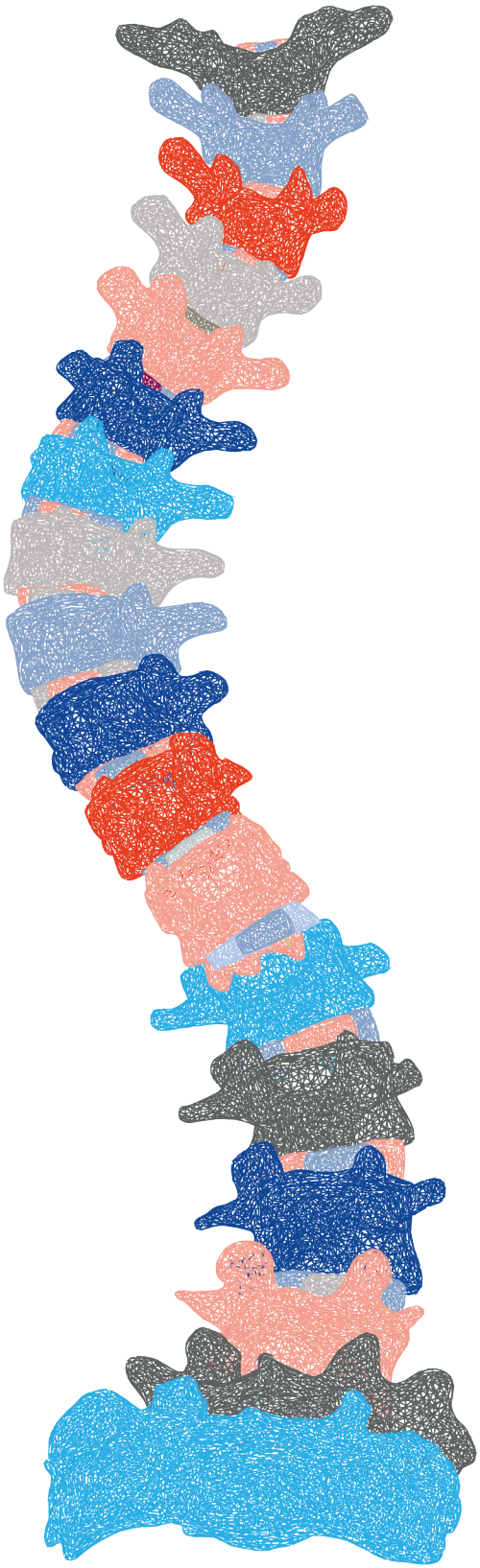
Rod Bending Accuracy Improvement for Spinal Fusion in Adolescent Idiopathic Scoliosis Patients

An exploratory Finite Element Analysis



December, 2022

Jette Smedema

TU Delft

UMC Utrecht

Rod Bending Accuracy Improvement for Spinal Fusion in Adolescent Idiopathic Scoliosis Patients

An exploratory Finite Element Analysis

By
J.A. Smedema

In partial fulfillment of the requirements for the degree of

Master of Science
In Biomedical Engineering

Delft University of Technology
To be defended on December 16th 2022

Supervisor: Dr. Ir. H.H. Weinans
Daily supervisors: C. Ngyuen, MSc
J. Magré, MSc
Thesis committee: Dr. Ir. N. Tumer
H. Pahlavani, MSc

Preface

After finishing my internship at the gait lab in AMC Amsterdam and working with cerebral palsy patients, there was one other topic I wanted to experience; using finite element analysis to find a solution for a medical issue. I would like to thank Harrie Weinans to give me the opportunity to provide me with such an assignment and for supervising me. The focus of this assignment lied on adolescent idiopathic scoliosis patients that require surgery to reduce the spinal deformation. With only having a mild type of scoliosis myself, I can only imagine the importance of spinal surgery procedures. Thank you, Tom Schlösser for letting me attend a spinal fusion surgery and explaining the entire procedure. You were very patient and took the time to answer all my questions. I am also grateful for Joëll Magré and Chien Nguyen, my daily supervisors. You both know I regularly experienced difficulties throughout my graduation. Thank you for advising me on my project and leading me towards the right direction.

Words can't describe how thankful I am for Joël van der Weijde, who was also working on his graduation project. Thank you for helping me understand the basics of scripting and for our regular meetups. Your positive mindset and friendship helped me to find my way through my graduation. I wish you all the best.

Last but not least, I would like to thank my family and friends for their loving support. Graduation was a big challenge for me. I have grown and am convinced that the skillset that I've developed will be of great use in the future.

Abstract

Background
Adolescent idiopathic scoliosis (AIS) with a Cobb angle larger than 45° is a spinal deformity that needs surgical treatment to stop progression and reduce pain. Spinal fusion is a surgical procedure where rods are attached to the spine through screws to diminish the Cobb angle.

Objective
The metal rods are pre-bent to correct the spinal deformity as much as possible. This rod shape is currently based on experience of the orthopedic surgeon. The aim of this thesis is to use Finite element analysis (FEA) to predict a suitable rod shape for AIS patients that need to undergo surgery.

Method
An FE model of an AIS spine was created based on high quality CT images. Rotation of vertebrae led to the optimal shape of the spine in the coronal plane. Vertebral bodies in this model were one by one rotated to a new position from top to bottom, to bring the spine to a new state and align it as much as possible to a normal physiological shape. However, alignment of the spine is an iterative process and rotation of every subsequent vertebra affects the shape of the entire spine. For each vertebral rotation, the reaction moments to maintain the new position is determined in the FE spine model. The acquired reaction moments are transferred to an FE rod model to provide the pre-bending of a rod that can bring the spine in its new state.

Results
The FE spine model was corrected to reduce the Cobb angle as much as possible. The highest stresses were found at the spinal apex. The reaction moments were transferred to a rod that became deformed in such a manner that straightened the spine model and corrected the Cobb angle to its pre-determined position with 33% reduction.

Discussion
This is an exploratory study that examines the possibility to predict ideal rod shapes pre-operatively to reduce surgical duration and improve the quality of the procedure. An accessory rod shape was accomplished but could not be validated due to lack of comparable studies. Improved FE spine models will lead to higher reliability of FE pre-bent rod models. This can be accomplished by, for example, inclusion of ligaments, muscles and other structures in the FE model and use of time-dependent material properties. The current model did not lead to a perfect aligned physiological spine and optimization algorithms should be developed in future models to further improve pre-bending of the rods.

Conclusion
Pre-operative prediction can improve the quality and reduce the time of spinal fusion. Improvements to the FE spine- and rod model, such as automation, generalization, inclusion of more structures, and optimization algorithms will lead to optimal pre-bending of rods and better and quicker surgical outcome.

Table of Contents

PREFACE **4**

ABSTRACT **5**

TABLE OF CONTENTS **6**

1. GENERAL INTRODUCTION **9**

1.1. Epidemiology 9
1.2. Curvature angle measurement 10
1.3. Morphology 10
1.4. Spinal Fusion 12
1