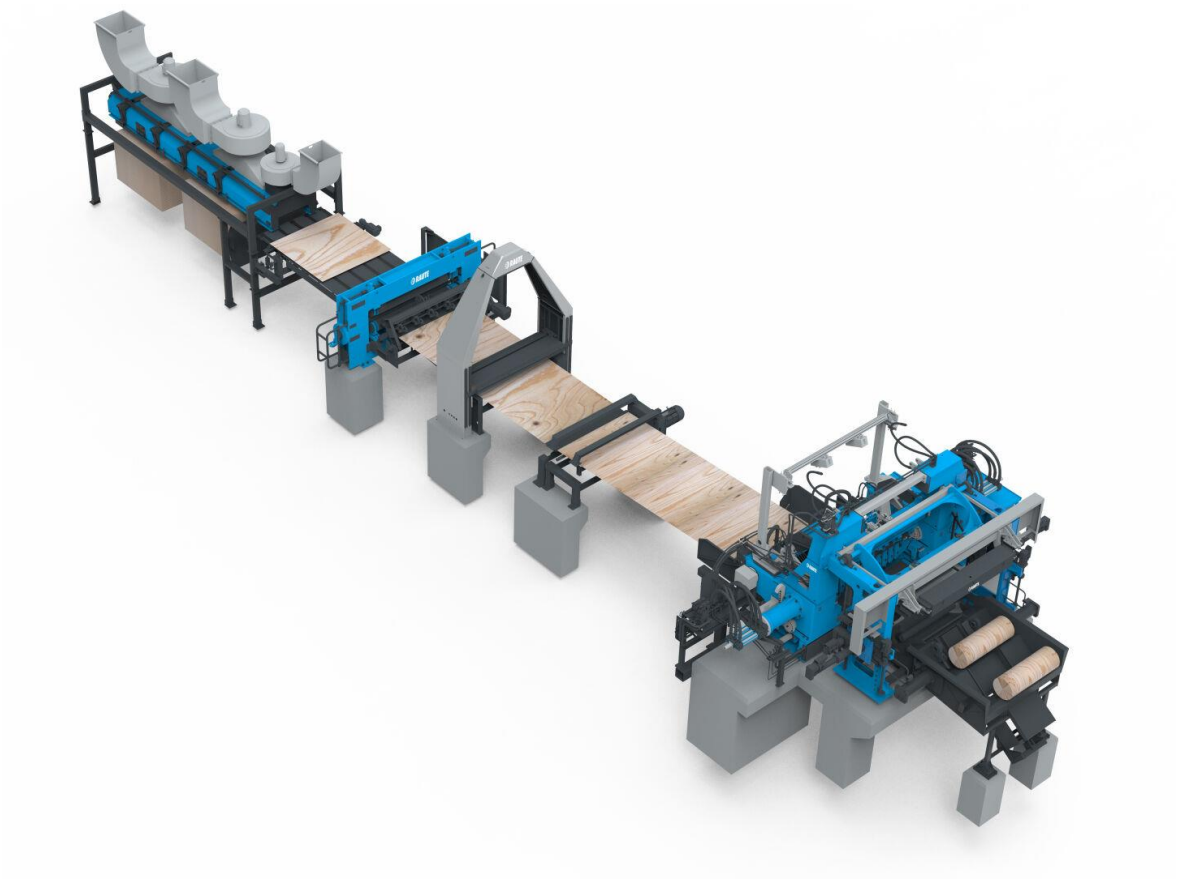


Figure 2. Raute veneer peeling line R5. (Raute Peeling line R5 picture, 2021)

Figure 2 shows that before the veneer lathe there is a log loader which loads logs to the lathe to be peeled. Figure 2 also shows that wood bark can be removed before peeling. The veneer peeling process is continuous and there are multiple other stages after peeling. For example, Figure 2 contains a moisture measuring device which measure the veneer's moisture when it passes through. There is also a visual grading device which grades veneer according to visual properties. In Figure 2 there is a veneer cutter which cuts peeled veneer to the desired sheet width. After grading and clipping, the veneer sheets are stacked with vacuum belt stacker which in the end the peeling line, see Figure 2. The stacked veneers are then moved to green veneer storage or to the next process line which is veneer drying. The drying of

veneers is necessary to achieve the moisture content needed in the next manufacturing stage. In peeling, wood logs have a higher moisture content to ensure a good quality of veneer in the process. A veneer process line can have walkways for workers to safely move around machines during production, giving easier access to certain parts of the machines during service breaks or for inspection during production stop.

1.1 Objective of the work

In this master's thesis, the finite element model of a veneer peeling process is studied by using parameter analysis. The focus of this work is to conduct the parameter analysis of the model to see how different parameters affect the model. Parameter analyses are done for Quasi-static and dynamic analyses. This master's thesis work is done for Raute.

The FE model of the veneer peeling process has already been created during previous co-operation between LUT University and Raute. The model is made by using ABAQUS 2020 software. For his thesis, the model is recreated in ABAQUS 2020 by using Python scripting. Scripting is used to ensure that any parameter change in the model takes place as desired.

The reasons for performing parameter analysis for the veneer peeling model are to check how sensitive the model is to parameter change; how different parameters affect the model's performance, and how parameters affect the simulation times and convergence of the model.

2 Methods

Next, the methods used to conduct the parameter sensitivity analysis of the veneer peeling process are explained. These methods are explained at the general level without deep details of each method. The methods are the finite element method, cohesive zone modelling, contact modelling, and sensitivity analysis. For each method, examples are presented at a general level.

2.1 Finite element method

The finite element method (FEM) is a numerical method in order to calculate stresses and strains in a structure. This is done by breaking the structure into smaller parts which are connected to each other. These parts are called elements, and every element has its own degrees of freedoms and stiffness matrices. FEM also need boundary conditions to ensure the structure does not float away during analysis. Usually, if no boundary conditions are defined, the result is an error in the solution, or the analysis does not even start. There are two types of displacement types in the FEM: linear and non-linear. Linear FEM will usually be used when displacements and strains are small. Non-linear FEM will usually be used when displacements are large, or the strain is large or due to geometrical shape, for example a long structure and thin cross section. Non-linear FEM is needed when the rotation or displacement is large, since linear FEM cannot correct a deformed shape function compared to non-linear. An example case or non-linear FEM is Cook’s problem.

FEM is often used to solve 2D and 3D problems. There are different element shapes for both 2D and 3D. In 2D, there are two shapes: triangles and quadrilaterals. In 3D, there are three shapes: tetrahedrons, hexahedrons and pentahedrons. For 1D cases there is only one type of shape, a beam. To connect different elements, node points are used. For 1D, 2D and 3D there is a different number of nodes in the element that can be selected. Figure 3 shows different finite element shapes and different node variants.

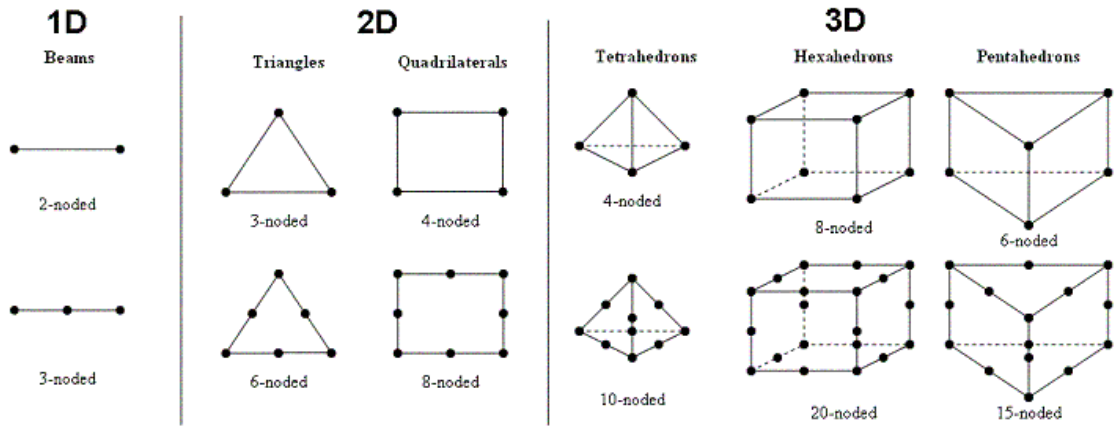


Figure 3. Different finite element shapes used in the FEM with different number of nodes (FEM Shapes picture)

More node points can be used in the finite elements when more accurate shapes are desired. This is because extra middle nodes help the element to deform more precisely. A drawback of using more nodes is increased computational effort to solve a system because adding more nodes adds more degrees of freedoms to the system. In the case of 2D, there are two translations and one rotational degree of freedoms, which is a total of three degrees of freedom. In the case of 3D, there is a total of six degrees of freedom, three translations and three rotational. The case of 3D degrees of freedoms can quickly start rising if more nodes are used in the finite elements. For example, four node tetrahedra have a total of $4 \cdot 6 = 24$ degrees of freedom and 10 node tetrahedra have a total of $10 \cdot 6 = 60$ degrees of freedom. The difference between elements is 36 degrees of freedom. When adding thousands of finite elements to the model the difference increases vastly and the computational effort rises.

For FEM a material model which defines how finite elements react to external loads is also necessary. There are different material models developed for the finite element method. The simplest is a linear material model which follows Hook's law. There are different material models for different materials like rubber, composite, and metals. Material property is connected to the stiffness matrix of the finite elements. The stiffness matrix defines what size deformation is under applied loads.

For a finite element method there are different kinds of elements which have been developed for different uses such as a beam, shell, solid continue and plate element. Different types of elements are developed for different uses. For example, beam elements are used when beams are analysed. There is also a difference between element type which forces element type can

handle. For example, a rod element can only take push or pull force and it does not take moment. On the other hand, beam element can take forces in both *x* and *y* -axis direction and rotation displacements. In the case of a 3D problem, a solid continuous element can take all the different deformations and generate a result from them.

The solution in a finite element method is based on the interaction between force, stiffness, and deformation matrices. A relationship between matrices is shown in the following equation:

$$\mathbf{F} = \mathbf{K_g}\boldsymbol{u} \tag{1}$$

where **F** is an applied external force matrix, $\mathbf{K_g}$ is a global stiffness matrix which is made of stiffness matrices, and $\boldsymbol{u}$ is a vector of displacements which are caused by external forces. Often in finite element method, deformations are of interest. To perform this equation, (1) is multiplied by reverse global stiffness matrix, and the equation will be following

$$\boldsymbol{u} = \mathbf{K_g^{-1}F} \tag{2}$$

Where $\mathbf{K_g^{-1}}$ is a reverse global stiffness matrix and $\boldsymbol{u}$ and **F** are the same as in equation (1).

Before either equation (1) or (2) can be used, the global stiffness matrix needs to be formed. The global stiffness matrix is formed from local stiffness matrices. Local stiffness matrices are created according to the finite elements in the case. After local stiffness matrices are created, they are combined to correct places according to the global coordinates. Local stiffness matrices are combined according to the finite element's connections. This means that if there are elements before and after a selected element, these elements' local stiffness matrices are connected to other matrices at the same degree of freedoms. After the global stiffness matrix is done, the models boundary conditions must be considered. Boundary conditions are applied to the global stiffness matrix by deleting corresponding rows and lines from the global stiffness matrix. This is done because boundary conditions limit how a system can deform and by deleting corresponding rows and lines these variables which are affected by boundary conditions are not calculated to the deformation vector.

Figure 4 shows an example cantilever case which is used to demonstrate how the stiffness matrix is done. The cantilever beam has been split into three elements. Every element has its own local stiffness matrix. The local stiffness matrices are called $\mathbf{K_1}$, $\mathbf{K_2}$ and $\mathbf{K_3}$. There are four nodes at the cantilever beam. The node locations are at the support, the border between

element one and two, border between element two and three, and at the free end. To simplify example nodes, they have two degrees of freedom each which are in the *x* and *y* direction. The total amount of degrees of freedom in system is $4 \cdot 2 = 8$.

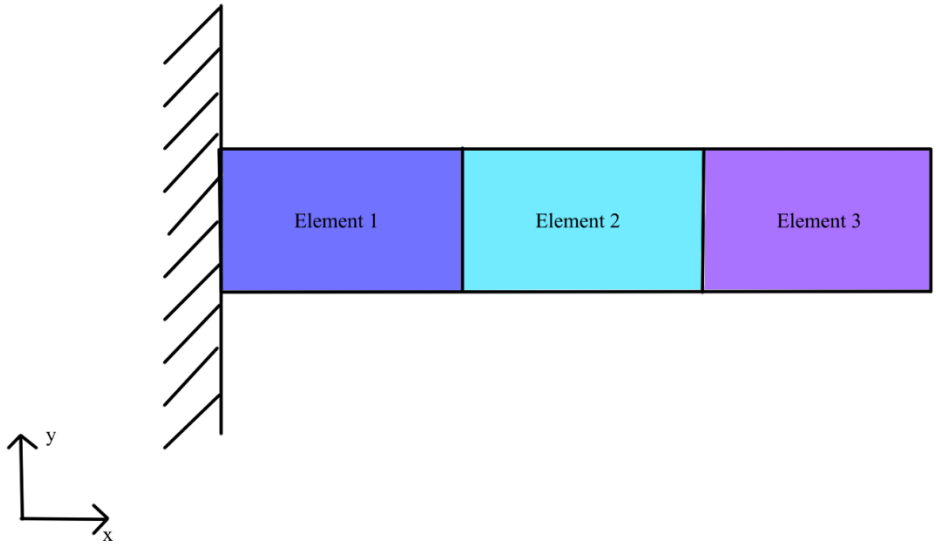


Figure 4. Example of the finite elements for cantilever case without external force

Local stiffness matrices $\mathbf{K_1}$, $\mathbf{K_2}$ and $\mathbf{K_3}$ are following

$$\mathbf{K_1} = \frac{EI}{L^3}\begin{bmatrix} 12 & -6L & -12 & -6L \\ -6L & 4L^2 & -6L & -2L^2 \\ -12 & -6L & 12 & -6L \\ -6L & -2L^2 & -6L & 4L^2 \end{bmatrix}$$

$$\mathbf{K_2} = \frac{EI}{L^3}\begin{bmatrix} 12 & -6L & -12 & -6L \\ -6L & 4L^2 & -6L & -2L^2 \\ -12 & -6L & 12 & -6L \\ -6L & -2L^2 & -6L & 4L^2 \end{bmatrix}$$

$$\mathbf{K_3} = \frac{EI}{L^3}\begin{bmatrix} 12 & -6L & -12 & -6L \\ -6L & 4L^2 & -6L & -2L^2 \\ -12 & -6L & 12 & -6L \\ -6L & -2L^2 & -6L & 4L^2 \end{bmatrix}$$

Where *E* is Young's modulus of the material, *L* is the length of the element, and *I* is the moment of inertia of the cross section. The global stiffness matrix is created by combining local stiffness matrices. Combining is done at the border of the elements by adding stiffnesses together at connection point. The global stiffness matrix K_g follow

$$\mathbf{K_g} =$$

$$\frac{EI}{L^3}\begin{bmatrix} 12 & -6L & -12 & -6L & 0 & 0 & 0 & 0 \\ -6L & 4L^2 & -6L & -2L^2 & 0 & 0 & 0 & 0 \\ -12 & -6L & 12+12 & -6L+(-6L) & -12 & -6L & 0 & 0 \\ -6L & -2L^2 & -6L+(-6L) & 4L^2+4L^2 & -6L & -2L^2 & 0 & 0 \\ 0 & 0 & -12 & -6L & 12+12 & -6L+(-6L) & -12 & -6L \\ 0 & 0 & -6L & -2L^2 & -6L+(-6L) & 4L^2+4L^2 & -6L & -k^{(3)} \\ 0 & 0 & 0 & 0 & -12 & -6L & 12 & -6L \\ 0 & 0 & 0 & 0 & -6L & -2L^2 & -6L & 4L^2 \end{bmatrix}$$

where *E*, *I* and *L* are the same as for local stiffness matrices. From the global stiffness matrix it can be seen that size is connected with the number of degrees of freedom in the model. In the example there are eight degrees of freedoms and global stiffness matrix has the size of 8 × 8. This also confirms that adding more degrees of freedoms to the model increases the computational effort for the model, and the size of a global stiffness matrix increases quickly for a traditional finite element method.

There are different analysis types which can be used in the finite element method. One of these is the quasi-static analysis which is used in the thesis. Quasi-static analysis is used to simulate dynamic problems when dynamic events happen slowly. The quasi-static analysis solution is based on the dynamic equilibrium equation:

$$\mathbf{M}\ddot{\boldsymbol{u}}(\mathrm{t}) + \mathbf{C}\dot{\boldsymbol{u}}(\mathrm{t}) + \mathbf{K}\boldsymbol{u}(\mathrm{t}) = \boldsymbol{f}(\mathrm{t}) \qquad (3)$$

where **M** is a mass matrix, **C** is a dampening matrix, **K** is a stiffness matrix, $\ddot{\boldsymbol{u}}$ is an acceleration vector, $\dot{\boldsymbol{u}}$ is a velocity vector, $\boldsymbol{u}$ is a displacement vector and $\boldsymbol{f}$ is an external force vector. From equation (3) it may be noticed that displacement vectors and force vectors are time-dependent. Also, equation (3) has inertia forces which are due to acceleration $\mathbf{M}\ddot{\boldsymbol{u}}(\mathrm{t})$ and damping $\mathbf{C}\dot{\boldsymbol{u}}(\mathrm{t})$. In Quasi-static analysis inertial and dampening force should only have marginal importance compared with static forces. Static force is the stiffness term in the equation (3). In case when inertial and dampening force are marginal equation (3) takes the form of the static analysis equilibrium. A new form of the equation (3) does not have the same connection between the stiffness matrix and displacement what static analysis equilibrium should have if it has been built from the start. The difference between the two is that in the static analysis stiffness matrix **K** is non-linear and it is dependent on the

displacement vector, and in the dynamic analysis stiffness matrix **K** is linear and is independent of displacement vector. (Natário, P., Silvestre, N. and Camotim, D., 2014)

Besides the traditional finite element method, there are further developed finite element methods that extend the methods' usability to wider a range of problems. One of the developed methods is extended finite element method (XFEM). An extended finite element method has been created to overcome a shortcoming in contact, fracture, and damage in the traditional finite element method. The traditional finite element method cannot handle discontinuities and jumps in displacements. The extended finite element method handles jumps in displacement by adding the extra degrees of freedom to the nodes around a discontinuity point. The added degrees of freedom accomplish that extended finite element method can consider jumps in displacements. (Harish, 2020)

2.2 Cohesive zone modelling

Cohesive zone modelling (CZM) is usually used when studying fracture mechanics in the finite element model. An example of fracture mechanics is use cohesive zone modelling for predicting a cracking process in the model. Cohesive zone modelling elements are placed between continuum elements. Cohesive zone modelling requires additional description which allows separation in the element. This additional description is called traction separation law. There are different traction separation laws that can be used, for example bilinear, parabolic and exponential. Traction separation laws can be either an energy or separation base. When traction separation law is a separation base it shows relation between surface traction and displacement jumps in the cohesive zone. When traction separation law is an energy base, certain energy threshold needs to be met before the separation can happen. This threshold is defined by area under the traction separation law curve. Figure 5 shows four different types of separation laws. (Kazák, 2008; Volokh, 2004)

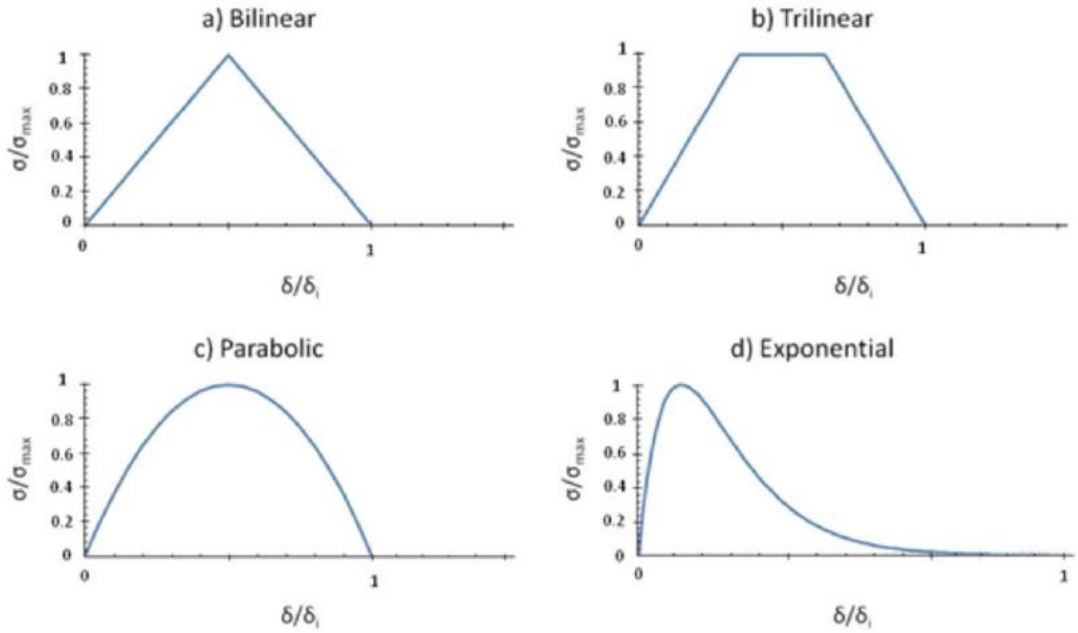


Figure 5. Four different types of separation laws. (Dogan et al, 2012)

Figure 5, part a) shows how bilinear traction separation law works. Bilinear consists of two lines. Up to the peak value is damage initialization, and decreasing slope shows damage evolution. In figure 5 the *y* axis has stress and the *x* - axis has strain. According to bilinear traction separation law, both damage initialization and damage evolution is full linear. Also, the area under the separation law gives separation energy for each separation law. When the strain is one then damage propagation has reached its end and damage is outside of the separation law. That means that damage is permanent, and it starts growing more: for example, a crack is formed and the crack start growing during changing loads in the material. Different traction separation laws show different models for damage evolution. When comparing different separation laws, a clear difference can be seen between them. Figure 5, when comparing a) bilinear and b) trilinear, clearly shows that a damage initialization phase in these curves is linear. Also, damage initialization happens over longer a period in trilinear than in bilinear. When comparing with c) parabolic and d) exponential from Figure 5 and checking a damage initialization phase for both parabolic and exponential, it may be seen that both have a non-linear curve, but there is difference in how fast an initiation phase happens due to a derivative value in the curve. Parabolic and exponential separation laws have a critical point when the derivative has a value of zero around the peak of the curves. When checking the location of the peak, it can be seen that exponential has it peak much earlier than three others. The others have a peak around the same place. When comparing damage propagation, bilinear and trilinear have constant value for a slope, but bilinear has a

steeper one. Parabolic, on the other hand, changes, increasing when approaching a damage end point for separation law. Exponential has the longest damage propagation of all four of these separation laws. Exponential damage propagation derivative changes its direction multiple times before reaching the damage end point. Exponential separation law is the only one of these which approaches the damage end point very slowly. It can be seen that at the beginning of the damage evolution it happens faster than at the end of the damage propagation. Separation energy seems to be the smallest for bilinear and biggest for either trilinear or parabolic. This is done merely by eye which means these are not the absolute results of the area under each separation law curve.

Next, we will look of a cohesive zone model with an example of an adhesive bond joint. This example is given in Figure 6.

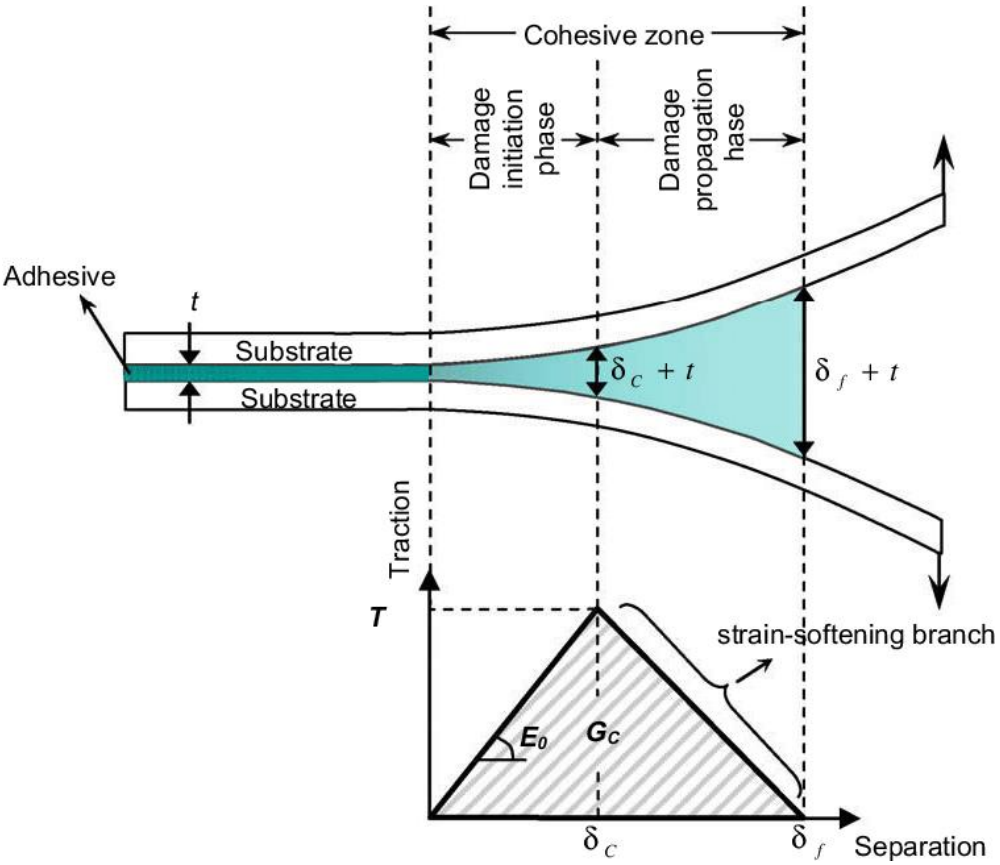


Figure 6. Adhesive bond joint example to demonstrate cohesive zone modelling. (Khoramishad, Crocombe, Kali and Ashcroft, 2010)

From Figure 6 it can be seen that light blue area is a cohesive zone and uses cohesive elements. To cohesive zone modelling to work it needs a path to that damage follows and it needs to be placed between two continues elements parts. The traction separation law of the

Cohesive zone can be thought of like coil springs which are added between two surfaces. A damage initiation phase can be thought of like the stiffness of the coils spring when load is applied at the end on the joint. When load is applied, coil springs will deform according to the stiffness of the coil springs. When the traction separation law reaches critical point G_c in Figure 6, the spring is at its maximum elastic tension. Moving to the damage propagation phase, this can be thought of with the coil spring being deforming plastically. This means that coils in the spring are going to straighten and the spring is becoming even longer. When traction separation law hits its maximum strain value for separation the coil spring can be imagined as snapping into two parts. In other words, the coil spring breaks in half at some point of the spring. An amount of displacement in the cohesive zone is defined by the separation law and initial thickness of the cohesive zone. This displacement is shown in Figure 6 with two points: a critical and final point. Separations for these points follow in the Figure 6:

critical point: $\delta_c + t$,

final point: $\delta_f + t$,

where δ_c is critical separation displacement, δ_f is final separation displacement and t is an initial thickness of the cohesive zone. Selecting separation law for a model depends on the case and type of failure is being studied. This is because different separation laws have different shapes as seen from Figure 5 and different separation laws will want to be used for different cases which suits its best to give the realistic results of the damage.

2.3 Contact modelling

Two different contacts are used in the model in Abaqus. These contacts are tangential and normal contacts. As tangential contact friction behaviour is used, and as normal contact hard contact behaviour is used.

Abaqus defines friction as stresses between the interface of the bodies. In the Abaqus there are different friction models available for example the classical isotropic Coulomb friction method and anisotropic extension of the basic Coulomb friction model. The basic Coulomb friction model defines sticking and sliding by using shear stress that two surfaces can transfer to each other up to a certain limit. The phase before surfaces start moving relatively to each

other is called sticking. The limit point defines when sticking changes to sliding. This limit point is called critical shear stress and it is defined as a linear function of the contact pressure and friction coefficient. A sticking region is shown in Figure 7. (ABAQUS Friction, 2017)

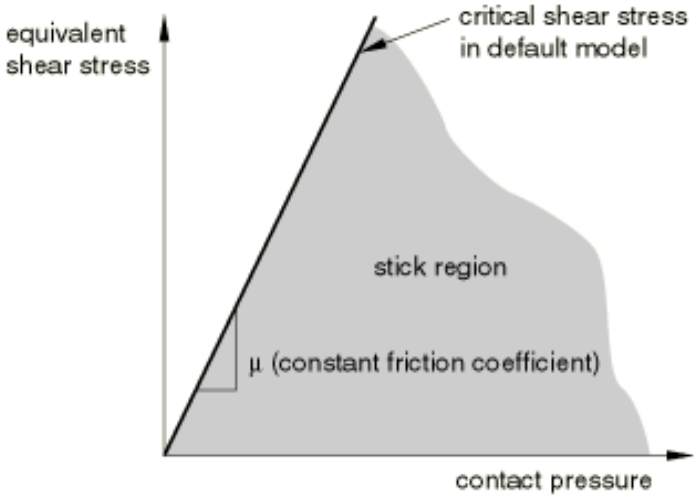


Figure 7. Slip region for normal Coulomb friction method (ABAQUS Friction, 2017)

From Figure 7 it may be seen that critical shear stress has a slope of a constant friction coefficient. The area under the critical shear stress is a sticking region where tow contacts pairs will stay together. When contact pressure and equivalent shear stress have value which is at critical shear stress line contact will still stick but it is at the edge to start slipping. When contact pressure and equivalent shear stress have a combination which is outside stick region area then contacts are slipping to relative to each other. A maximum limit can also be defined for critical shear stress. In Figure 8 shows sticking and slip regions when critical shear stress is limited. (ABAQUS Friction, 2017)

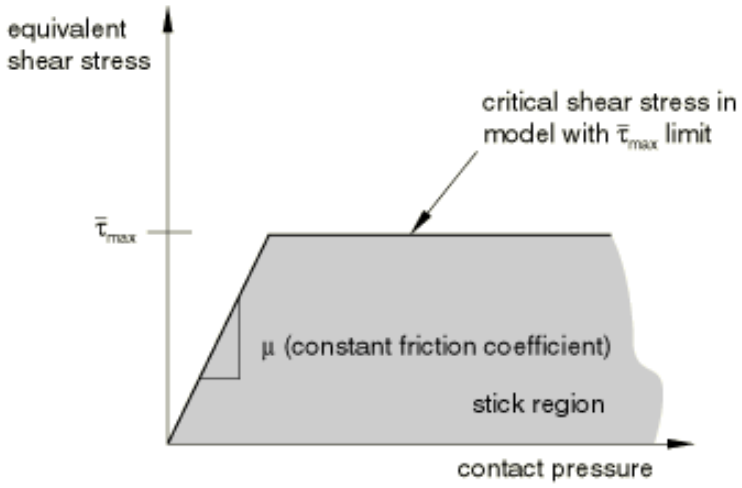


Figure 8. Slip regions with limited critical shear stress (ABAQUS Friction, 2017)

From Figure 8 may it be seen when applied a limit for the critical shear stress plot changed in the way that is has like a roof. The area under the curve is still the sticking region and the area outside the curve is the slipping region. The reason limited critical shear stress is used is contact pressure stress may get very large. One example is when modelling a manufacturing process which will cause very large contact stresses. If there is no limit in present, Coulomb theory can make critical shear stress greater than the yield stress of the material under the contact surface. (ABAQUS Friction, 2017)

2.4 Sensitivity analysis

Sensitivity analysis is used for determining how uncertainty in the model inputs affects mathematical model's or output of a system. Sensitivity of inputs is represented with a numerical value called a sensitivity index. There are different indexes for sensitivities these are first-order, second-order, and total-order index. First-order indexes measure the contribution of the output variance according to the single model input. Second-order indexes measure the contribution of the output variance according to interaction between two model inputs. Total-order index measures the contribution of the output variance which is a result of the model input. Total-order index includes both first and second-order effects and all higher-order interactions. (SALib Basic, 2021)

In this thesis SALib is used for sensitivity analysis. SALib is Python library which does not intervene with a model of interest. SALib create model inputs by using one of the sample functions and compute sensitivity analysis according to the inputs and outputs. Outputs are provided by the model of the interest. A typical sensitivity analysis by using SALib has four steps: 1. Define inputs and range for the model. 2. Make inputs by using selected sample function. 3. Run the model of interest with created inputs and save output data from the model. 4. Uses analysis function to compute sensitivity indexes from outputs. SALib has many analysis methods, such as Sobol, Morris and FAST. Also, there are multiple sample functions, for example Latin hypercube sampling, the method of Morris and the Sobol sequence. When using SALib user can only use one sampler and analysis function. (SALib Basic, 2021; SALib API, 2021)

Let us look at an example for sensitivity analysis by using simple mathematical equation as an example. Selected mathematical equation is second order polynomial function which is following

$$f(\mathrm{x}) = a + b\mathrm{x}^2 \qquad (4)$$

Where *a* and *b* are parameters for which sensitivity analysis is performed. Variable x will not take part in the sensitivity analysis. Sensitivity analysis is done by using Saltelli sampler and Sobol analysis. Figure 9 shows plotted results from the sensitivity analysis of equation (4). (SALib Basic, 2021)

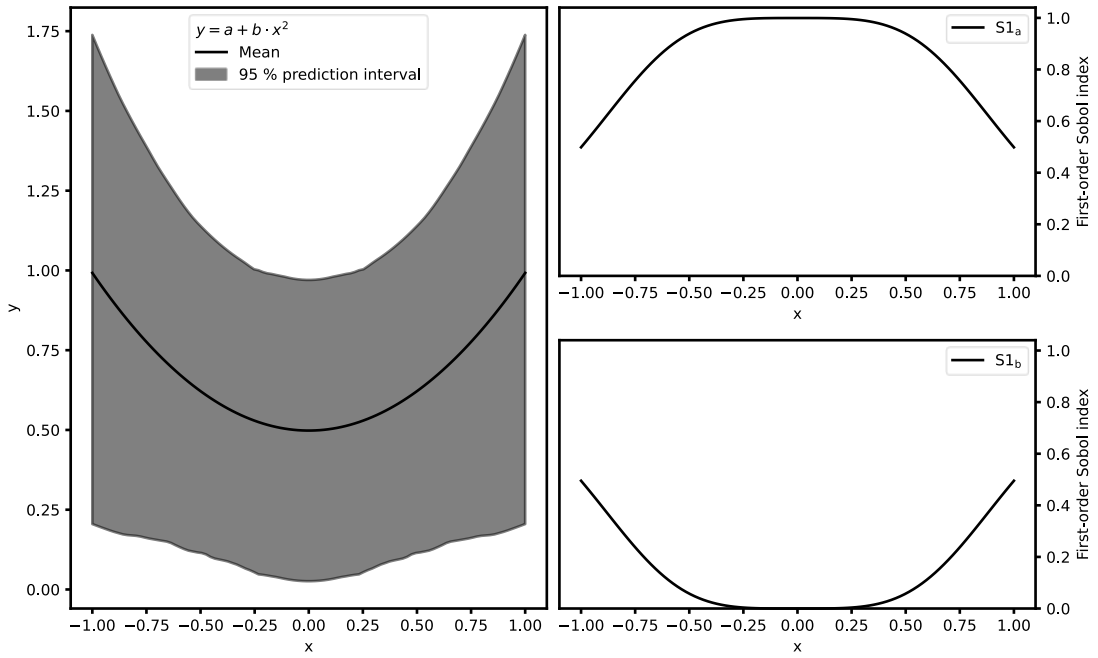


Figure 9. Results of the example second order polynomial function sensitivity analysis Fig source: (SALib example result picture, 2021)

In Figure 9 the results of the sensitivity analysis for equation (4) may be seen. The left side of Figure 9 shows the prediction of the curve for different *a* and *b* values when x is between -1 and 1.The grey area is 95 % prediction interval meaning that analyser is 95 % sure that plot of the equation (4) is in this area. The black line shows the mean value for predictions. The prediction looks parabolic, which equation (4) represents. On the right side of Figure 9 are the figures for first-order Sobol indexes for *a* and *b* plotted with x values from -1 to 1. When, checking First-order Sobol index for variable *a* observation can be done that *a* has significant impact on the results when x is between -0.5 and 0.5. In this region, *a* has first-order sobol index between 1 and 0.8. When, the first-order Sobol index is closer to 1 more

impact variable has in the solution and when closer to the zero less impact variable has or does not impact at all if zero. Variable *a* has less impact when x is between ±0.5 and ±1 but it has some impact on the solution since the first-order Sobol index is between 0.8 and 0.5. For variable *b* it has no impact at all when x is between -0.25 and 0.25 because the first-order Sobol index is zero. Variable *b* starts to have an impact from ±0.75 to ±1 because the first-order Sobol index has a value between 0.2 and 0.4. Variable *b* has less impact than the variable *a* for solution of the equation (4). These results are excepted because when knowing how equation (4) works. Variable *a* is constant in the equation and for second order polynomial constant defines a starting point for the bottom or the top of parabolic. Variable *b* is multiplier for the slope which means *b* only defines how fast or slowly the parabolic will rise or fall.

3 Modelling

The model of the veneer peeling lathe is based on real-life operations. The veneer peeling lathe operates with a log loader. The Lathe secure spindles to the log and starts rotating it. When the desired rotational speed is reached, peeling is begun by pushing a cutting knife against the log. There are also support rolls for supporting the log straightness during peeling. Figure 10 shows the peeling process.

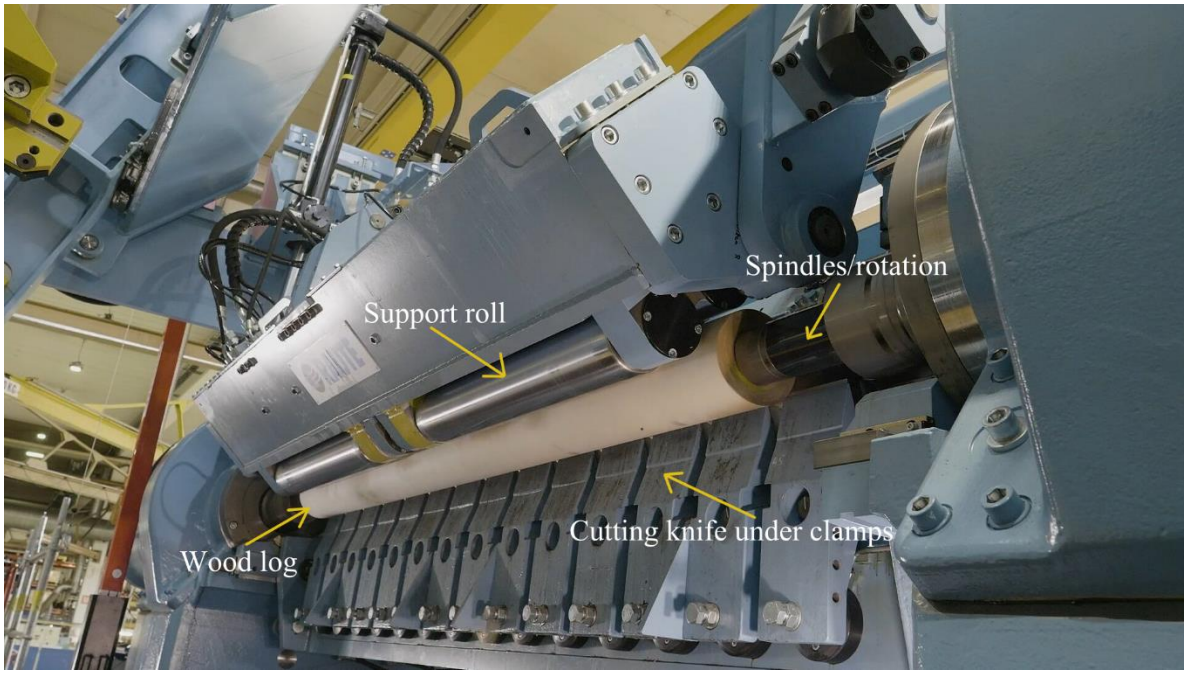
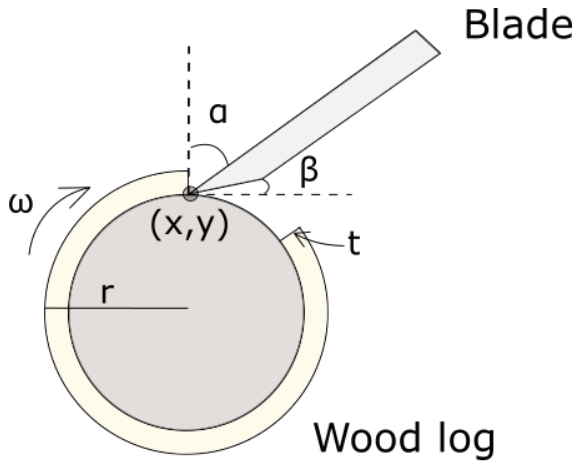


Figure 10. Veneer peeling in action (Raute Lathe R7, 2021)

In Figure 10 different parts of the veneer peeling lathe may be seen. The main parts are shown with an arrow and with text. The main parts are a cutting knife, which is in this case under clamps so cannot be seen in Figure 10. Support rolls can be seen on the top of the wood log, and there are multiple support rolls. Last but not least, Figure 10 shows spindles that rotates the wood log when it is being peeled.

The model of the veneer peeling lathe consists of a wood log and cutting knife. For the model, sensitivity analysis is conducted for different parameters. Figure 11 shows a principle of the model geometry with parameters.



ω = angular velocity of wood log

α = Rake angle

β = Clearance angle

t = Cut deep

r = Wood log radius

(x,y) = Blade's tips position

Figure 11. Principal picture of the model with parameters

There are some limitations in the model compared with a real life process. In real life, veneer comes out continuously from the wood log when peeling is happening, and the log radius gets smaller by every rotation. In the model, continuous peeling can only be achieved for an assigned simulation time. For the model, reduction of the wood log radius by every full rotation cannot be simulated because of the changes for a wood material model which a radius change would cause. Another difference is that in real life the cutting knife starts peeling when it is pushed against the wood log. In the model, this is not possible because cohesive elements which are used need to be placed alongside an expected crack path.

The model is performed with ABAQUS 2020 FEA software. The model is made by using centimetre-gram-second (CGS) unit system.

3.1 Wood log

A wood log is modelled as a cylinder shape which has a peeling layer on top. The peeling layer only covers about ¾ of the log's perimeter. Wood log geometry is sketched by using lines to create the thickness of the peel and arcs to connect lines. The line's starting point is the desired log radius and an end point is at the desired peel thickness. For example, the log radius of 25 cm and peel thickness of 0.3 cm correspond to the starting point of (0, 25) and end point (0, 24.7). Both these points are made on the top and on the right. For the right the points are, for example, (25, 0) and (24.7, 0). After these, the lines are made and outer points are connected by using an arc tool. The Same is done for inner points.

This sketch is then extruded with a length of 140 cm. After extrusion, the rest of the log under the peeling layer is separated from the part with a partition tool. Partition is done by sketching a circle around one end of the log; then, along the length of the log, geometry is partitioned. Then, another partition is made for the cohesive zone in the geometry. This partition is made by sketching on the face of the peeling layer. On the face, a rectangular sketch is made with a height of 0.0001 cm, and the length is the same as the wood log. After sketching, partition is done by selecting a sketch and a direction alongside the inner circle. Partition is done by using s revolving partition tool. A reference point is added to the log's centre which will be used when defining the boundary condition for the log. The coordinate for the reference point is (0, 0, 0). The log is partitioned twice more in the x and y planes to created four sections in the inner part of the wood log.

Four surfaces are defined for the wood log. These surfaces are a wood log inner core top surface, cohesive layer bottom surface, a peel layer bottom surface and a peeling layer front surface. The wood log inner core top surface is selected by using the replace selected tool to only show the inner core of the wood log. After this, the inner core top surface is selected as a surface. The Cohesive layer bottom surface is selected by first selecting a cohesive layer to display only. After this is done, the cohesive layer bottom surface is selected as a surface. The peel layer bottom surface is selected by first selecting the peel layer to display only. Then, a bottom surface of the peel layer is selected as a surface. While only the peel layer is showing, the peeling layer front surface is selected. This surface is the face of the peel layer in the z axis direction.

The wood of the log is an orthotropic material. Orientation is needed to define for the wood log part. An orientation stacking direction is selected the wood log's longitudinal axis and orientation uses the global coordinate system.

3.2 Cutting knife

Cutting knife geometry is done by the knife being placed on the top of the log and next to the peel, and this is accomplished by using lines in the sketch of the cutting knife. In table 1, coordinates for starting and ending points are shown for x and y coordinates. In table 1, r_i is for an inner radius of the wood log. From the example given, in the wood log part section r would be 24.7 cm.

Table 1. x and y coordinates for lines in cutting knife sketch; r_i is inner radius of the wood log

Name	Start point (x,y)	End point (x,y)
1st line	(0, r_i)	(5.85, r_i)
2nd line	(5.85, r_i)	(13.84, r_i + 2.91)
3rd line	(13.84, r_i + 2.91)	(13.16, r_i + 4.79)
4th line	(13.16, r_i + 4.79)	(0, r_i)

After the sketching is done, it is extruded with the length of 160 cm. The cutting knife has two surfaces which are a blade lower surface and blade top surface. The blade lower surface is selected as lower flat part of the cutting knife. The blade top surface is selected as a top surface of the cutting knife which is at incline angle.

3.3 Materials and material assignments

The model has three materials: wood, steel, and cohesive. Next, the material parameters for each material are discussed. The wood is modelled after Spruce Sitka softwood. The wood is modelled using Saint Venant-Kirchhoff material model. The parameters for Spruce Sitka that have been used can be found in (Green et al. 1999). Parameters for a fracture of the wood are adopted from (Reiterer, Sinn, and Stanzl-Tschegg, 2002). Fracture parameters are used in the cohesive material in the cohesive zone to define fracture characteristics.

Wood has three material behaviours damping, density and elastic. Damping has a value of 0.005 in the structural part. Wood has a density of 0.55 g/cm^3, which is used to define density. Since wood is an orthotropic material, the elastic material properties elastic type need to be changed to the orthotropic. For elastic data of the material properties are given in table 2.

Table 2. Data for wood's elastic behaviour (Green et al, 1999)

Name	Value
D1111	150960000000
D1122	53238000000
D2222	72900000000
D1133	41087000000
D2233	50702000000
D3333	43145000000
D1212	36026000000
D1313	31150000000
D2323	3294300000

Steel has two material behaviours density and elastic. Steel has a density of 7.8 g/cm^3, which is used to define density. Steel is an isotropic material and the type can be left as is because ABAQUS default option is isotropic for the elastic type. Steel has a Young's modulus of 2.1E+12 and Poisson's ratio of 0.29. Both of these are placed in a data section for elastic behaviour.

The cohesive has many material behaviours, which are maxs damage, damping, density and elastic. In maxs damage all normal stress data on data section uses a value of 9.559E+7 ba. Maxs damage has two sub-options, which are damage evolution and damage stabilization cohesive. Damage evolution is set at energy and traction separation law is selected as linear. Fracture energy uses the value of 840000 (Reiterer et al, 2002) in the damage evolution. The damage stabilization has only one setting to set. This is a viscosity coefficient which is uses the value of 0.002. Damping behaviour uses the same values as wood. Density of the cohesive layer is 0.5 g/cm^3. Elastic behaviour type needs to be changed to traction since cohesive zone uses traction separation law. Values for traction data are given in table 3.

Table 3. Values for elastic behaviour data for cohesive material (Reiterer et al, 2002)

Name	Value
E/Enn	13770000000
G1/Ess	19940000000
G2/Ett	9940000000

After materials are defined, sections can be created. Sections consist of material property, type of material and material specific options which a type of material defines. Three sections are made which are Wood_section, Steel_section and Cohesive_section. Wood_section is solid, homogenous and the material is wood. Steel_section is solid, homogeneous and the material is steel. Cohesive_section is cohesive, the material is cohesive, the response is traction separation. After sections, next step is assigning sections to the parts cells. The whole cutting knife geometry is assigned as Steel_section. For the wood log core of the log and peeling layer is assigned as Wood_section. The wood log's cohesive layer is assigned as Cohesive_section.

3.4 Solver settings

For the model, a new step is created, called peeling. The peeling step will be placed after the initial step. The new step is static general with non-linear geometry enabled. The time period for the step is 0.015 s and maximum number of increments is 1000. The increment step is initially 0.001, with a minimum value of 1E-12 and maximum a value of 0.001. These values for increments are defined to be relatively small to ensure that contacts work in the model. If an increment is relatively too big, contacts break in the model and will end up giving incorrect results and the model do not represent reality as it should. Other settings for analysis are shown in Figure 12.

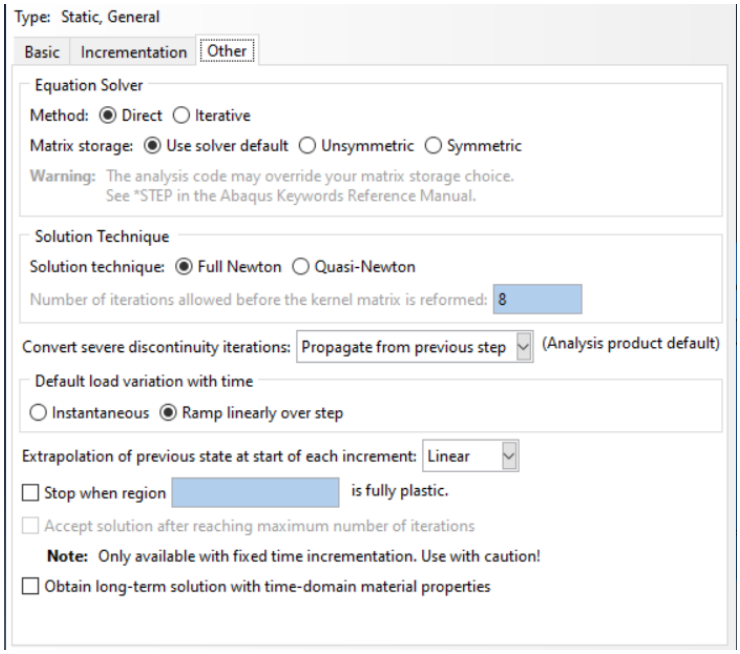


Figure 12. Settings for solver in ABAQUS

3.5 Assembly

The model is assembled from created parts (the wood log and cutting knife). These parts are added to the assembly with an instance tool. A meshing option on the tool is set at dependent, which means part itself will have meshed. After parts are added to the assembly, set is created for the cutting knife. This set is called CuttingEdge and is selected from the cutting knife's cutting edge. The second reference point is added to the wood log in the assembly since only one reference point can be added in the part module. The second reference point is added to the opposite end of the wood log. The coordinates for the second reference point are (0, 0, 140). The cutting knife is moved along *z*-axis to be centred relatively to the log. This is done by using a translation tool. The necessary translation for the cutting knife is half the difference between the lengths of the wood log and the cutting knife. This difference is 20 cm in length which means that translation is 10 cm for the cutting knife. The whole assembly is shown in Figure 13.

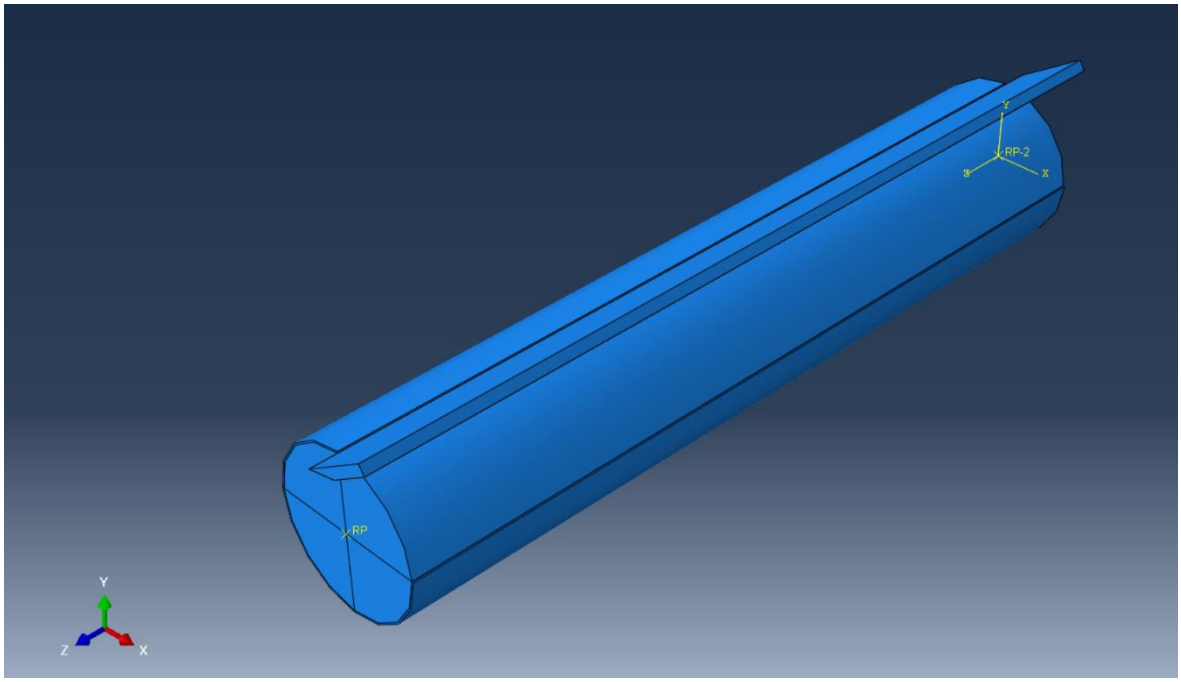


Figure 13. Complete assembly of the model

3.6 Contacts

There are two interaction properties in the model. These are called friction and hard contact. For hard contact, there are two behaviours, tangential and normal. Tangential behaviour is selected as a penalty with the friction coefficient set to 0.4 and other settings are left to default. For normal behaviour, pressure-overclose is set to “Hard” Contact, the constraint enforcement method is set to default and separation is allowed after contact. For the friction there is one behaviour, tangential. Tangential behaviour uses exactly the same settings as hard contact.

There is one interaction created for the model. This interaction is general contact which is created at initial step and then propagated to the peeling step. For contact domain three contact surfaces are selected. These contact surface pairs are presented in the table 4.

Table 4. Contact surface pairs for contact domain in the general contact

First surface	Second surface
Cohesive layer bottom surface	Blade top surface
Peeling layer front surface	Blade top surface

Table 4 continue. Contact surface pairs for contact domain in the general contact

Peel layer bottom surface	Blade top surface

For global property assignment hard contact interaction property is selected. One contact is added to the individual property assignment, which uses different contact property to a global one. The surface pair is wood log inner core top surface and Blade bottom surface with friction interaction property. Other settings in the general contact are left as default.

3.7 Constraints and boundary conditions

There are three boundary conditions in the model, one for the cutting knife and two for the wood log. The boundary condition for the cutting knife is fixed and the boundary condition is applied to the whole blade. The cutting knife boundary conditions are a revolute and cylindrical. One boundary condition is applied to other end of the wood log and the other boundary condition to the opposite end of the wood log. The revolute end is done by using constraint and boundary condition. The used constraint is a coupling and the control point is one of the reference points, with the surface being the wood log inner part's end surface. The coupling type is kinematic and all degrees of freedom are constrained. For the same reference point, the displacement/rotation boundary condition is defined with all translations and rotations, except Rz, which is left free. For the peeling step, Rz is the given value for how much the log needs to rotate during the simulation time. This angle θ is calculated with following equation:

$$\theta = \frac{vt}{r} \tag{5}$$

where θ is the angle which needs to be travelled in the simulation time, v is peeling speed, t is simulation time, and r is the radius of the wood log.

The cylindrical condition has the same constraint settings as the fixed end, but it is located at the opposite end of the wood log. The cylindrical joint is also used other reference point as a control point. The boundary condition is defined to the reference point with free Uz and other degrees of freedom are locked. Rz rotation is defined by using equation (5) for the peeling step. Figure 14 shows which parts and areas have boundary conditions and what boundary conditions are used for these.

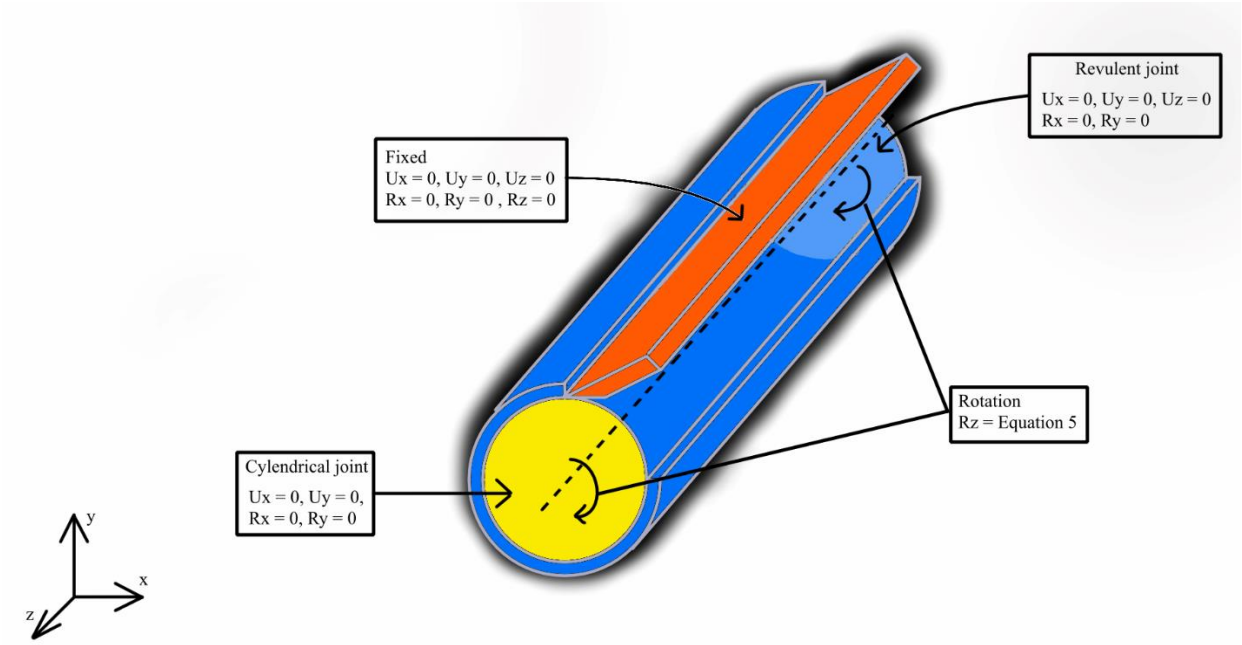


Figure 14. Illustration of the boundary conditions in the model

3.8 Mesh

The wood log and the cutting knife are meshed separately. The cutting knife has global seed size set to 8 and to control mesh individual edges are seeded. The mesh structure is 4x3x40 and there are two edges that have minor difference. These edges are located at *xy*-plane and use 3 elements. Figure 15 shows the mesh for the cutting knife.

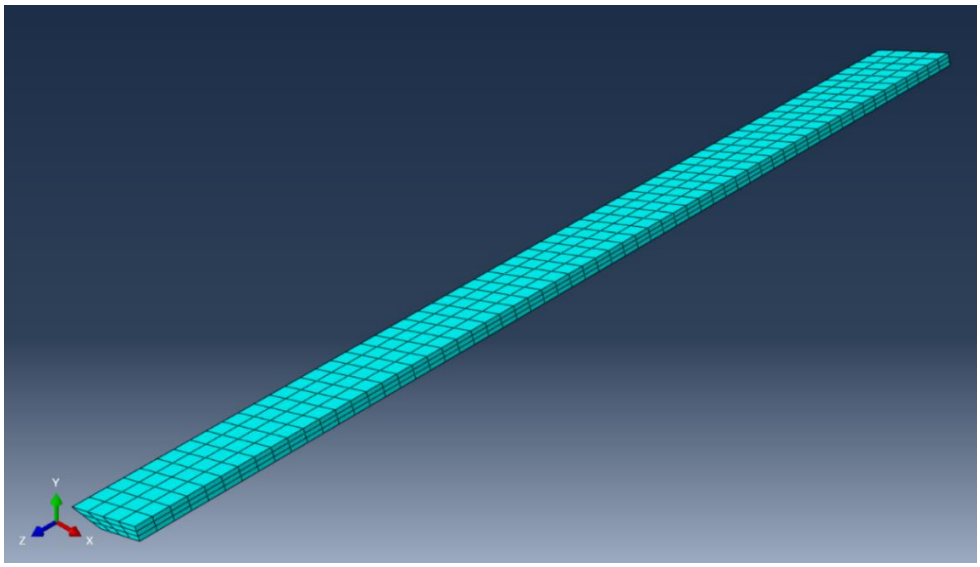


Figure 15. Mesh for the cutting knife

For the cutting knife C3D8IH elements are used. Element is an 8-node linear brick with hybrid formulation and incompatible modes. Incompatible modes are added to the elements internally which add extra degrees of freedoms. Incompatible modes are used to prevent parasitic shear stress when elements are loaded in the bending. Also, extra degrees of freedom are used to eliminate artificial stiffening due to poison's ratio behaviour in the bending. When using normal displacement elements, axial stress linear variation is accompanied by a linear variation of the stress on the perpendicular to the bending direction in the bending case. These will cause an incorrect estimation of the stresses and overestimation of stiffness. By using incompatible modes, these effects can be prevented (ABAQUS Incompatible modes, 2017). For the veneer peeling model incompatible modes are needed because there is severe bending when the peeling layer is separated from the wood log. Incompatible modes are needed to keep stiffness in the correct value because if the peeling layer is excessively stiff this will cause peeling to use more force to bend according to the blade geometry.

Hybrid formulation is used when material behaviour is almost incompressible or incompressible. When behaviour is incompressible, the solution for the problem cannot be solved alone by using displacements' history, because hydrostatic pressure can be added without changing displacements. When material behaviour is almost incompressible, even small changes in the displacements can cause large changes in the pressure. This is why pure displacement solution cannot be used: it is too sensitive numerically, which means computer rounding off could cause the method to fall apart. Hybrid elements calculate hydrostatic pressure by using independent interpolation, and elements use pressure to calculate elastic strain. (ABAQUS Elements, 2017) Hybrid elements are used in the blade because steel is almost an incompressible material compared with wood when the peeling process is in action.

The library for C3D8IH element is standard and the family used is 3D stress. The stack direction for the cutting knife is that top elements are on the knife's spine. The bottom ones will be on the bottom side of the cutting edge.

The mesh for the wood log uses the global seed size of 1.7 and the rest of the settings are as default. Only one edge is seeded by using seed edge. This edge is the arc between the peeling layer in the wood log. The arc is seeded with 10 elements. Figure 16 shows the mesh used for the wood log. The inner part of the wood log uses C3D8IH element and the peeling also

uses the same element. Cohesive layer uses COH3D8 element which is a standard library of cohesive and it is linear. COH3D8 is 3 dimensional and 8 node cohesive elements. Element has element deletion on and viscosity is 0.002. All the elements are hex elements which are used in both parts. Element stacking direction for inner part of the wood log is left as default. For the cohesive layer, the top surface is selected to be the lower end face and the bottom surface is selected to face against the cutting knife. For the peeling layer top surface is the same as peeling layer front surface and the bottom surface is then the end-face surface of the peeling surface.

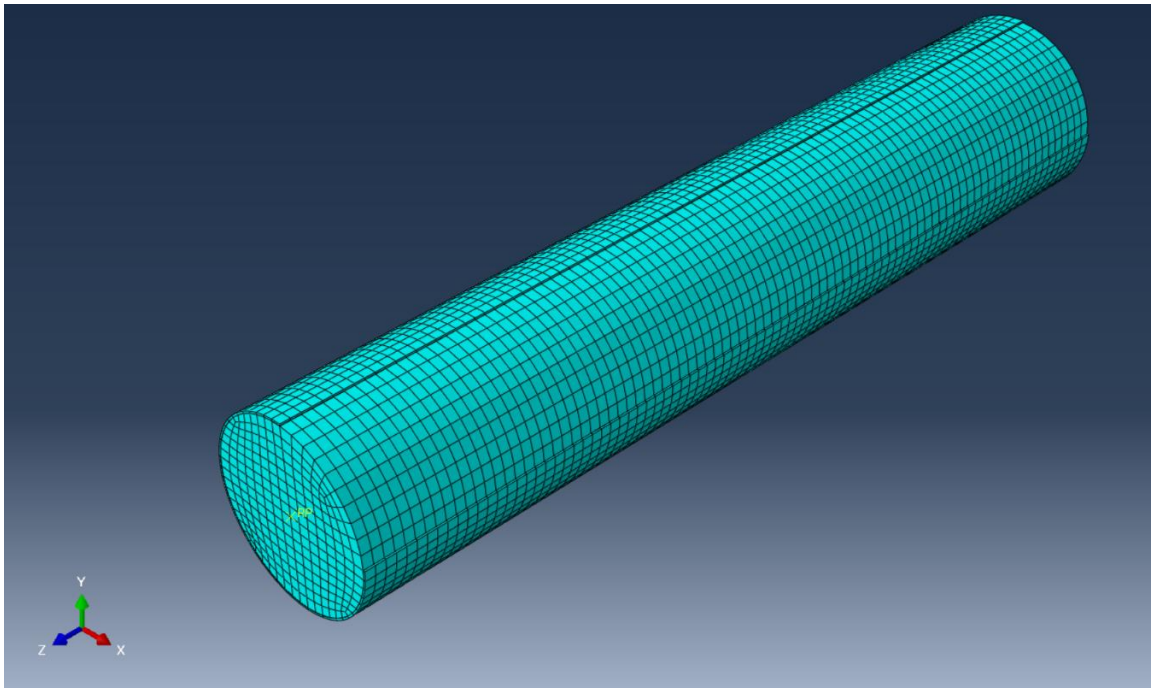


Figure 16. Mesh for the wood log

3.9 Field and history output request

Field output request takes data every increment from selected model area or the whole model. Field output is used to check certain things in the model after simulation for example any element failure in the results. History output is used for taking results from the analysis of the model. The interesting results from the model are the cutting knife's cutting force, which is taken from the cutting knife's CuttingEdge set. History output is saved at every increment and variables are RF1 and RF2. RF1 is tangential reaction, the force and RF2 is horizontal reaction force. To get cutting force which is the counter force of reaction force is

multiplied by -1. This operation is done in the script. Figures 17 and 18 show the examples of the reaction force results after analysis are completed. Figure 17 shows reaction force magnitude results in the cutting knife at 2.28E-3 s and figure 18 shows results at 1.5E-2 s.

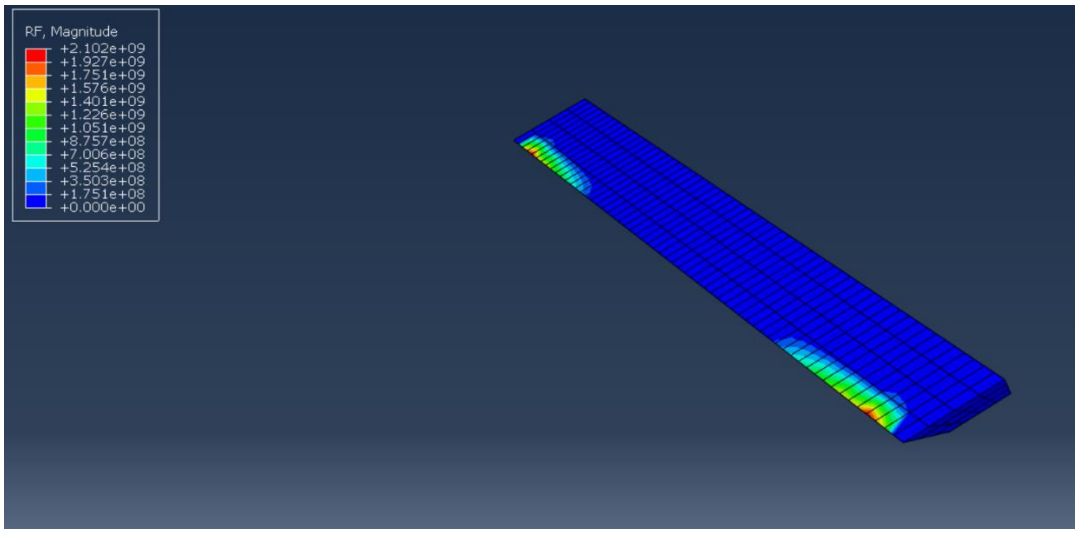


Figure 17. Example of reaction force magnitude at 2.28E-3 s.

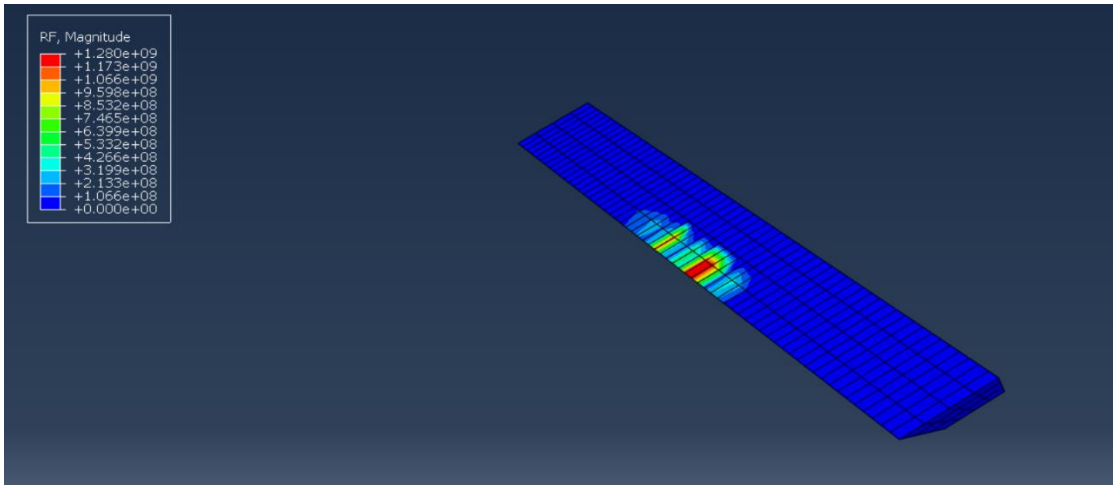


Figure 18. Example of the reaction force magnitude at 1.5E-2 s.

3.10 Job

The model is submitted for a solver by jobs. The job settings used for memory are 95 % maximum memory usage by the solver. Parallelization is used and settings depend on the system which job will be run. For example, a computer has Intel i7-6700 which has 4 cores and 8 threads. The job can use 6 threads of the CPU and leave 2 threads for background operations. Also, for this case multiprocessing mode is threads which will use CPU cache

memory for information passing between threads during a workload of analysis. If the computer has compatible GPU, GPGPU acceleration can be used which make that GPU helps CPU to solve the analysis. GPU acceleration can be only used in the Abaqus/standard analysis. One thing that need to be kept in mind is tokens availed in the Abaqus licence. The necessary tokens are calculated based on amount CPU cores, threads and number of GPUs used in the analysis. If using super or a cloud computing multiprocessing, the mode can be changed to MPI, but this requires the system to supports MPI (message passing interface). Other vice thread multiprocessing can be used. Parallelization is used to divide the workload between cores or threads to make solving the analysis faster than using one core/thread on the CPU.

3.11. Scripting

There are different scripts that are created for the work. These are scripts for the model, reading results, plotting the results and sensitivity analysis. Model script is first used. Abaqus model has been made in the form of python script. The use of script is to ensure that changes in the model parameters happen as intended and the process is automated. For some reason script model cannot be made to give the same results as known when working the model and for this reason the script model is not used in the work. Figure 19 presents the working principle of the model script. The whole python script is presented in Appendix 1 for model script.

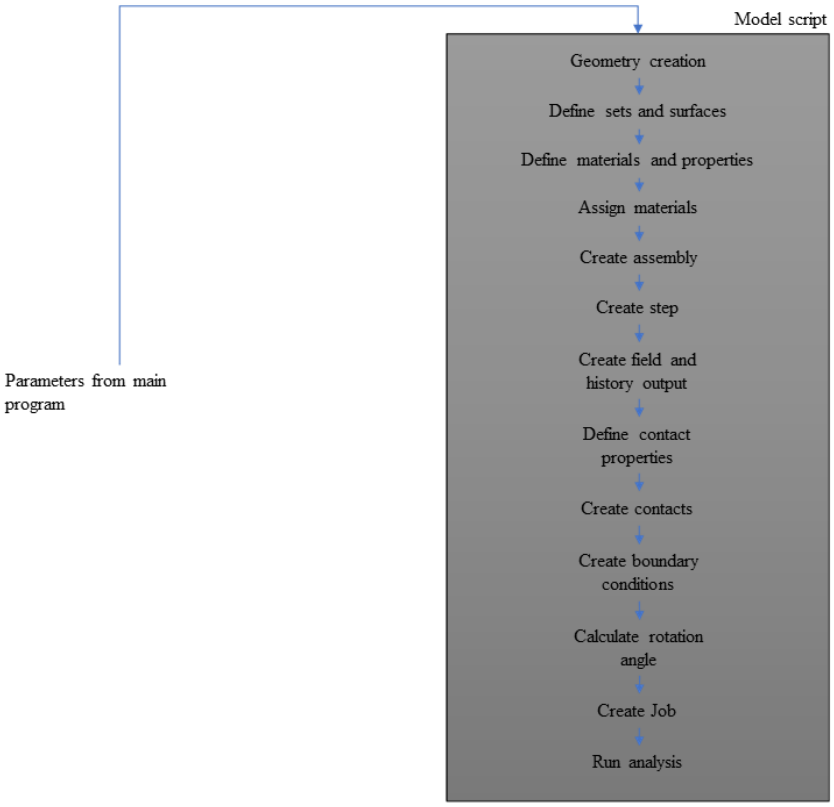


Figure 19. Flow chart of the model script.

Results processing is done with a script which reads data from files. Plots are figured from corresponding data. This script is also written by using Python. For plotting a plotly library is used, and for data a handling Numpy library is used. With the plotly, figures are created as html plots and the user can check a plot by using the web browser. For example, a 3D figure user can rotate it in the browser according to the coordinate axes and other usual things like zoom in or zoom out. In figure 20, the working principle of the plotting script is explained. The whole python script is presented in Appendix 2 for plotting script.

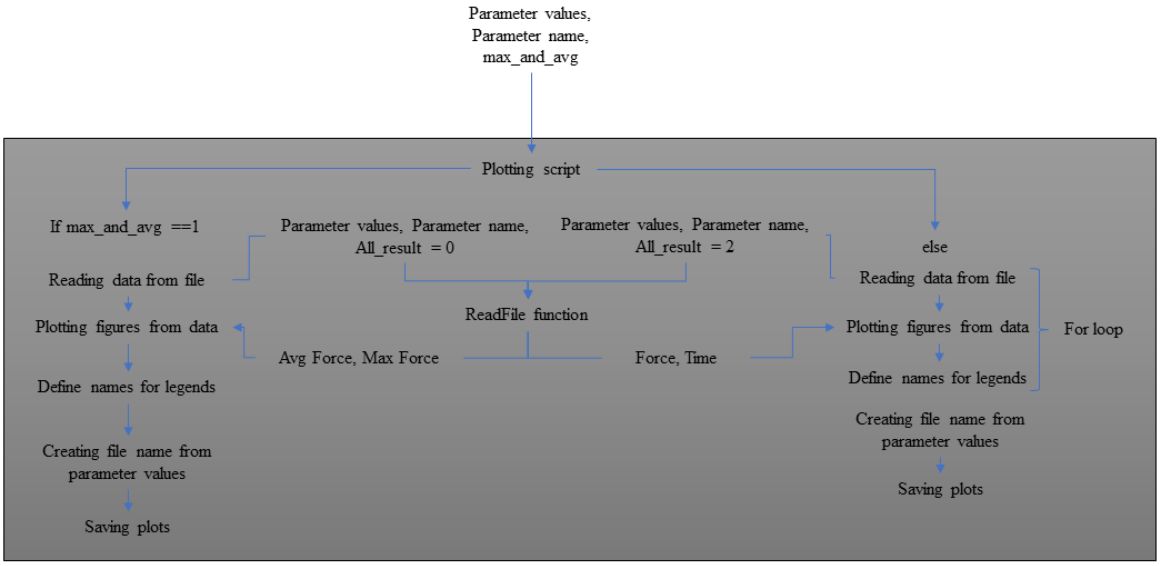


Figure 20. Flow chart of the plotting script.

Reading data from results is done with the ReadFile function. The ReadFile function reads data from text files and converts data to the numpy vectors. There is an option for processing all results at once or one parameter at a time. ReadFile function also makes needed conversion for force from dyne to newton. In Figure 21 is presented how the ReadFile function works. The whole python script is presented in Appendix 3 for Readfile function.

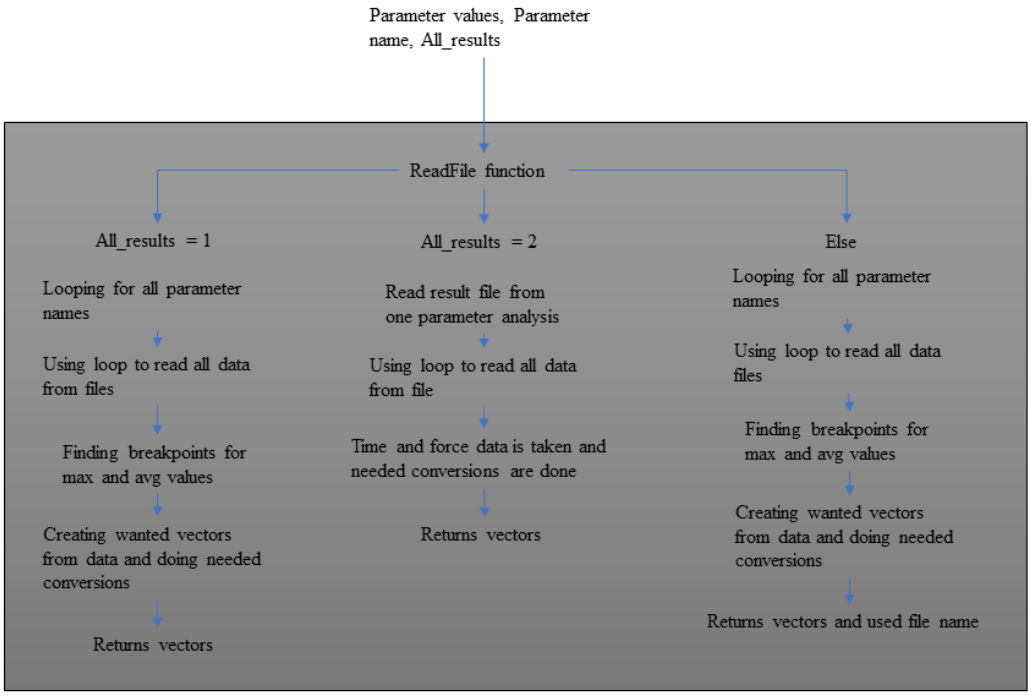


Figure 21. Flow chart of the ReadFile function.

Sensitivity analysis is done by creating python script which handles sensitivity analysis. The script uses a SALib library (Herman and Usher, 2017) for sensitivity analysis which has different sensitivity analysis function and tools. Also, Numpy is used to process numerical operation and things in the script. Figure 22 presents the working principle of the sensitivity analysis script. The whole python script is presented in Appendix 4 for sensitivity analysis script.

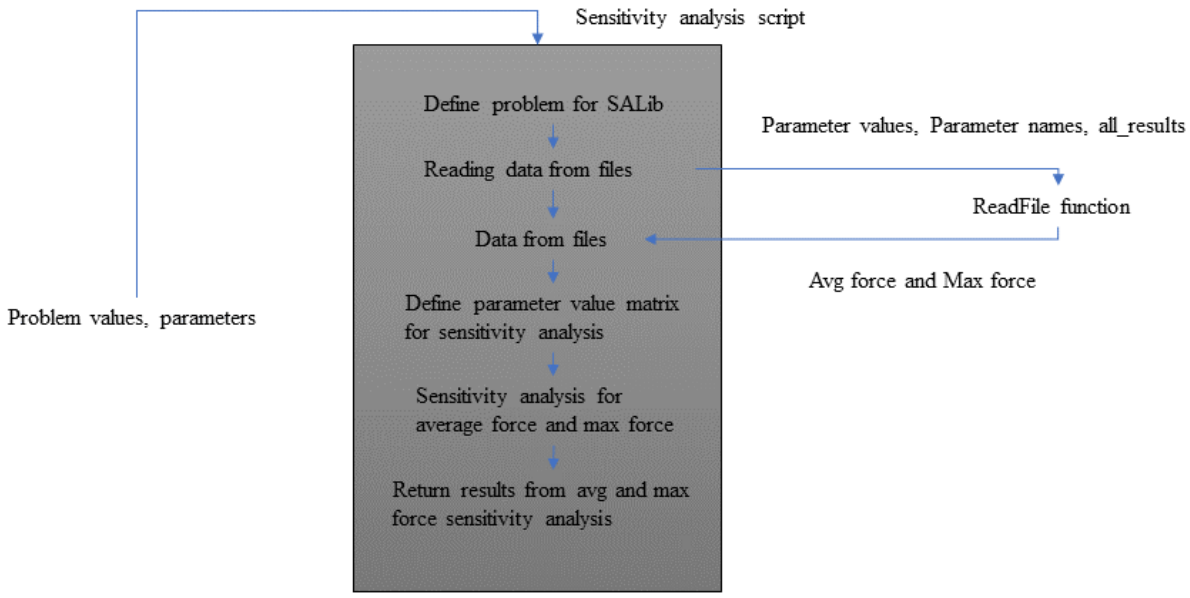


Figure 22. Flow chart of the sensitivity analysis script.

For results and sensitivity analysis, a main function is created. The main function processes all things and above explained scripts and functions are called by main function. In the main function, parameter values and step are defined and created. Figure 23 presents the working principle of the main function. The whole python script is presented in Appendix 5 for the main function.

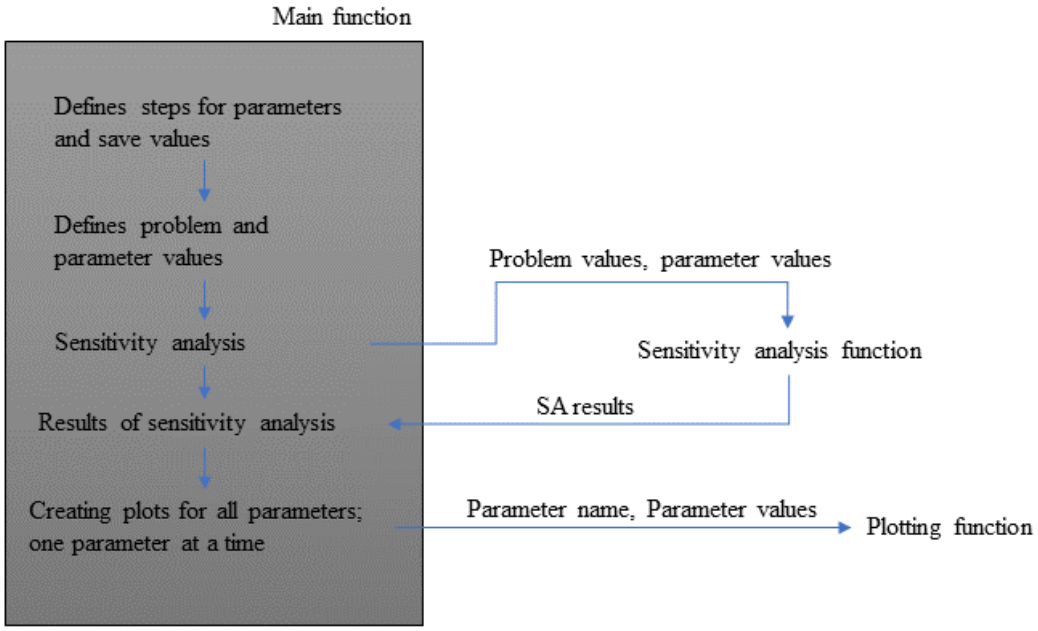


Figure 23. Flow chart of the main function.

4 Results

The veneer peeling model is analysed and the results of the analysis are presented here. First, let us review conditions for the veneer peeling model for analysis. The veneer peeling model consists of two individual parts which are the wood log and the cutting knife. The wood log is made of Spruce Sitka and the cutting knife is made of steel. The wood log has a radius of 25 cm, and it has the peeling layer in which thickness is one of the parameters for analysis. The wood log has two boundary conditions. These boundary conditions can be seen in Figure 14. In figure 14, the yellow end has a cylindrical joint and light blue has a revulent joint. In Figure 14, the cutting knife is in orange and has fixed boundary condition. Also, both ends of the wood log have rotation around z-axis which is applied to the middle of the end surfaces yellow and light blue. The amount of rotation is calculated by equation (5).

The model of the veneer peeling process is analysed with the different parameter values and parameter are presented in Figure 11. A total of four parameters is changed one at a time. These parameters are a position of the cutting knife tip on x-axis, a cutting knife clearance angle, peeling velocity and thickness of the peel. An example of the results for the veneer peeling process is shown in Figure 24.

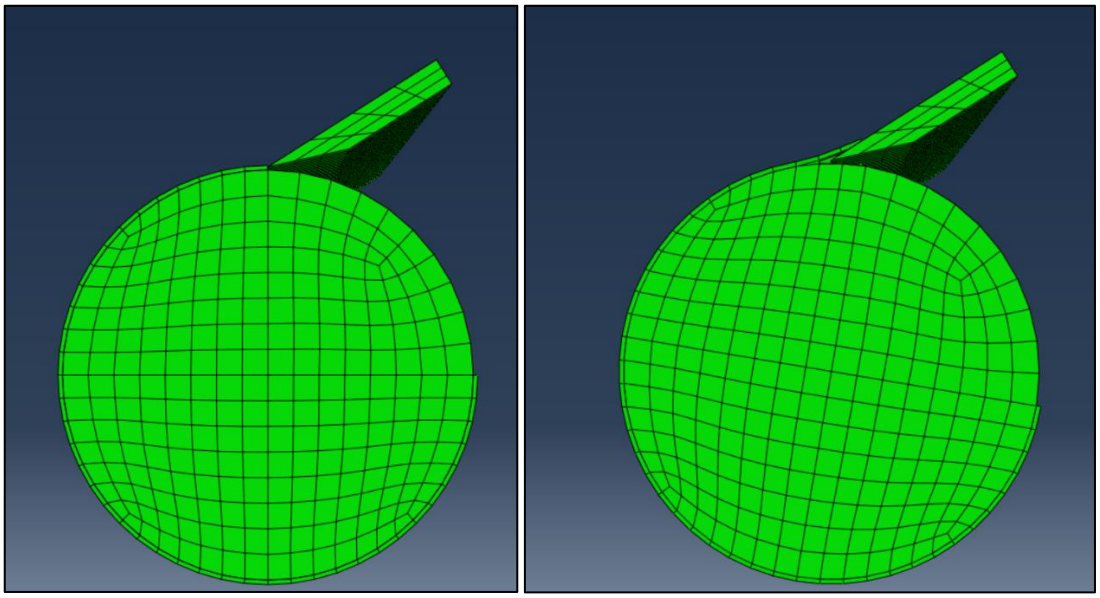


Figure 24. Example of the results for Veneer peeling process model

In Figure 24, the picture on the left shows analysis at the start and the picture on the right shows end results of the analysis. From the right side picture in figure 24 it may be seen that the peeling layer has been separated from the core of the wood log.

Sensitivity analysis is done for the model by changing one parameter value at a time. Parameters that are selected for analysis are following peel thickness, the rake angle of the cutting knife, peeling speed, and cutting knife tip position. The total amount of analysis used in the analysis is 36. Table 5 present a range for different parameters for each parameter, with a total of 9 parameter values for a different parameter. Parameters' maximum and minimum values in table 5 are given by the Raute company. All model analyses are computed using CSC computing cluster Puhti, which is more time efficient than the workstation. One analysis in Puhti takes around 2 h and the same analysis on workstation takes around 8 h. Also, when using Abaqus in workstation it lost a connection to the licence server and stopped analysis. For these reasons, the CSC Puhti computing cluster is chosen for running the analysis of the model.

Table 5. Minimum, maximum and step values for parameters used in sensitivity analysis.

Parameter	1 Minimum value	2	3	4	5	6	7	8	9 Maximum value
Peel thickness (mm)	0.5	0.938	1.375	1.812	2.25	2.688	3.126	5.563	4.0
Clearance angle of the cutting knife (deg)	0	3.4375	6.875	10.3125	13.75	17.1875	20.625	24.0625	27.5
Peeling speed (m/s)	2	2.375	2.75	3.125	3.5	3.875	4.24	4.625	5.0
Cutting knife tip position x (mm)	0	0.025	0.05	0.075	0.1	0.125	0.15	0.175	0.2

Results from analyses are saved to a text file and every analysis has its own file for results. Results from text files are plotted with a plotting script and sensitivity analysis is done with a sensitivity script. These scripts have been explained in the Scripting chapter. Sensitivity analysis is done with SALib (Herman and Usher, 2017). Inputs are manually made for SALib (Herman and Usher, 2017) because no sampler is used due to the one-at-a-time approach where no sampler is used for defining parameter values.

These parameters are selected to see how changing these parameters affects the cutting knife's cutting force in a tangential direction. When the parameter is not studied it uses the default value for analysis. These default values are presented in Table 6.

Table 6. Default values for parameters in analysis

Parameter name	Default value
Thickness of the peek	3.5 mm
Clearance angle of the cutting knife	0 deg
Peeling speed	2 m/s
Cutting knife's tip position	(0,0) mm

The reason for analysing one parameter at a time is to see the importance of the parameter for analysis results. Also, the sensitivity of a parameter can be seen, meaning how sensitive the model is for that parameter. Results are plotted using the created python scripts which are discussed in the Scripting chapter. Plots are made using a python library plotly. Addition to the plotting of the results from the sensitivity analysis is done for results by using the SALib python library (Herman and Usher, 2017). For sensitivity analysis, the maximum and average force are inputted. The RBG – FAST method is used for sensitivity analysis and the problem variable is defined manually for analysis because a manual one-at-a-time analysis is done for parameters. RBG – FAST method is selected because it is needed fewer parameter runs than other methods to getting results for first order indexes. It is not certain if 36 parameters run is enough to get sensitivity analysis results with the RBG – FAST method.

Results from different parameter analyses are presented next. First results over simulation time for one parameter at a time are presented. In the plots, legends have the value of the parameter in the analysis. First blade tip position results are presented in Figure 25.

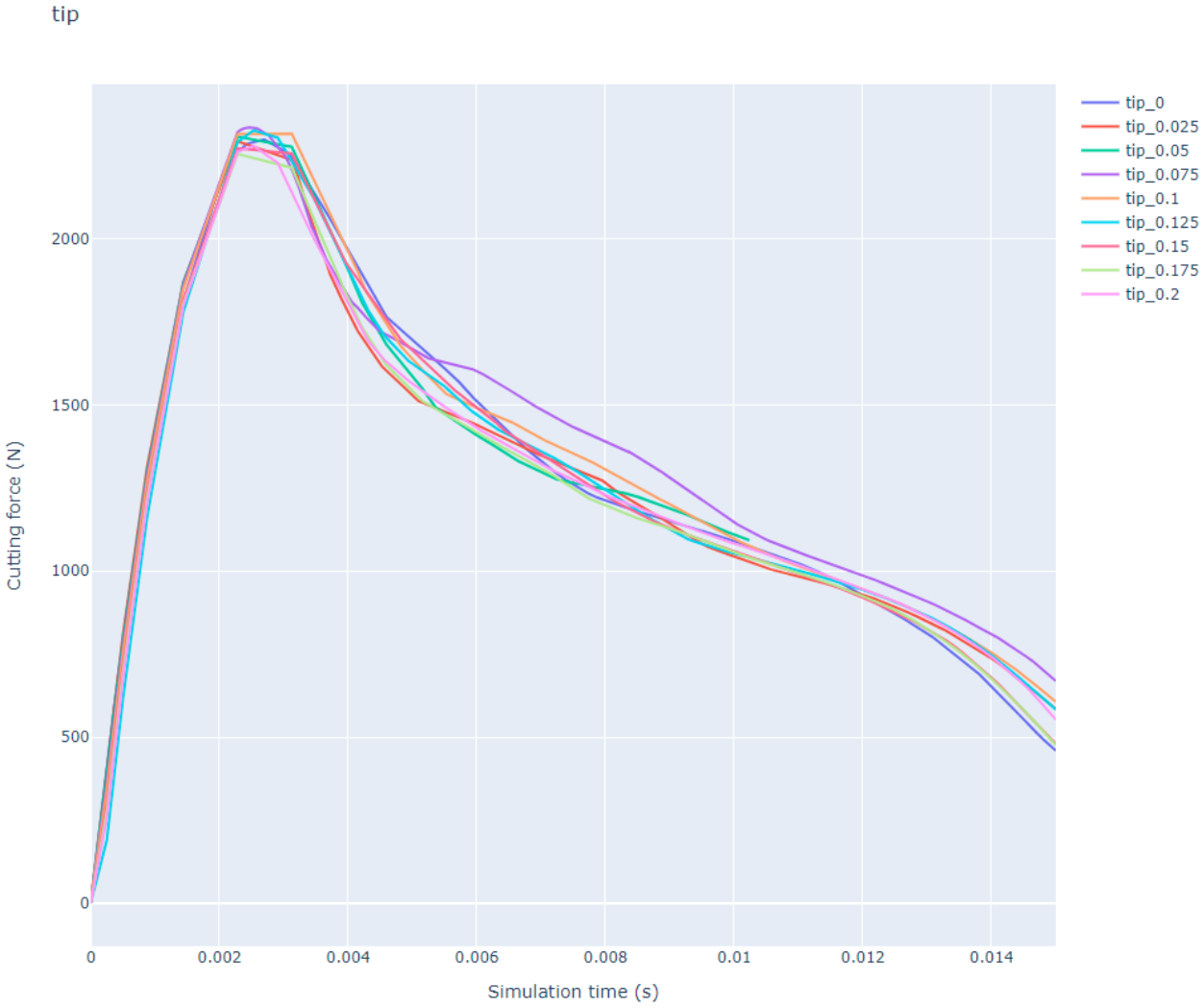


Figure 25. Cutting knife tip positions results over simulation time

In Figure 25, the legends tells the value of the tip position in the *x*-axis. For example, tip_0.025 means that tip is moved 0.025 mm in the positive direction of the x-axis. In Figure 25 it may be seen that all the curves are together from start to around 0.002 s. Maximum values are close together for different tip positions. After the maximum value is achieved, all curves start to decline and most of the curves are very close together. In Figure 25 two outliers may be noticed. The first one is tip_0.05 solution has not converged, and the second is tip_0.075; results are higher than the rest of the others from 0.006 s onwards. The tip_0.075 results might be caused by a numerical nature of the solver or complex nature of the model. Figure 26 shows results for the clearance angle of the cutting knife.

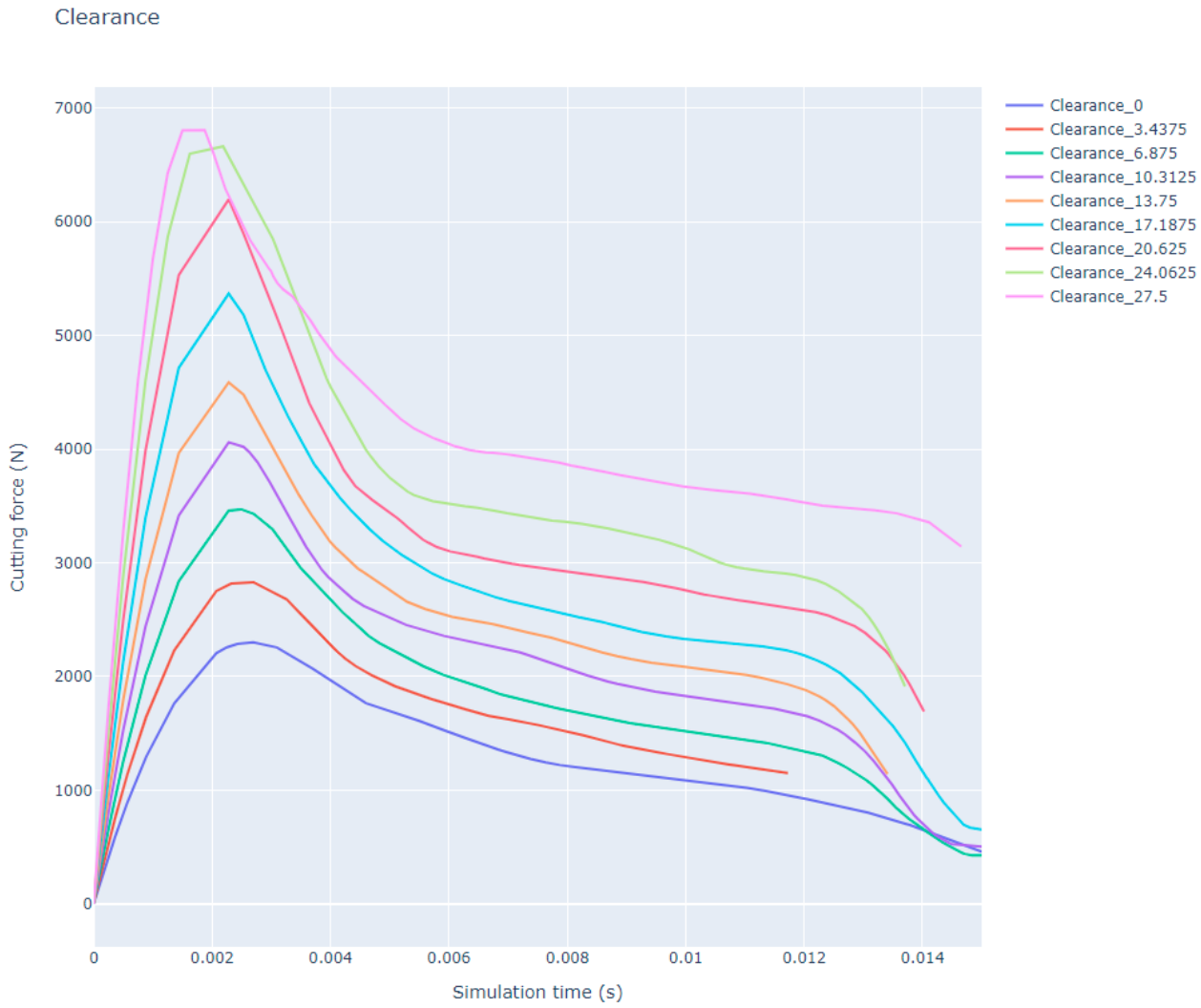


Figure 26. Clearance angle results over simulation time

In Figure 26, the legends tell the clearance angle in degrees for the cutting knife for example Clearance_3.4375 means that clearance angle is 3.4375 deg. In Figure 26 it may be seen that by increasing clearance angle cutting force increase as well. All different clearance angles have approximately the same shape and maximum value is around the same area for all angles. In Figure 26 can also see that many of the angle results have not converged. This is due to coarser mesh which have been used for these results since with finer mesh results converged with no problems. Figure 27 presents results for peeling speed.

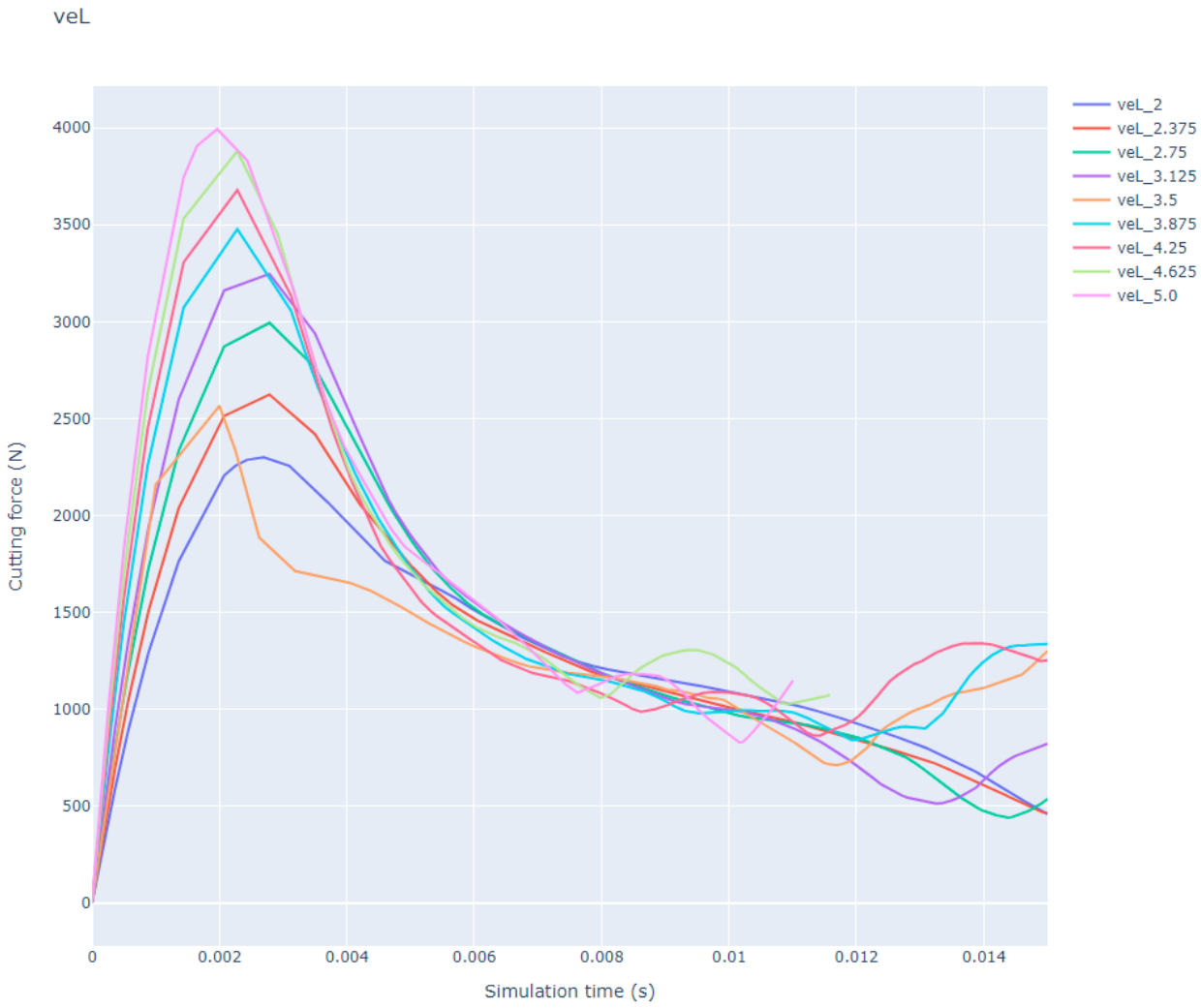


Figure 27. Peeling speed results over simulation time

In Figure 27, the legends tell value for different peeling speeds: for example, veL_3.5 means peeling velocity of 3.5 m/s. In Figure 27 it may be seen that increasing peeling velocity increase cutting force's maximum value at the start of the peeling process. Values seem to drop approximately to the same general area after an initial peak. This cannot be said for certain because there are multiple parameters that have oscillation in the results. Also, in Figure 27 it may be seen two the highest peeling speeds have not converged. This might be caused by coarser mesh or complexity of the model. In Figure 27 it may also be seen that for some peeling speeds cutting force is increasing instead of decreasing as in Figure 25 and 26. Figure 28 presents results for the thickness of the peeling layer.

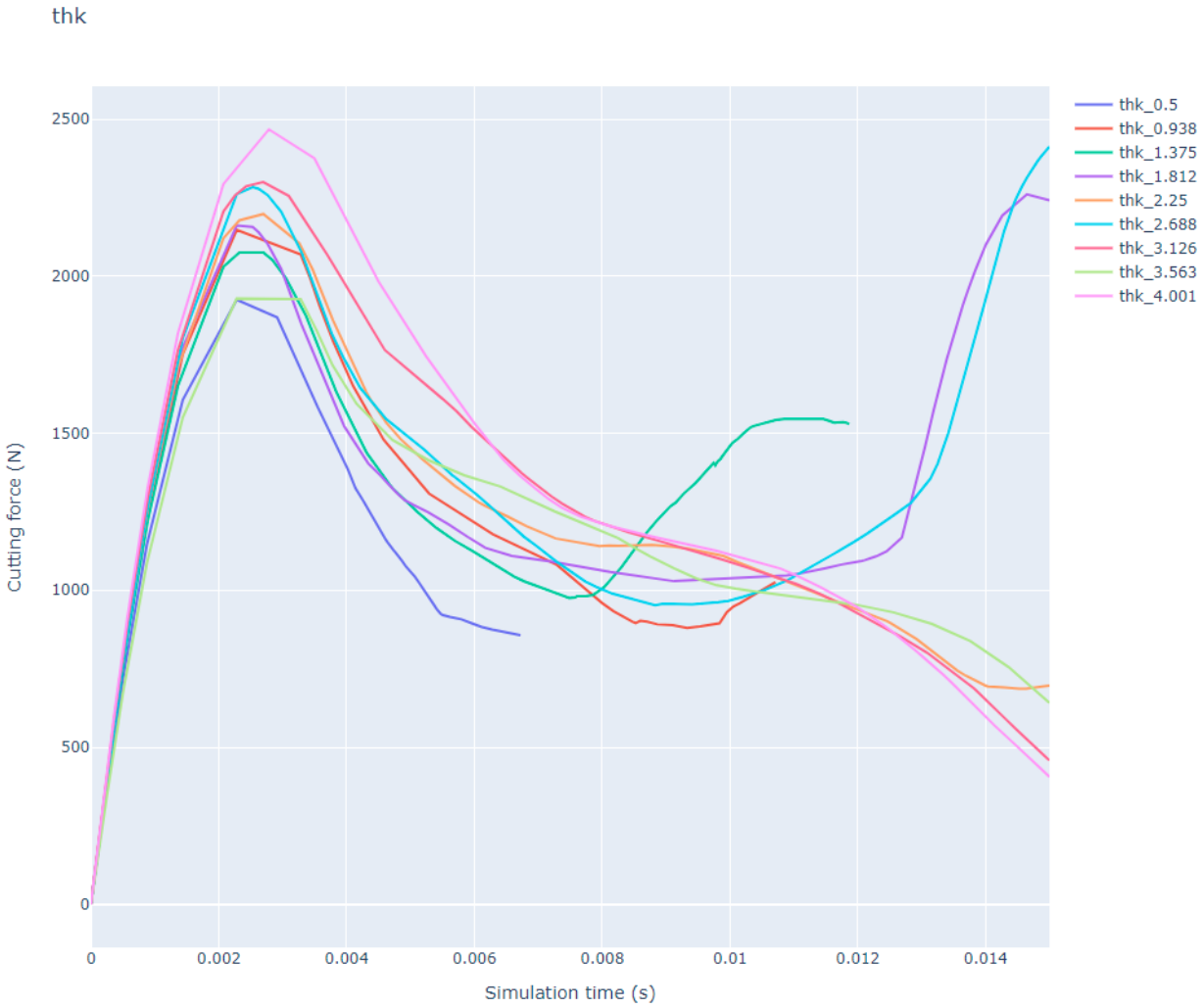


Figure 28. Thickness results over simulation time

In Figure 28, the legends tell the value of the thickness: for example, thk_2.25 means that thickness of the peel is 2.25 mm. In Figure 28 it may be seen that up to an initial peak all the curves are close to each other. Also, increasing thickness appears to increase peak cutting force. In Figure 28 it may also be noticed that there are three thickness values that have not converged. Also, thickness values 2.688 and 1.812 seems to oscillate because both reaches higher peak at the end than at the start. In addition, thickness 1.812 seems to decline at the end of simulation time. Oscillation can be seen more clearly on the thickness 1.375 despite its solution not being converged. There are some results that show same behaviour when comparing with other results in Figures 25, 26 and 27. For thickness results, it is very hard to say how reliable results are from 0.004 s onwards because of variation in the results.

5 Discussion

To further investigate results presented in the figures 25-28, maximum and average cutting force are plotted over parameter values. Maximum cutting force is taken from 0 to 0.004 s of simulation time. Maximum is taken in this time span because in some cases values starts to rise again at the end, which is not line with all results. The average cutting force is taken from 0.004 – 0.01 s of simulation time. This time span is used for an average because initial peak value does not want to get on average and want to see as representative average as possible. Also, for some parameters cutting force values starts oscillating heavily after 0.01 s. Post processing of the results is done by using python scripts which are explained in the chapter Scripting. Next in Figure 29 maximum and average cutting forces of the different tip positions are presented.

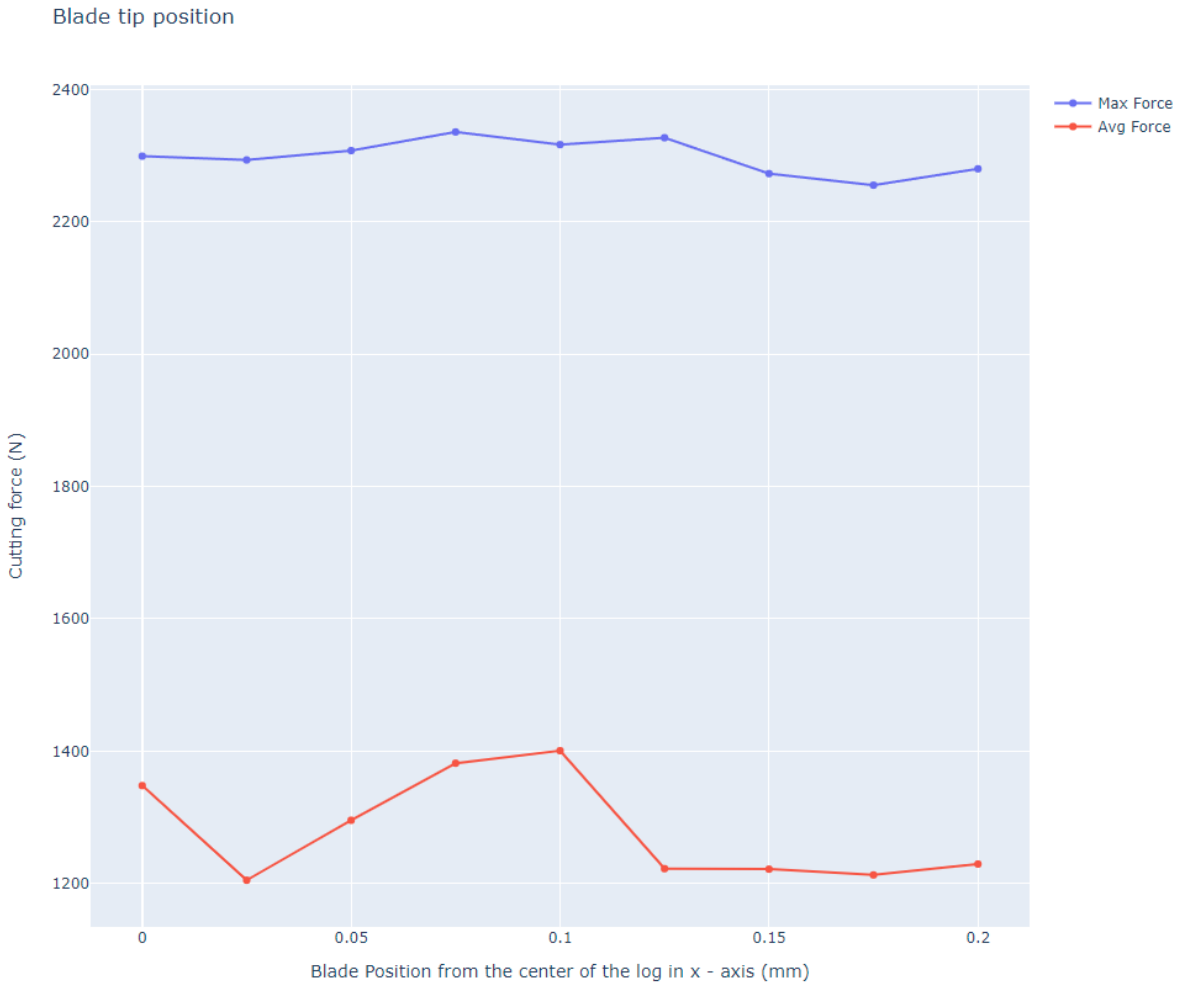


Figure 29. Average and maximum cutting forces for the different cutting knife's tip positions

In Figure 29 it may be seen that maximum cutting force does not change much, which was also seen in Figure 25. Average force, on the other hand, has some interesting behaviour. From a zero average, force drops lower and starts increasing up to 0.1. After 0.1, average again values drop lower and stay approximately at the same value range. For this amount of data it is hard to say whether there is any trend to the average force because an initial increase can be caused by the numerical solver or noise in the simulation. The blade tip position does not seem to be a sensitive parameter for maximum cutting force and for average force changes are hard to be sure but looks like it is not significant changes that tip position makes. Next, Figure 30 presents the results for different clearance angles of the cutting knife for maximum and average cutting force.

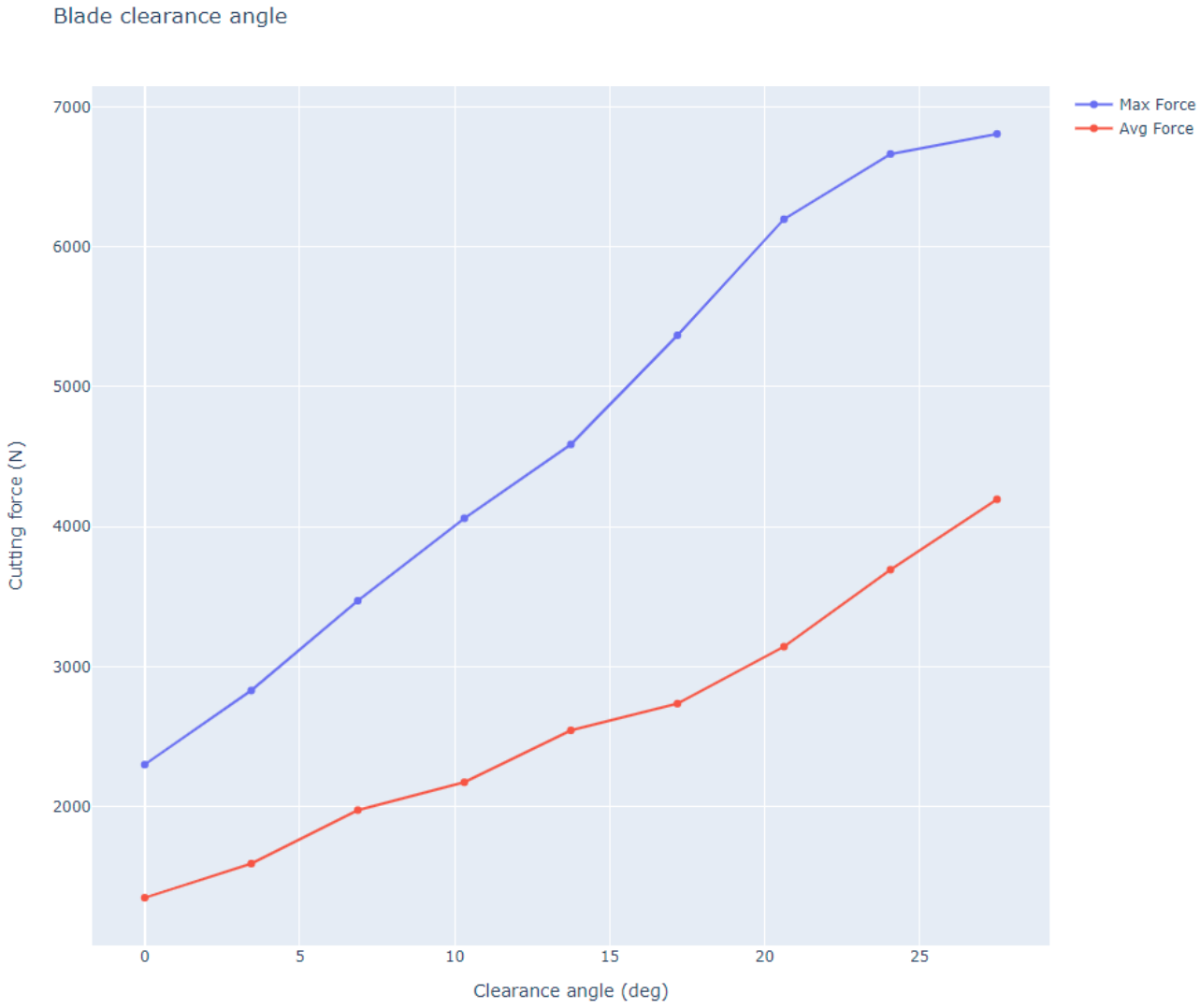


Figure 30. Average and maximum force for different clearance angles for the cutting knife

In Figure 30 it may be seen that increasing the clearance angle of the cutting knife also increases maximum and average force. The shape of the maximum force curve seems to be essentially linear behaviour. The same observation can be made for average force. From starting the value of the clearance angle to the end value, the maximum cutting force triples and the average force seems to quadruple cutting force. Figure 30 observations can also be seen in Figure 26: increasing clearance angle increases cutting force at the knife's edge. The clearance angle seems to be a sensitive parameter in the model. Next, Figure 31 shows results for different peeling speeds for maximum and average cutting force.

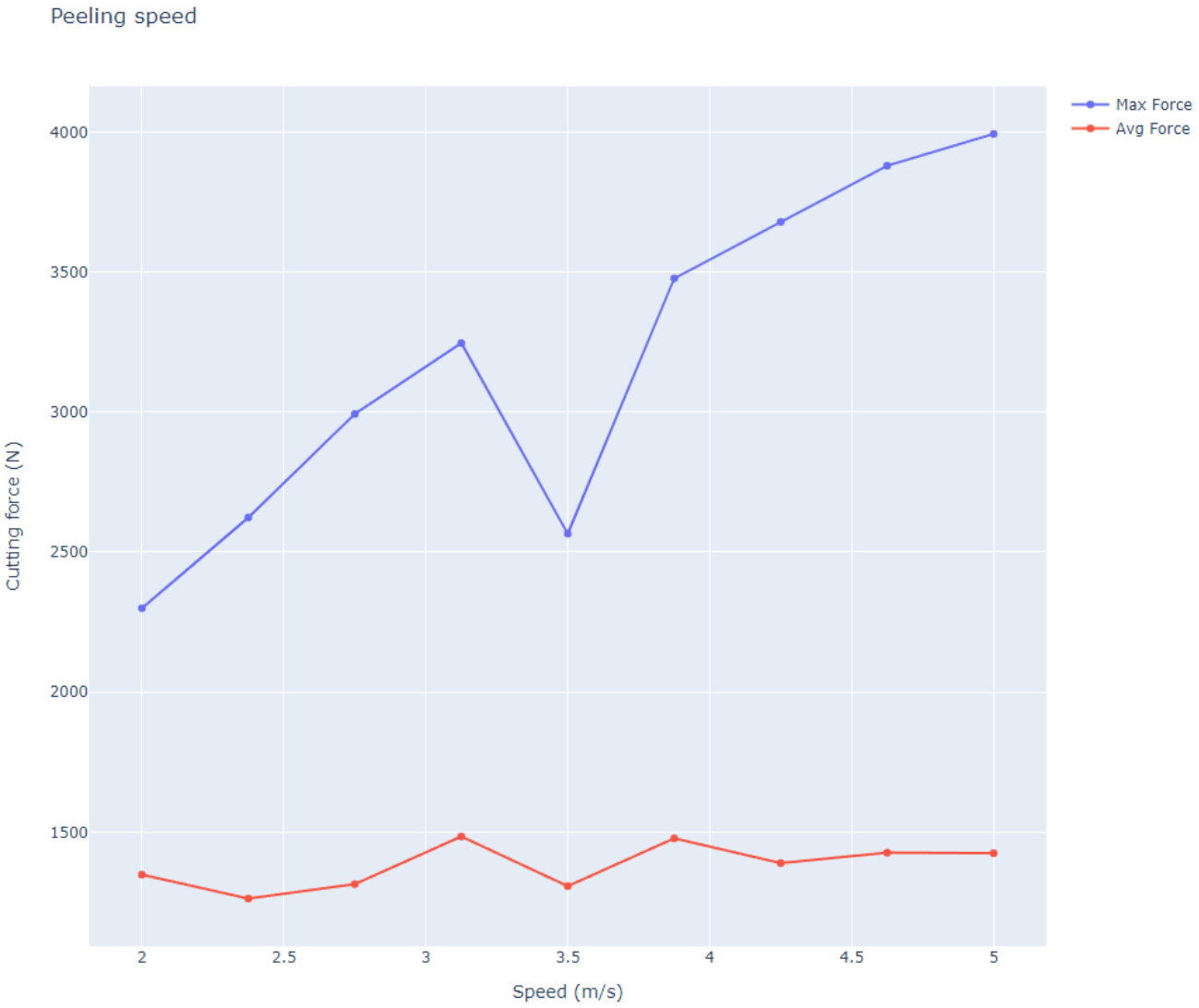


Figure 31. Average and maximum cutting force for different peeling speed values

In Figure 31 it may be clearly seen that 3.5 m/s peeling speed maximum cutting force value is not correct. This value can also be seen in Figure 27, which is not in line with other results for its values. For 3.5 m/s, something happened during analysis. If the 3.5 m/s value is moved

to line with other, results for maximum force can see that increasing peeling speed increases maximum cutting force. Cutting force seems to have linear behaviour between cutting force and peeling speed. Average cutting force has only small variation. The average appears to stay around the same value area for all different peeling speeds. Maximum cutting force doubles its value from start to end. The average cutting force does not have same increase as maximum, but average seems to increase only slightly. Peeling velocity seems to have more important role for maximum cutting force than for average. This means that peeling velocity is a more sensitive parameter for maximum cutting force than for average. Next, Figure 32 shows results for different thickness of the peel values for maximum and average cutting force.

Figure 32. Average and maximum cutting force for different thickness values

In Figure 32 it may be seen that thickness value of 3.56 mm has some error in the results because it is so out of line compared with other results near it. Maximum force appears to have an increasing trend when thickness increases if 3.56 mm is moved in line with others. From the average force it may be seen that 0.5 mm thickness does not have average value. The reason for this can be seen in Figure 28: 0.5 mm analysis has not converged and the time range for average has been taken between 0.004 s and 0.01 s. The thickness of 0.5 mm has stopped around 0.007 s so there is no full data from the desired time range, and its average is set at zero. The average cutting force appears to increase up to thickness 1.812 mm while increasing thickness. From 2.25 mm onward, average cutting force seems to oscillate. Oscillation seems to increase compared with lower ones and upper ones. When checking 2.688 mm and 3.563 mm thicknesses, it may be seen that 3.563 has higher cutting force than 2.688 mm. The same observation can be made for thicknesses 3.126 mm and 4 mm. After these observations, considering the average onwards from thickness 2.25 mm average might increase only very slightly up to the thickness of 4 mm. maximum cutting force can be seen to increase about half from start to finish and average force increase about 2/5 amount.

When examinung all Figures 29-32 to check which parameters' sensitivity and ranking them for maximum and average cutting force, the following observations may be made. The biggest impact on maximum force is the clearance angle which quadruples the cutting force from a minimum to maximum angle. The second highest impact appears to have peeling velocity which approximately doubles maximum cutting force. Thickness of the peel has only low impact on the maximum cutting force because it increases cutting force by approximately 0.5. The blade tip position does not have impact on maximum cutting force since value stays approximately around the same value.

When checking the same things for average cutting force following observations may be made. The blade clearance angle has the biggest impact on average force since it approximately doubles average force in its value range. Thickness of the peel seems to impact average force up to 2 mm. After 2.5 mm, average force oscillates but a trend of oscillation is increasing slowly. Peeling speed seems to have minor sensitivity to the average cutting force because average force has slightly increased trend, but this increase is very small even compared with the thickness of the peel. The blade tip position's effect on average cutting force is hard to be sure because from average it is hard to see any trends or be sure of the effect. If we consider that maximum increase is 1/5, it could be as important as peeling

speed because the clearance angle and thickness of the peel have much higher impact on average cutting force. Table 7 presents a recap of the sensitivities for maximum and average cutting force for different parameters. In Table 7 ranking is done by using numbers where 1 has most impact and 4 has the least impact on selected force.

Table 7. Parameter importance for maximum and average cutting force

Parameter name	Importance for maximum cutting force	Importance for average cutting force
Thickness of the peel	3	2
Clearance angle of the cutting knife	1	1
Peeling speed	2	3
Cutting knife's tip position	4	4

In Table 7 it may be seen that for both maximum and average force the most sensitive parameter is the clearance angle of the cutting knife. Then it is a link between the thickness of the peel and peeling speed: the peeling speed is more sensitive to maximum cutting force and thickness of the peel is more sensitive to average cutting force. The cutting knife's tip position is the least sensitive parameter for both maximum and average cutting force.

For results sensitivity analysis is done by using SALib python library (Herman and Usher, 2017). RBG-FAST is selected as an analysis method. Maximum and average forces are submitted separately as results from model to the sensitivity analysis. Input is manually made to match results parameters in the form that analysis understand. Since the total number of parameter runs is 36 for sensitivity analysis, this number of runs might not be big enough for RBG-FAST to give results. Table 8 presents results of the sensitivity analysis for model results for 36 different parameter values.

Table 8. Sensitivity analysis results for run of 36 parameter values

Index	Maximum cutting force	Average cutting force
First order index	NaN	NaN
First order confidence	NaN	NaN

In the results in Table 8, it may be seen that the sensitivity analysis using RBG-FAST could not say anything about sensitivities of parameters. This is probably due to an insufficiently large sample size. Sensitivity analysis needs a sufficiently large sample size because analysis tries to fill probability space to see how different parameters affect the given results.

6 Conclusion

From the results of parameter analysis for the veneer peeling process, the following conclusions may be made. In Table 7 it may be seen that most important parameter for veneer peeling process when considering tangential cutting force is the clearance angle. In Table 7, clearance angle has most impact on maximum and average force. Also, in Table 7 it may be seen that cutting knife tip position has least important for maximum and average cutting force. When, checking peeling speed and peel thickness both are equally important in the table 7. Both parameters share second place after the clearance angle. The difference between peeling speed and peeling thickness is that peeling speed is more important for maximum cutting force and peeling thickness is more important for average cutting force.

When checking results from sensitivity analysis in Table 8, the following conclusions can be made. The conclusions that can be made are more runs are needed for analyser to say anything about sensitivity to parameters. Sensitivity analysis needs to have a sufficiently large sample to fill the probability space and calculate connections between input and output variables.

Attempts were made to automate the veneer peeling process analysis with python scripting, which is supported by ABAQUS. A script of the model was created in Python and written in a fully automated way. Automation is done by using variables for parameters which will change. When testing the script and checking results from the script some interesting elements were noticed in the results. Results did not match at all with reference results and after trying to tune the script to make results match between the two. The decision was made to leave the script model out of the picture because the result cannot be made to match and more time spent to get it working was not worth it for this study. Instead, the one-at-the-time method was applied for the parameter analysis, which was done manually instead.

Aspects of this work which can be look further in other works are sensitivity analysis and doing the python script model, which gives correct results. Both of these aspects can be combined for future work as sensitivity analysis needs a larger sample and automated script makes running analysis more time-efficient and reduces the risk of the human error in the large sample for parameters. More efficiency can be achieved when using the script because

the script process can be fully automated and the next analysis can be queued immediately after the previous analysis. When the process is automated and handled by script, it minimizes the human error when applying values for different parameters. In addition to sensitivity and python script, parameter sensitivity analysis can be performed of the veneer peeling lathe for dynamic analysis. Dynamic analysis has been planned for the thesis but had to left out because of an amount of time it took to do Quasi-static analysis runs for the model and time used try to make python script work.

References

ABAQUS Elements. 2017. Solid (continuum) elements. [Web documentation]. [Cited 2022-03-09]. Available: https://abaqus-docs.mit.edu/2017/English/SIMACAEELMRefMap/simaelm-c-solidcont.htm

ABAQUS Friction. 2017. Frictional Behavior. [Web documentation]. [Cited 2022-03-09]. Available: https://abaqus-docs.mit.edu/2017/English/SIMACAEITNRefMap/simaitn-c-friction.htm

ABAQUS Incompatible modes. 2017. Continuum elements with incompatible modes. [Web documentation]. [Cited 2022-03-09]. Available: https://abaqus-docs.mit.edu/2017/English/SIMACAETHERefMap/simathe-c-incompatible.htm

Dogan, F., Hadavina, H., Donchev, T. and PS, B. 2012. Delamination of impacted composite structures by cohesive zone interface elements and tiebreak contact. In: Central European Journal of Engineering, Volume 2. Pages 612-626. [Cited 2022-03-09].

FEM Shapes picture. 2011. FEMShapes.gif. [Picture]. [Cited 2022-03-09]. Available: http://3.bp.blogspot.com/-rUQMZ1BuVnI/TiLwGJlGGOI/AAAAAAAAASw/fkaV3Z1aQFo/s1600/FEMShapes.gif

Green. David, W., Jerrold, E., Winandy. David, E. and Kretschmann. 1999. Wood Handbook:
Wood as an Engineering Material Chapter 4. Wisconsin: Forest Products Laboratory USDA Forest Service Madison.

Harish, A, 2020. Finite Element Method – What Is It? FEM and FEA Explained: SIMSCALE Blog. 20th December 2020 [viewed 9 March 2022]. Available: https://www.simscale.com/blog/2016/10/what-is-finite-element-method/

Herman, J. and Usher, W. 2017. SALib: An open-source Python library for sensitivity analysis. In: Journal of Open Source Software, volume 2. Number 9. doi:10.21105/joss.00097

Kazák, V. 2008. Cohesive Zone Modelling. ResearchGate [Online]. [Cited 2022-03-09].

Khoramishad, H,. Crocombe, K,. Kali, B. and Ashcroft, I. 2010. Predicting fatigue damage in adhesively bonded joints using cohesive zone model. In: International Journal of Fatigue. Volume 32. Pages 1146-1158. [Cited 2022-03-09].

Metsä Wood, 2021, Products/Kerto LVL. [Web page]. [Cited 2022-03-09]. Available: https://www.metsawood.com/global/Products/kerto/Pages/Kerto.aspx#

Natário, P., Silvestre, N. and Camotim, D. 2014. Web crippling failure using quasi-static FE models. In: Thin-Walled Structures. Volume 84. Pages 34-39. [Cited 2022-03-09].

Raute. 2021. Veneer Production. [Web page]. [Cited 2022-03-09]. Available: https://www.raute.com/industries/veneer-production/

Raute Lathe R7. 2021. veneer-lathe-R7-test-run2.jpg. [Online picture]. [Cited 2022-03-09]. Available: https://materials.raute.com/file/dl/c=system_x1080/PHDDog/YztfLZYMqp5t9Q_laR6K6A/veneer-lathe-R7-test-run2.jpg?t=1612264269

Raute Peeling line R5 picture. 2021. Peeling-Line-R5-left.jpg. [Picture]. [Cited 2022-03-09]. Available: https://materials.raute.com/file/dl/c=system_x1080/9IGmzA/cfV-MpwzEhAcVNXIgbO9cg/Peeling-Line-R5-left.jpg?t=1639137728

Raute R5 lathe picture. 2021. Veneer-Lathe-R5-front.jpg. [Picture]. [Cited 2022-03-09]. Available: https://materials.raute.com/file/dl/c=system_x1080/zQ1YqQ/ucb3N1iiH-xJ29eKrAQ8dQ/Veneer-Lathe-R5-front.jpg?t=1620113942

Raute Veneer Lathe R5. 2021. Veneer Lathe R5. [Web page]. [Cited 2022-03-09]. Available: https://www.raute.com/lines-and-machines/lines/veneer-peeling/veneer-peeling-line-r5/veneer-lathe-r5/

Raute Veneer Lathe R7-Hybrid. 2021. Veneer Lathe R7-Hybrid. [Web page]. [Cited 2022-04-07]. Available: https://www.raute.com/lines-and-machines/lines/veneer-peeling/veneer-peeling-line-r7-hybrid/veneer-lathe-r7-hybrid/

Raute Veneer peeling. 2021. Veneer peeling. [Web page]. [Cited 2022-03-09]. Available: https://www.raute.com/lines-and-machines/lines/veneer-peeling/

Raute Veneer Peeling Line R7-Hybrid. 2021. Veneer Peeling Line R7-Hybrid. [Web page]. [Cited 2022-07-04]. Available: https://www.raute.com/lines-and-machines/lines/veneer-peeling/veneer-peeling-line-r7-hybrid/

Reiterer, A., Sinn, G and Stanzl-Tschegg, S.E. 2002. Fracture characteristics of different wood species under mode I loading perpendicular to the grain. In: Materials Science and Engineering: A. Volume 332. Issue 1-2. Pages 29–36.

SALib API, 2021, Concise API Reference. [Web documentation]. [Cited 2022-03-09]. Available: https://salib.readthedocs.io/en/latest/api.html

SALib Basic. 2021. Basics. [Web page]. [Cited 2022-03-09]. Available: https://salib.readthedocs.io/en/latest/basics.html

SALib example result picture. 2021. Example_parabola.svg. [Online picture]. [Cited 2022-03-09]. Available: https://salib.readthedocs.io/en/latest/_images/example_parabola.svg

Smith, S.S and Hansen, R. 2001. From the Woods: Hardwood Veneer. PennState Extension. [Web Article]. [Cited 2022-03-09]. Available: https://extension.psu.edu/from-the-woods-hardwood-veneer

Volokh, K.Y. 2004. Comparison between cohesive zone models. In: Communications in numerical methods in engineering. Number 20. Pages 845-856. [Cited 2022-03-09].

Appendices

Appendix 1: Python script for model creation

```
def Veneer_Peeling_Script(r2, z1, peel_thickness, z2, xpos, ypos, RotationAngle, t, v, Dyn):

    if Dyn == 1:
        file_name = 'Dynamic_Peeling_r_'+str(r2)+'__t_'+str(peel_thickness)+'__beta_'+str(RotationAngle)+'__x_'+str(xpos)+'__v_'+str(v)
    else:
        file_name = 'Static_Peeling_r_'+str(r2)+'__t_'+str(peel_thickness)+'__beta_'+str(RotationAngle)+'__x_'+str(xpos)+'__v_'+str(v)

    file_name_ready = file_name.replace('.', '-')
    with open(file_name_ready + '.py','w') as f:

        print(

'''from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *

from datetime import datetime


Model = 'Model-1'
```

```
#---------------------------------------- Creating part ----------------------------------------

# x1 = 40
# y1 = -20
# z = 1000
# PartName = 'Cantiliver_block'


#Rectangular

# mdb.models[Model].ConstrainedSketch(name='_profile_', sheetSize=200.0)
# mdb.models[Model].sketches['_profile_'].rectangle(
#     point1=(x0, y0), point2=(x1, y1))
# mdb.models[Model].Part(dimensionality=THREE_D,
#                        name=PartName, type=DEFORMABLE_BODY)
# mdb.models[Model].parts[PartName].BaseSolidExtrude(
#     depth=z, sketch=mdb.models['Model-1'].sketches['_profile_'])
# del mdb.models[Model].sketches['_profile_']

#Circle

x0 = 0  #central coordinate of X
y0 = 0  #central coordinate of Y
z0 = 0  #central coordinate of Z
Cohesive_thickness = 0.0001
#r1 = 19     #inner radius

#Calculate Core radius for wood log'''
'\n'
'r2 = '+str(r2)+' #Define r2 for rest of the code\n'
'r1 = '+str(r2)+' - '+str(peel_thickness)+' #r2 is radius of the wood log\n'
'\n'
'z1 = '+str(z1)+' #Define z1 for code\n'
'z2 = '+str(z2)+' #Define z2 for code\n'
'\n'
'''WoodBlockName = 'Wood_Block'
```

```
#-------------------------------------------------- Wood log ----------------------------------------------------

mdb.models[Model].ConstrainedSketch(name='_profile_', sheetSize=200.0)
mdb.models[Model].sketches['_profile_'].ArcByCenterEnds(center=(x0, y0), direction=CLOCKWISE, point1=(x0, r1), point2=(r1, y0))
mdb.models[Model].sketches['_profile_'].Line(point1=(x0, r1), point2=(x0, r2))
mdb.models[Model].sketches['_profile_'].Line(point1=(r1, y0), point2=(r2, y0))
mdb.models[Model].sketches['_profile_'].ArcByCenterEnds(center=(x0, y0), direction=COUNTERCLOCKWISE, point1=(x0, r2), point2=(r2, y0))
mdb.models[Model].Part(dimensionality=THREE_D, name=WoodBlockName, type=DEFORMABLE_BODY)
mdb.models[Model].parts[WoodBlockName].BaseSolidExtrude(depth=z1, sketch=mdb.models[Model].sketches['_profile_'])
del mdb.models[Model].sketches['_profile_']

mdb.models[Model].ConstrainedSketch(gridSpacing=2.99, name='_profile_', sheetSize=119.93, transform=
    mdb.models[Model].parts[WoodBlockName].MakeSketchTransform(sketchPlane=mdb.models[Model].parts[WoodBlockName].faces[4], sketchPlaneSide=SIDE1,
    sketchUpEdge=mdb.models[Model].parts[WoodBlockName].edges[7], sketchOrientation=RIGHT, origin=(0, 0, 0)))
mdb.models[Model].parts[WoodBlockName].projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=mdb.models[Model].sketches['_profile_'])
mdb.models[Model].sketches['_profile_'].ArcByCenterEnds(center=(0, 0),
                                                        direction=CLOCKWISE, point1=(r1, 0), point2=(0, r1))
mdb.models[Model].parts[WoodBlockName].PartitionFaceBySketch(faces=mdb.models[Model].parts[WoodBlockName].faces.getSequenceFromMask(('[#10 ]', ), ),
                                                             sketch=mdb.models[Model].sketches['_profile_'],
                                                             sketchUpEdge=mdb.models[Model].parts[WoodBlockName].edges[7])
del mdb.models[Model].sketches['_profile_']

mdb.models[Model].parts[WoodBlockName].PartitionCellBySweepEdge(cells=mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask(('[#1 ]',
    ), ), edges=(mdb.models[Model].parts[WoodBlockName].edges[0], ), sweepPath=mdb.models[Model].parts[WoodBlockName].edges[4])

mdb.models[Model].ConstrainedSketch(gridSpacing=6.1, name='_profile_', sheetSize=244, transform=
                                    mdb.models[Model].parts[WoodBlockName].MakeSketchTransform(
                                        sketchPlane=mdb.models[Model].parts[WoodBlockName].faces[6], sketchPlaneSide=SIDE1,
                                        sketchUpEdge=mdb.models[Model].parts[WoodBlockName].edges[3],
                                        sketchOrientation=RIGHT, origin=(0.0, r1, z1)))
mdb.models[Model].parts[WoodBlockName].projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=mdb.models[Model].sketches['_profile_']) #cohesive layer
mdb.models[Model].sketches['_profile_'].Line(point1=(x0, -z1), point2=(Cohesive_thickness, -z1))
mdb.models[Model].sketches['_profile_'].Line(point1=(Cohesive_thickness, -z1), point2=(Cohesive_thickness, y0))
mdb.models[Model].sketches['_profile_'].Line(point1=(Cohesive_thickness, y0), point2=(x0, y0))
mdb.models[Model].sketches['_profile_'].Line(point1=(x0, y0), point2=(x0, -z1))
mdb.models[Model].parts[WoodBlockName].PartitionFaceBySketch(faces=mdb.models[Model].parts[WoodBlockName].faces.getSequenceFromMask((
    '[#40 ]', ), ), sketch=mdb.models[Model].sketches['_profile_'],
    sketchUpEdge=mdb.models[Model].parts[WoodBlockName].edges[3])

del mdb.models[Model].sketches['_profile_']

mdb.models[Model].parts[WoodBlockName].PartitionCellBySweepEdge(cells=mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask((
    '[#1 ]', ), ), edges=(mdb.models[Model].parts[WoodBlockName].edges[0],
                          mdb.models[Model].parts[WoodBlockName].edges[7],
                          mdb.models[Model].parts[WoodBlockName].edges[15],
                          mdb.models[Model].parts[WoodBlockName].edges[16]), sweepPath=
                          mdb.models[Model].parts[WoodBlockName].edges[6])
```

```
#-------------------------------------------- Blade ------------------------------------------------------------

BladeName = 'Blade'

y = 20  #startng point

difference = y - r1  #Cheking difference between r1 and y to set blade position

if difference >= 0:    #r1 is smaller than y. Need to lower blade starting point
    y = y - abs(difference)
    y1 = 22.91 - abs(difference)
    y2 =  24.79 - abs(difference)

else:                       #r1 is bigger than y. Need to rise blade starting point
    y = y + abs(difference)
    y1 = 22.91 + abs(difference)
    y2 =  24.79 + abs(difference)

x = 5.85
x1 = 13.84
x2 = 13.16

mdb.models[Model].ConstrainedSketch(name='_profile_', sheetSize=200.0)
mdb.models[Model].sketches['_profile_'].Line(point1=(x0, y), point2=(x, y))
mdb.models[Model].sketches['_profile_'].Line(point1=(x, y), point2=(x1, y1))
mdb.models[Model].sketches['_profile_'].Line(point1=(x1, y1), point2=(x2, y2))
mdb.models[Model].sketches['_profile_'].Line(point1=(x2, y2) , point2=(x0, y))
mdb.models[Model].Part(dimensionality=THREE_D,name=BladeName, type=DEFORMABLE_BODY)
mdb.models[Model].parts[BladeName].BaseSolidExtrude(depth=z2, sketch=mdb.models[Model].sketches['_profile_'])
del mdb.models[Model].sketches['_profile_']


#Lines

# PartName3 = 'Lines_python'
# x3 = 10
# x4 = 5
# y4 = 10

# z3 = 10

# mdb.models[Model].ConstrainedSketch(name='_profile_', sheetSize=200.0)
# mdb.models[Model].sketches['_profile_'].Line(point1=(x0, y0)
#     , point2=(x3, y0))
# mdb.models[Model].sketches['_profile_'].Line(point1=(x3, y0)
#     , point2=(x4, y4))
# mdb.models[Model].sketches['_profile_'].Line(point1=(x4, y4)
#     , point2=(x0, y0))
# mdb.models[Model].Part(dimensionality=THREE_D,
#                        name=PartName3, type=DEFORMABLE_BODY)
# mdb.models[Model].parts[PartName3].BaseSolidExtrude(
#     depth=z3, sketch=mdb.models['Model-1'].sketches['_profile_'])
# del mdb.models[Model].sketches['_profile_']
```

```
#-------------------------------------------- Reference point ---------------------------------------------

mdb.models[Model].parts[WoodBlockName].ReferencePoint(point=(x0, y0, z0))

# refpoints = mdb.models[Model].parts[WoodBlockName].referencePoints #to check order number of referance point

# print refpoints

#------------------------------------------------ Sets ------------------------------------------------------

PartSetName_Blade_whole = 'Whole_Blade'
PartSetName_Blade_Mesh_edges = 'Blade_Mesh_Controll_Edges'
PartSetName_Blade_Cutting_edge = 'Blade_Cutting_Edge'
PartSetName_WoodBlock_Peel_Layer = 'WoodBlock_Peel_Layer'
PartSetName_WoodBlock_inner = 'Inner_WoodBlock'
PartSetName_WoodBlock_cohesivezone = 'Cohesive_area'
PartSetName_WoodBlock_RP ='WoodBlock_RP'
PartSetName_WoodBlock_Core = 'WoodBlock_Core'

mdb.models[Model].parts[BladeName].Set(cells=mdb.models[Model].parts[BladeName].cells.getSequenceFromMask(('[#1]', ), ), name=PartSetName_Blade_whole)
mdb.models[Model].parts[BladeName].Set(edges=mdb.models[Model].parts[BladeName].edges.getSequenceFromMask(('[#c50 ]', ), ), name=PartSetName_Blade_Mesh_edges)
mdb.models[Model].parts[WoodBlockName].Set(cells=mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask(('[#2 ]', ), ), name=PartSetName_WoodBlock_Peel_Layer)
mdb.models[Model].parts[WoodBlockName].Set(cells=mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask(('[#1 ]', ), ), name=PartSetName_WoodBlock_cohesivezone)
mdb.models[Model].parts[WoodBlockName].Set(name=PartSetName_WoodBlock_RP, referencePoints=(mdb.models[Model].parts[WoodBlockName].referencePoints[6], ))
mdb.models[Model].parts[WoodBlockName].Set(cells=mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask(('[#4 ]', ), ), name=PartSetName_WoodBlock_Core)
mdb.models[Model].parts[BladeName].Set(edges=mdb.models[Model].parts[BladeName].edges.getSequenceFromMask(('[#100 ]', ), ), name=PartSetName_Blade_Cutting_edge)

# PartSetName_Woodblock_Face_inner = 'Face_Inner'
# PartSetName_Woodblock_Face_Outer = 'Face_Outer'

# mdb.models[Model].parts[WoodBlockName].Set(faces=mdb.models[Model].parts[WoodBlockName].faces.getSequenceFromMask(('[#1 ]', ), ), name=PartSetName_Woodblock_Face_inner)
# mdb.models[Model].parts[WoodBlockName].Set(faces=mdb.models[Model].parts[WoodBlockName].faces.getSequenceFromMask(('[#1 ]', ), ), name=PartSetName_Woodblock_Face_Outer)

# mdb.models[Model].parts[PartName].Set(faces=mdb.models[Model].parts[PartName].faces.getSequenceFromMask(('[#20]', ), ), name='BC_face')
# mdb.models[Model].parts[PartName2].Set(faces=mdb.models[Model].parts[PartName2].faces.getSequenceFromMask(('[#1]', ), ), name='Whole_thing_python')
# mdb.models[Model].parts[PartName2].Set(name='RP_python', referencePoints=(mdb.models[Model].parts[PartName2].referencePoints[2], ))
```

```
#---------------------------------------------------- Surfaces ----------------------------------------------------

WoodBlock_Peel_face = 'Outer_Peel_face'
WoodBlock_Inner_TopSurface = 'WoodBlock_Inner_TopSurface'
WoodBlock_CohesiveLayer_Bottom_Surface = 'WoodBlock_CohesiveLayer_Bottom_Surface'
WoodBlock_Peel_Bottom_Surface = 'WoodBlock_Peel_Bottom_Surface'
WoodBlock_Core_Top_Surface = 'WoodBlock_Core_Top_Surface'
WoodBlock_Peel_Bottom_Surface = 'WoodBlock_Peel_Bottom_Surface'
WoodBlock_Surface_at_RP = 'WoodBlock_RP_Surface'
WoodBlock_Surface_at_RP_1 = 'WoodBlock_RP_1_Surface'


Blade_Upper_Surface = 'Blade_Upper_Surface'
Blade_Lower_Surface = 'Blade_Lower_Surface'



mdb.models[Model].parts[WoodBlockName].Surface(name=WoodBlock_Peel_face, side1Faces=mdb.models[Model].parts[WoodBlockName]
                                        .faces.getSequenceFromMask(('[#10 ]', ), ))
mdb.models[Model].parts[BladeName].Surface(name=Blade_Upper_Surface, side1Faces=mdb.models[Model].parts[BladeName]
                                        .faces.getSequenceFromMask(('[#8 ]', ), ))
mdb.models[Model].parts[BladeName].Surface(name=Blade_Lower_Surface, side1Faces=mdb.models[Model].parts[BladeName]
                                        .faces.getSequenceFromMask(('[#4 ]', ), ))
mdb.models[Model].parts[WoodBlockName].Surface(name=WoodBlock_Core_Top_Surface, side2Faces=mdb.models[Model].parts[WoodBlockName]
                                            .faces.getSequenceFromMask(('[#20 ]', ), ))
mdb.models[Model].parts[WoodBlockName].Surface(name=WoodBlock_CohesiveLayer_Bottom_Surface, side1Faces=mdb.models[Model].parts[WoodBlockName]
                                        .faces.getSequenceFromMask(('[#20 ]', ), ))
mdb.models[Model].parts[WoodBlockName].Surface(name=WoodBlock_Peel_Bottom_Surface, side2Faces=mdb.models[Model].parts[WoodBlockName]
                                        .faces.getSequenceFromMask(('[#1 ]', ), ))
mdb.models[Model].parts[WoodBlockName].Surface(name=WoodBlock_Surface_at_RP, side1Faces=mdb.models[Model].parts[WoodBlockName]
                                        .faces.getSequenceFromMask(('[#40 ]', ), ))
mdb.models[Model].parts[WoodBlockName].Surface(name=WoodBlock_Surface_at_RP_1, side1Faces=mdb.models[Model].parts[WoodBlockName]
                                        .faces.getSequenceFromMask(('[#80 ]', ), ))
# mdb.models[Model].parts[WoodBlockName].Surface(name=WoodBlock_Inner_InnerSurface, side1Faces=mdb.models[Model].parts[WoodBlockName]
#     .faces.getSequenceFromMask(('[#23 ]', ), ))
```

```
#---------------------------------------------------- Orientations ----------------------------------------------------

mdb.models[Model].parts[WoodBlockName].MaterialOrientation(additionalRotationType=ROTATION_NONE, axis=AXIS_1, fieldName='',
                                                           localCsys=None, orientationType=SYSTEM, region=Region(
                                                           cells=mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask(
                                                               mask=('[#7 ]', ), )), stackDirection=STACK_3)


#---------------------------------------------- Material stuff ----------------------------------------------------

materialSteel = 'Steel'
Rho_Steel = 7.8
E_Steel = 2100000000000
nu_Steel = float(0.29)

mdb.models[Model].Material(name=materialSteel)
mdb.models[Model].materials[materialSteel].Density(table=((Rho_Steel, ),))
mdb.models[Model].materials[materialSteel].Elastic(table=((E_Steel, nu_Steel), ))

materialWood = 'Wood'
Rho_Wood = 0.55
D1111 = 150960000000
D1122 = 53238000000
D2222 = 72900000000
D1133 = 41087000000
D1212 = 36026000000
D1313 = 31150000000
D2233 = 50702000000
D2323 = 3294300000
D3333 = 43145000000

mdb.models[Model].Material(name=materialWood)
mdb.models[Model].materials[materialWood].Damping(structural=0.005)
mdb.models[Model].materials[materialWood].Density(table=((Rho_Wood, ),))
mdb.models[Model].materials[materialWood].Elastic(table=((D1111, D1122,
    D2222, D1133, D2233, D3333, D1212, D1313, D2323), ), type=ORTHOTROPIC)

materialCohesive = 'Cohesive'

mdb.models[Model].Material(name=materialCohesive)
mdb.models[Model].materials[materialCohesive].MaxsDamageInitiation(table=((95590000, 95590000, 95590000), ))
mdb.models[Model].materials[materialCohesive].maxsDamageInitiation.DamageEvolution(power=2,
                                                                                   table=((840000, ), ), type=ENERGY)
mdb.models[Model].materials[materialCohesive].maxsDamageInitiation.DamageStabilizationCohesive(cohesiveCoeff=0.002)
mdb.models[Model].materials[materialCohesive].Damping(structural=0.005)
mdb.models[Model].materials[materialCohesive].Density(table=((0.5, ),))
mdb.models[Model].materials[materialCohesive].Elastic(table=((13770000000, 19940000000, 9940000000),), type=TRACTION)
```

```
# ------------------------------------------ Material Sections ------------------------------------------------

sectionSteel = 'Steel_section'
sectionWood = 'Wood_section'
sectionCohesive = 'Cohesive_section'

mdb.models[Model].HomogeneousSolidSection(material = materialSteel, name = sectionSteel, thickness = None)
mdb.models[Model].HomogeneousSolidSection(material = materialWood, name = sectionWood, thickness = None)
mdb.models[Model].CohesiveSection(initialThickness = Cohesive_thickness, initialThicknessType=SPECIFY, material = materialCohesive, name = sectionCohesive,
                                  outOfPlaneThickness = None, response =TRACTION_SEPARATION)

#------------------------------------- Material assignment to parts -------------------------------------------

mdb.models[Model].parts[BladeName].SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE, region=
                    mdb.models[Model].parts[BladeName].sets[PartSetName_Blade_whole],sectionName=sectionSteel, thicknessAssignment=FROM_SECTION)
mdb.models[Model].parts[WoodBlockName].SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE, region=
                    mdb.models[Model].parts[WoodBlockName].sets[PartSetName_WoodBlock_Core],sectionName=sectionWood, thicknessAssignment=FROM_SECTION)
mdb.models[Model].parts[WoodBlockName].SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE, region=
                    mdb.models[Model].parts[WoodBlockName].sets[PartSetName_WoodBlock_Peel_Layer],sectionName=sectionWood, thicknessAssignment=FROM_SECTION)
mdb.models[Model].parts[WoodBlockName].SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE, region=
                    mdb.models[Model].parts[WoodBlockName].sets[PartSetName_WoodBlock_cohesivezone],sectionName=sectionCohesive, thicknessAssignment=FROM_SECTION)

# mdb.models[Model].parts[PartName].Set(cells = mdb.models[Model].parts[PartName].cells.getSequenceFromMask(('[#1]',),), name='Set-1')
# mdb.models[Model].parts[PartName].SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE, region=
#                     mdb.models[Model].parts[PartName].sets['Set-1'],sectionName='Steel_section', thicknessAssignment=FROM_SECTION)


#------------------------------------------------------ step ------------------------------------------------

Initial = 'Initial'
StepName = 'Peeling'
'''
'if '+str(Dyn)+' == 1:\n'
'    #Dynamic Implicit\n'
'    mdb.models[Model].ImplicitDynamicsStep(initialInc=0.001, minInc=1.5e-12, name=StepName, nlgeom=ON, previous='+"Initial"+', timePeriod= '+str(t)+')\n'

'else:'
'    #static\n'
'    mdb.models[Model].StaticStep(name=StepName, previous='+"Initial"+')\n'
'    mdb.models[Model].steps[StepName].setValues(nlgeom = ON, maxNumInc = 1000, initialInc = 0.001, maxInc = 0.001, minInc = 1e-12, timePeriod = '+str(t)+')\n'
'''
#Fild and History Outputs
F_out_name = 'Pyhton_Request'
# H_out_name = 'Python_Request_History'
F_out_name_CuttingEdge = 'FieldOut_Cutting_Edge'
H_out_name_CuttingEdge = 'HistoryOut_Cutting_Edge'

# mdb.models[Model].FieldOutputRequest['F-Output-1'].setValues(variables = ('S', 'U', 'RF', 'SDEG', 'STATUS'))
mdb.models[Model].FieldOutputRequest(createStepName=StepName, name=F_out_name, variables=('S', 'U', 'RF', 'SDEG', 'STATUS'))
# mdb.models[Model].HistoryOutputRequest(createStepName=StepName, name=H_out_name, variables=('ALLAE', 'ALLCD', 'ALLDMD','ALLEE',
#                                                                                             'ALLFD', 'ALLIE', 'ALLJD', 'ALLKE','ALLKL', 'ALLPD',
#                                                                                             'ALLQB', 'ALLSD', 'ALLSE' ,'ALLVD', 'ALLWK', 'ETOTAL'))
```

```
#---------------------------------------------------- Assembly ----------------------------------------------------

assemblyBlade = 'Blade_assembly'
assemblyWoodBlock = 'WoodBlock_assembly'


mdb.models[Model].rootAssembly.DatumCsysByDefault(CARTESIAN)
mdb.models[Model].rootAssembly.Instance(dependent=ON, name=assemblyWoodBlock, part=mdb.models[Model].parts[WoodBlockName])
mdb.models[Model].rootAssembly.Instance(dependent=ON, name=assemblyBlade, part=mdb.models[Model].parts[BladeName])

mdb.models[Model].rootAssembly.EdgeToEdge(fixedAxis=mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].edges[16]
    , flip=OFF, movableAxis=mdb.models[Model].rootAssembly.instances[assemblyBlade].edges[8])



#------------------------------------------------ translation for part ------------------------------------------

#z position coordinate for Blade
'''
'zpos = ('+str(z2)+' - '+str(z1)+') // 2\n'
'''
#rotation and translation for Blade
'''
'mdb.models[Model].rootAssembly.translate(instanceList=(assemblyBlade, ), vector=('+str(xpos)+', '+str(ypos)+', -zpos))\n'
'mdb.models[Model].rootAssembly.rotate(angle='+str(RotationAngle)+', axisDirection=(x0, y0, 1), axisPoint=(x0, y0, z0), instanceList=(assemblyBlade, ))\n'
'''
#------------------------------------------------ rotation for part ---------------------------------------------

# RotationAngle = 0

# mdb.models[Model].rootAssembly.rotate(angle=RotationAngle, axisDirection=(0.0, 0.0, 10.0), axisPoint=(0.0, 0.0, 0.0), instanceList=('Cantiliver_assembly', ))

#Field output
#mdb.models[Model].FieldOutputRequest(createStepName=StepName, name=F_out_name_CuttingEdge, rebar=EXCLUDE, region=
#mdb.models[Model].rootAssembly.allInstances[assemblyBlade].sets[PartSetName_Blade_Cutting_edge], sectionPoints=DEFAULT, variables=('RF', ))

#History output
mdb.models[Model].HistoryOutputRequest(createStepName=StepName, name=H_out_name_CuttingEdge, rebar=EXCLUDE, region=
mdb.models[Model].rootAssembly.allInstances[assemblyBlade].sets[PartSetName_Blade_Cutting_edge], sectionPoints=DEFAULT, variables=('RF1', 'RF2', 'RF3', ))
# mdb.models[Model].HistoryOutputRequest['H-Output-1'].suppress()
```

```
#---------------------------------------------------- interaction ------------------------------------------------------
FrictionProperty = 'Friction'
HardContactProperty = 'Hard_Contact'

mu = 0.4  #Friction coeficient

mdb.models[Model].ContactProperty(FrictionProperty)
mdb.models[Model].interactionProperties[FrictionProperty].TangentialBehavior(dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
                                                                             formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
                                                                             pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
                                                                             table=((mu, ), ), temperatureDependency=OFF)

mdb.models[Model].ContactProperty(HardContactProperty)
mdb.models[Model].interactionProperties[HardContactProperty].NormalBehavior(allowSeparation=ON, constraintEnforcementMethod=DEFAULT, pressureOverclosure=HARD)
mdb.models[Model].interactionProperties[HardContactProperty].TangentialBehavior(dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
                                                                             formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
                                                                             pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
                                                                             table=((mu, ), ), temperatureDependency=OFF)

General_Contact = 'General_Contact'

mdb.models[Model].ContactStd(createStepName='Initial', name=General_Contact)
mdb.models[Model].interactions[General_Contact].includedPairs.setValuesInStep(
    addPairs=(
        (mdb.models[Model].rootAssembly.instances[assemblyBlade].surfaces[Blade_Upper_Surface],
        mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].surfaces[WoodBlock_Peel_face]),
        (mdb.models[Model].rootAssembly.instances[assemblyBlade].surfaces[Blade_Upper_Surface],
        mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].surfaces[WoodBlock_CohesiveLayer_Bottom_Surface]),
        (mdb.models[Model].rootAssembly.instances[assemblyBlade].surfaces[Blade_Upper_Surface],
        mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].surfaces[WoodBlock_Peel_Bottom_Surface])), stepName='Initial', useAllstar=OFF)
mdb.models[Model].interactions[General_Contact].contactPropertyAssignments.appendInStep(
    assignments=((GLOBAL, SELF, HardContactProperty), ), stepName='Initial')
mdb.models[Model].interactions[General_Contact].contactPropertyAssignments.appendInStep(
    assignments=((
    mdb.models[Model].rootAssembly.instances[assemblyBlade].surfaces[Blade_Lower_Surface],
    mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].surfaces[WoodBlock_Core_Top_Surface],
    FrictionProperty), ), stepName='Initial')

# mdb.models[Model].SurfaceToSurfaceContactStd(adjustMethod=NONE,
#     clearanceRegion=None, createStepName='Peeling', datumAxis=None,
#     initialClearance=OMIT, interactionProperty= HardContactProperty, master=
#     mdb.models[Model].rootAssembly.instances[assemblyBlade].surfaces[Blade_Upper_Surface]
#     , name='Normal_dir_contact', slave=
#     mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].surfaces[WoodBlock_Peel_Bottom_Surface]
#     , sliding=FINITE, thickness=ON)


# Friction_Contact = 'Friction_Contact'

# mdb.models[Model].SurfaceToSurfaceContactStd(adjustMethod=NONE,
#     clearanceRegion=None, createStepName='Initial', datumAxis=None,
#     initialClearance=OMIT, interactionProperty=FrictionProperty, master=
#     mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].surfaces[WoodBlock_Inner_TopSurface]
#     , name=Friction_Contact, slave=
#     mdb.models[Model].rootAssembly.instances[assemblyBlade].surfaces[Blade_Lower_Surface]
#     , sliding=FINITE, thickness=ON)
```

```
#---------------------------------------------------- Constraints ------------------------------------------------------

Interaction_WoodBlock = 'REVJ1'
Interaction_WoodBlock2 = 'REVJ2'

PartSetName_Assembly_RP_1 = 'Assembly_WoodBlock_RP_1'

mdb.models[Model].rootAssembly.ReferencePoint(point=
    mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].InterestingPoint(
    mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].edges[18], CENTER))

mdb.models[Model].rootAssembly.Set(name=PartSetName_Assembly_RP_1, referencePoints=(
    mdb.models[Model].rootAssembly.referencePoints[7], ))

mdb.models[Model].rootAssembly.regenerate()

mdb.models[Model].Coupling(controlPoint=mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].sets[PartSetName_WoodBlock_RP]
    , couplingType=KINEMATIC, influenceRadius=WHOLE_SURFACE, localCsys=None, name=Interaction_WoodBlock2, surface=
mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].surfaces[WoodBlock_Surface_at_RP],
    u1=ON, u2=ON, u3=ON, ur1=ON, ur2=ON, ur3=ON)

mdb.models[Model].Coupling(controlPoint=mdb.models[Model].rootAssembly.sets[PartSetName_Assembly_RP_1]
    , couplingType=KINEMATIC, influenceRadius=WHOLE_SURFACE, localCsys=None, name=Interaction_WoodBlock, surface=
mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].surfaces[WoodBlock_Surface_at_RP_1],
    u1=ON, u2=ON, u3=ON, ur1=ON, ur2=ON, ur3=ON)

#mdb.models[Model].Coupling(controlPoint=mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].sets[PartSetName_WoodBlock_RP]
#     , couplingType=KINEMATIC, influenceRadius=WHOLE_SURFACE, localCsys=None, name=Interaction_WoodBlock, surface=
#mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].sets[PartSetName_WoodBlock_Core],
#     u1=ON, u2=ON, u3=ON, ur1=ON, ur2=ON, ur3=ON)
```

```
#------------------------------------------------ Mesh ------------------------------------------------------------------
MeshSize_Blade = 8    #original value 2
MeshSize_WoodBlock = 1

#Blade
mdb.models[Model].parts[BladeName].setElementType(elemTypes=(ElemType(elemCode=C3D8IH, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
                                                                      kinematicSplit=AVERAGE_STRAIN, hourglassControl=DEFAULT,
                                                                      distortionControl=DEFAULT), ElemType(elemCode=C3D6, elemLibrary=STANDARD),
                                                    ElemType(elemCode=C3D4, elemLibrary=STANDARD)), regions=(
                                                        mdb.models[Model].parts[BladeName].cells.getSequenceFromMask(('[#1]', ), ), ))

mdb.models[Model].parts[BladeName].seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=MeshSize_Blade)
mdb.models[Model].parts[BladeName].seedEdgeByNumber(constraint=FINER, edges=mdb.models[Model].parts[BladeName].edges.getSequenceFromMask(
                                                    ('[#10a ]', ), ), number=40) #Longitual edges
mdb.models[Model].parts[BladeName].seedEdgeByNumber(constraint=FINER, edges=mdb.models[Model].parts[BladeName].edges.getSequenceFromMask(
                                                    ('[#c50 ]', ), ), number=4) #Side edges of blade
mdb.models[Model].parts[BladeName].seedEdgeByNumber(constraint=FINER, edges=mdb.models[Model].parts[BladeName].edges.getSequenceFromMask(
                                                    ('[#280 ]', ), ), number=3)  #Blade edges of slope
mdb.models[Model].parts[BladeName].seedEdgeByNumber(constraint=FINER, edges=mdb.models[Model].parts[BladeName].edges.getSequenceFromMask(
                                                    ('[#5 ]', ), ), number=3)  #Blade edge of thickness
mdb.models[Model].parts[BladeName].generateMesh()

mdb.models[Model].parts[BladeName].assignStackDirection(cells=mdb.models[Model].parts[BladeName].cells.getSequenceFromMask(('[#1 ]', ), ),
                                                    referenceRegion=mdb.models[Model].parts[BladeName].faces[0])


#Wood block
mdb.models[Model].parts[WoodBlockName].setElementType(elemTypes=(ElemType(elemCode=C3D8IH, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
                                                                      kinematicSplit=AVERAGE_STRAIN, hourglassControl=DEFAULT,
                                                                      distortionControl=DEFAULT), ElemType(elemCode=C3D6, elemLibrary=STANDARD),
                                                    ElemType(elemCode=C3D4, elemLibrary=STANDARD)), regions=(
                                                        mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask(('[#4]', ), ), ))
mdb.models[Model].parts[WoodBlockName].setElementType(elemTypes=(ElemType(elemCode=C3D8IH, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
                                                                      distortionControl=DEFAULT), ElemType(elemCode=C3D6, elemLibrary=STANDARD),
                                                    ElemType(elemCode=C3D4, elemLibrary=STANDARD)), regions=(
                                                        mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask(('[#2]', ), ), ))
mdb.models[Model].parts[WoodBlockName].setElementType(elemTypes=(ElemType(elemCode=COH3D8, elemLibrary=STANDARD, elemDeletion=ON, maxDegradation=0.002),
                                                      ElemType(elemCode=COH3D6, elemLibrary=STANDARD), ElemType(elemCode=UNKNOWN_TET,
                                                         elemLibrary=STANDARD)), regions=(
                                                        mdb.models[Model].parts[WoodBlockName].cells.getSequenceFromMask(('[#1 ]', ), ), ))

mdb.models[Model].parts[WoodBlockName].seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=MeshSize_WoodBlock)
mdb.models[Model].parts[WoodBlockName].seedEdgeByNumber(constraint=FINER, edges=mdb.models[Model].parts[WoodBlockName].edges.getSequenceFromMask(
                                                    ('[#60000 ]', ), ), number=5)
```

```
mdb.models[Model].parts[WoodBlockName].generateMesh()

mdb.models[Model].parts[WoodBlockName].assignStackDirection(cells=mdb.models[Model].parts[WoodBlockName].
                                    cells.getSequenceFromMask(('[#1 ]', ), ), referenceRegion=mdb.models[Model].parts[WoodBlockName].faces[0])
mdb.models[Model].parts[WoodBlockName].assignStackDirection(cells=mdb.models[Model].parts[WoodBlockName].
                                    cells.getSequenceFromMask(('[#2 ]', ), ), referenceRegion=mdb.models[Model].parts[WoodBlockName].faces[4])

# mdb.models[Model].parts[BladeName].setElementType(elemTypes=(ElemType(elemCode=C3D8, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
#                                                                       kinematicSplit=AVERAGE_STRAIN, hourglassControl=DEFAULT,
#                                                                       distortionControl=DEFAULT), ElemType(elemCode=C3D6, elemLibrary=STANDARD),
#                                                     ElemType(elemCode=C3D4, elemLibrary=STANDARD)), regions=(
#                                                         mdb.models[Model].parts[BladeName].cells.getSequenceFromMask(('[#1]', ), ), ))
# mdb.models[Model].parts[BladeName].seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=MeshSize_Blade)
# mdb.models[Model].parts[BladeName].generateMesh()

#Amplitude

#mdb.models[Model].TabularAmplitude(data=((0.0, 0.0), (1.0, 1.0)), name='Amp-python', smooth=SOLVER_DEFAULT, timeSpan=STEP)

#--------------- Defining omega for rotation and rotation angle phi for log ----------------------------------------
'''
'omega = '+str(v)+' / '+str(r2)+' #r2 is radius from center to top of peel in cm\n'

'phi = omega * '+str(t)+' #phi is how much in radians log need to rotate in the simulation time to get wanted omega\n'
'''
```

```
#---------------------------------- BCs and Loads ------------------------------------------------------------

Blade_BC_name = 'Fixed_blade'
WoodBlock_BC_name = 'WoodBlock_Rotation'
WoodBlock_BC_revJ1_name = 'WoodBlock_BC_RevJ1'
WoodBlock_BC_revJ2_name = 'WoodBlock_BC_RevJ2'

mdb.models[Model].rootAssembly.regenerate()

mdb.models[Model].EncastreBC(createStepName='Initial', localCsys=None, name=Blade_BC_name, region=
                             mdb.models[Model].rootAssembly.instances[assemblyBlade].sets[PartSetName_Blade_whole])

mdb.models[Model].DisplacementBC(amplitude=UNSET, createStepName='Initial',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name=
    WoodBlock_BC_revJ2_name, region=
    mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].sets[PartSetName_WoodBlock_RP]
    , u1=SET, u2=SET, u3=SET, ur1=SET, ur2=SET, ur3=SET)
mdb.models[Model].boundaryConditions[WoodBlock_BC_revJ2_name].setValuesInStep(
    stepName=StepName, ur3=-phi)

mdb.models[Model].DisplacementBC(amplitude=UNSET, createStepName='Initial',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name=
    WoodBlock_BC_revJ1_name, region=
    mdb.models[Model].rootAssembly.sets[PartSetName_Assembly_RP_1]
    , u1=SET, u2=SET, u3=SET, ur1=SET, ur2=SET, ur3=SET)
mdb.models[Model].boundaryConditions[WoodBlock_BC_revJ1_name].setValuesInStep(
    stepName=StepName, ur3=-phi)

# mdb.models[Model].DisplacementBC(amplitude=UNSET, createStepName='Initial',
#     distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name=
#     WoodBlock_BC_name, region=
#     mdb.models[Model].rootAssembly.instances[assemblyWoodBlock].sets[PartSetName_WoodBlock_RP]
#     , u1=0.0, u2=0.0, u3=0.0, ur1=0.0, ur2=0.0, ur3=0.0)
# mdb.models[Model].boundaryConditions[WoodBlock_BC_name].setValuesInStep(
#     amplitude = FREED ,stepName=StepName, ur3=-phi)

# F = -100

# mdb.models[Model].rootAssembly.Set(name='Force_set', vertices=
#                                    mdb.models[Model].rootAssembly.instances[assemblyName].vertices.getSequenceFromMask(('[#50]',),))
# mdb.models[Model].ConcentratedForce(cf2=F, createStepName=StepName, distributionType=UNIFORM, field='', localCsys=None,
#                                     name='Vertical_Force', region=mdb.models[Model].rootAssembly.sets['Force_set'])
```

```
#---------------------------------------------------- Job ------------------------------------------------------------
'''
#Replacing point with - if varaible has point in value
'Variable_list = [str('+str(r2)+'), str('+str(peel_thickness)+'), str('+str(xpos)+'), str('+str(RotationAngle)+'), str('+str(v)+')]  #List of variables in the Job name'
'''
Variable_list_for_Job_Name = [sub.replace('.', '-') for sub in Variable_list] #New list where point is replace with - to privent crash due to naming
#Variable_list_for_Job_Name = Variable_list.replace('.', ',')
'''
'if '+str(Dyn)+' == 1:\n'
    '''    Job_Name = 'Dynamic_Peeling_r_'+str(Variable_list_for_Job_Name[0])+'__t_'+str(Variable_list_for_Job_Name[1])+'__beta_'+str(Variable_list_for_Job_Name[3])+'__x_'
    +str(Variable_list_for_Job_Name[2])+'__v_'+str(Variable_list_for_Job_Name[4]) +''
else:
    Job_Name = 'Static_Peeling_r_'+str(Variable_list_for_Job_Name[0])+'__t_'+str(Variable_list_for_Job_Name[1])+'__beta_'+str(Variable_list_for_Job_Name[3])+'__x_'
    +str(Variable_list_for_Job_Name[2])+'__v_'+str(Variable_list_for_Job_Name[4]) + ''

CPUs = 4
GPUs = 1

print('Analysis starts')

start = datetime.now()

mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF, explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
        memory=95, memoryUnits=PERCENTAGE, model=Model, modelPrint=OFF, multiprocessingMode=MPI, name=Job_Name + '', nodalOutputPrecision=SINGLE,
        numCpus=CPUs, numDomains=CPUs, numGPUs=GPUs, queue=None, resultsFormat=ODB, scratch='', type=ANALYSIS, userSubroutine='', waitHours=0, waitMinutes=0)
mdb.jobs[Job_Name].submit()
mdb.jobs[Job_Name].waitForCompletion()

end = datetime.now()

print('Analysis completed')

simulation_time = end - start
#return simulation_time, Job_Name, StepName''', file=f)
    return file_name_ready
```

Appendix 2: Python script for plotting

```python
from ReadFile import Read_File_and_create_Matrix

def Plotting_results(Parameter_values, parameter_name, max_and_avg):

    import plotly
    import plotly.graph_objs as go
    import numpy as np

    if max_and_avg == 1:


        Avg_Force_storage , Max_force_storage, file_name = Read_File_and_create_Matrix(Parameter_values, parameter_name, all_results = 0)


        parameter_data = Parameter_values

        fig1 = go.Scatter(x = parameter_data,
                          y = Max_force_storage,
                          name = 'Max Force')

        fig2 = go.Scatter(x = parameter_data,
                          y = Avg_Force_storage,
                          name = 'Avg Force')

        if parameter_name == 'tip' or parameter_name == 'Blade_pos':
            title_x = 'Blade Position from the center of the log in x - axis (mm)'
            tittle = 'Blade tip position'
        elif parameter_name == 'thk':
            title_x = 'Thickness (mm)'
            tittle = 'Thickness of the peel'
        elif parameter_name == 'Clearance':
            title_x = 'Clearance angle (deg)'
            tittle = 'Blade clearance angle'
        elif parameter_name == 'veL':
            title_x = 'Speed (m/s)'
            tittle = 'Peeling speed'



        mylayout = go.Layout(title = tittle,
                             xaxis_title = title_x,
                             yaxis_title = 'Cutting force (N)')

        file_name = file_name.replace('.txt', '_figure.html')

        plotly.offline.plot({'data': [fig1, fig2],
                             'layout': mylayout},
                             auto_open=True,
                             filename= file_name
                            )
```

```python
    else:

        figure = {}

        for i in range(len(Parameter_values)):

            Force, Time = Read_File_and_create_Matrix(Parameter_values[i], parameter_name, all_results=2)
            fig = go.Scatter(x = Time,
                             y = Force,
                             name = parameter_name + '_' + str(Parameter_values[i])
                             )

            figure.update({str(i) : fig})

        mylayout = go.Layout(title = parameter_name,
                             xaxis_title = 'Simulation time (s)',
                             yaxis_title = 'Cutting force (N)')
        file_name = parameter_name + 'simulation.html'

        plotly.offline.plot({'data' : [figure['0'], figure['1'], figure['2'], figure['3'], figure['4'],
                                       figure['5'], figure['6'], figure['7'], figure['8']],
                             'layout' : mylayout},
                            auto_open = True,
                            filename = file_name
                            )
```

Appendix 3: Python script for Readfile function

```
import numpy as np


def Read_File_and_create_Matrix(Parameter_values, Parameter_name , all_results):


    Numpy_matrix_avg_force = np.zeros([0])
    Numpy_matrix_max_force = np.zeros([0])

    if all_results == 1:  #Data read and hangling for SA analysis

        for name in range(len(Parameter_name)):

            Parameter_name_loop = Parameter_name[name]

            for value in range(len(Parameter_values[Parameter_name_loop])):

                temp = Parameter_values[Parameter_name_loop]
                temp_value = str(temp[value])
                temp_value = temp_value.replace('.', '')
                file_name = 'RF1_' + Parameter_name_loop + temp_value + '.txt'

                result_file = open(file_name,'r')
                results_temp = result_file.read().splitlines()
                result_file.close()

                Force_storage = []
                time_storage = []
                Force2 = np.zeros(0)
                Force_max = np.zeros(0)

                for n in range(len(results_temp)):
                    split = results_temp[n].split('\t')
                    time_storage.append(float(split[0]))
                    Force_storage.append(-1 * float(split[-1]))

                for time in range(len(time_storage)):

                   if time_storage[time] >= 0.004 and time_storage[time] <= 0.005:
                       start_point = time_storage.index(time_storage[time])
                       break

                for time2 in range(len(time_storage)):
                    if time_storage[time2] >= 0.01 and time_storage[time2] <= 0.011:
                        end_point = time_storage.index(time_storage[time2])
                        break

                for time3 in range(len(time_storage)):
                    if time_storage[time3] >= 0.0035 and time_storage[time3] < 0.004:
                        cut_off_point = time_storage.index(time_storage[time3])
                        break
```

```
            try:

                for k in range(start_point, end_point):
                    Force2 = np.append(Force2, Force_storage[k])

                for k2 in range(0, cut_off_point):
                    Force_max = np.append(Force_max, Force_storage[k2])

            except UnboundLocalError:
                end_point = 0
                start_point = 0

            for k in range(start_point, end_point):
                Force2 = np.append(Force2, Force_storage[k])

            for k2 in range(0, cut_off_point):
                Force_max = np.append(Force_max, Force_storage[k2])

            # Numpy_Force = np.array(Force_storage)
            Numpy_Force = np.multiply(Force_max, 1E-5)
            Force2 = np.multiply(Force2, 1E-5)

            Force_sum = np.sum(Force2)
            Divider = len(Force2)

            Avg_Force = Force_sum / Divider
            Numpy_matrix_avg_force = np.append(Numpy_matrix_avg_force, Avg_Force)


            Max_Force = np.max(Numpy_Force)


            Numpy_matrix_max_force = np.append(Numpy_matrix_max_force, Max_Force)


    return Numpy_matrix_avg_force, Numpy_matrix_max_force
```

```
elif all_results == 2:  #For data read and hanling for plots over simulation time

    temp_value = str(Parameter_values)
    temp_value = temp_value.replace('.', '')
    file_name = 'RF1_' + Parameter_name + temp_value + '.txt'

    result_file = open(file_name,'r')
    results_temp = result_file.read().splitlines()
    result_file.close()

    Force_storage = []
    time_storage = []

    for n in range(len(results_temp)):
        split = results_temp[n].split('\t')
        time_storage.append(float(split[0]))
        Force_storage.append(-1 * float(split[-1]))


    Numpy_Force = np.array(Force_storage)
    Numpy_Force = np.multiply(Numpy_Force, 1E-5)
    Numpy_Time = np.array(time_storage)

    return Numpy_Force, Numpy_Time
```

```
    else:  #Data read and handling for ploting max and avg forces

        for value in range(len(Parameter_values)):

            temp_value = str(Parameter_values[value])
            temp_value = temp_value.replace('.', '')
            file_name = 'RF1_' + Parameter_name + temp_value + '.txt'

            result_file = open(file_name,'r')
            results_temp = result_file.read().splitlines()
            result_file.close()

            Force_storage = []
            time_storage = []
            Force2 = np.zeros(0)
            Force_max = np.zeros(0)

            for n in range(len(results_temp)):
                split = results_temp[n].split('\t')
                time_storage.append(float(split[0]))
                Force_storage.append(-1 * float(split[-1]))


            for time in range(len(time_storage)):
                if time_storage[time] >= 0.004 and time_storage[time] <= 0.005:
                    start_point = time_storage.index(time_storage[time])
                    break

            for time2 in range(len(time_storage)):
                if time_storage[time2] >= 0.01 and time_storage[time2] <= 0.011:
                    end_point = time_storage.index(time_storage[time2])
                    break

            for time3 in range(len(time_storage)):
                if time_storage[time3] >= 0.0034 and time_storage[time3] < 0.004:
                    cut_off_point = time_storage.index(time_storage[time3])
                    break
```

```
            try:

                for k in range(start_point, end_point):
                    Force2 = np.append(Force2, Force_storage[k])

                for k2 in range(0, cut_off_point):
                    Force_max = np.append(Force_max, Force_storage[k2])

            except UnboundLocalError:
                end_point = 0
                start_point = 0

                for k in range(start_point, end_point):
                    Force2 = np.append(Force2, Force_storage[k])

                for k2 in range(0, cut_off_point):
                    Force_max = np.append(Force_max, Force_storage[k2])


            # Numpy_Force = np.array(Force_max_range)
            Numpy_Force = np.multiply(Force_max, 1E-5)
            Force2 = np.multiply(Force2, 1E-5)

            Force_sum = np.sum(Force2)
            Divider = len(Force2)

            Avg_Force = Force_sum / Divider
            Numpy_matrix_avg_force = np.append(Numpy_matrix_avg_force, Avg_Force)

            Max_Force = np.max(Numpy_Force)

            Numpy_matrix_max_force = np.append(Numpy_matrix_max_force, Max_Force)


        return Numpy_matrix_avg_force, Numpy_matrix_max_force, file_name
```

Appendix 4: Python script for Sensitivity analysis

```
from SALib.analyze import rbd_fast
import numpy as np
from ReadFile import Read_File_and_create_Matrix

def SA_analysis_function(problem_values, parameters):


    v_default = 2
    beta_default = 90
    x_default = 0
    t_default = 3.2

    parameter_names = problem_values['names']

    t_values = parameters['thk']
    v_values = parameters['veL']
    beta_values = parameters['Clearance']
    x_values = parameters['tip']

    numpy_avg_storage = np.zeros(0)
    numpy_max_storage = np.zeros(0)

    numpy_parameters = np.matrix([0,0,0,0])

    #Define problem variable for SA analysis

    problem = {
        'num_vars' : problem_values['number'],
        'names' : problem_values['names'],
        'bounds' : problem_values['bounds']
        }
```

```python
#------------------------------------- Read data from result files ---------------------------------------------------

    numpy_avg_storage , numpy_max_storage = Read_File_and_create_Matrix(parameters, parameter_names, all_results=1)


    #-------------------------------------------------------------------------------------------------------------------
    # [Avg, Max] = Read_File_and_create_Matrix(t_values, parameter_name, force_component, velocity = 2)

    # numpy_avg_storage = np.append(numpy_avg_storage, Avg)
    # numpy_max_storage = np.append(numpy_max_storage, Max)

    #---------------------------------------------- Creating input variable matrix which conteins input data ---------------------------------

    for name in range(len(parameter_names)):

        if parameter_names[name] == 'thk':
            for value in range(len(t_values)):
                numpy_temp = np.matrix([v_default, beta_default, x_default, t_values[value]])
                numpy_parameters = np.append(numpy_parameters, numpy_temp, axis=0)

        elif parameter_names[name] == 'veL':
            for value in range(len(v_values)):
                numpy_temp = np.matrix([v_values[value], beta_default, x_default, t_default])
                numpy_parameters = np.append(numpy_parameters, numpy_temp, axis=0)

        elif parameter_names[name] == 'rake':
            for value in range(len(beta_values)):
                numpy_temp = np.matrix([v_default, beta_values[value], x_default, t_default])
                numpy_parameters = np.append(numpy_parameters, numpy_temp, axis=0)

        elif parameter_names[name] == 'tip':
            for value in range(len(x_values)):
                numpy_temp = np.matrix([v_default, beta_default, x_values[value], t_default])
                numpy_parameters = np.append(numpy_parameters, numpy_temp, axis=0)

    numpy_parameters = np.delete(numpy_parameters, 0,0)

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

    # return numpy_parameters, problem


    #Sensitivity analysis

    Si_avg = rbd_fast.analyze(problem, numpy_parameters, numpy_avg_storage)
    Si_max = rbd_fast.analyze(problem, numpy_parameters, numpy_max_storage)

    return Si_avg, Si_max
```

Appendix 5: Python script for main function

```
from Plotting_results import Plotting_results
from SA_function import SA_analysis_function

#-----------------------------------------------Analysis steps---------------------

Veneer_speed = [2, 5] #m/s
Blade_angle = [0, 27.5] #deg
Blade_tip = [0, 0.2] #mm
thickness = [0.5, 4.0] #mm

number_of_steps = 8

Veneer_speed_range = Veneer_speed[1] - Veneer_speed[0]
Blade_angle_range = Blade_angle[1] - Blade_angle[0]
Blade_tip_range = Blade_tip[1] - Blade_tip[0]
thickness_range = thickness[1] - thickness[0]

Veneer_speed_step = Veneer_speed_range / number_of_steps
Blade_angle_step = Blade_angle_range / number_of_steps
Blade_tip_step = Blade_tip_range / number_of_steps
thickness_step = thickness_range / number_of_steps

Veneer_speed_storage = [Veneer_speed[0]]
Blade_angle_storage = [Blade_angle[0]]
Blade_tip_storage = [Blade_tip[0]]
thickness_storage = [thickness[0]]



for i in range(number_of_steps):
    v_temp = Veneer_speed_storage[i] + Veneer_speed_step
    Veneer_speed_storage.append(v_temp)

for i in range(number_of_steps):
    beta_temp = Blade_angle_storage[i] + Blade_angle_step
    Blade_angle_storage.append(beta_temp)

for i in range(number_of_steps):
    tip_temp = Blade_tip_storage[i] + Blade_tip_step
    Blade_tip_storage.append(round(tip_temp,3))

for i in range(number_of_steps):
    t_temp = thickness_storage[i] + thickness_step
    thickness_storage.append(round(t_temp,3))
```

```python
#---------------------------------------------- SA analysis and prep for it ----------------------------------------
problem_values = {
    'number' : 1,
    'names' : [ 'veL' ,'Clearance', 'tip', 'thk'  ],
    'bounds' : [[Veneer_speed[0], Veneer_speed[1]],
                [Blade_angle[0], Blade_angle[1]],
                [Blade_tip[0], Blade_tip[1]],
                [thickness[0], thickness[1]]
                ]
                }

parameter_values = {
    'veL' : Veneer_speed_storage,
    'Clearance' : Blade_angle_storage,
    'tip' : Blade_tip_storage,
    'thk' : thickness_storage
    }

Si_avg ,Si_max = SA_analysis_function(problem_values, parameter_values)

print('SA from Average force: ', Si_avg['S1'])
print('SA from max force: ', Si_max['S1'])

for name in range(len(problem_values['names'])):
    names = problem_values['names']
    temp_name = names[name]
    Plotting_results(parameter_values[temp_name], temp_name, max_and_avg = 1)



#-------------------------------------------------------------------------------
```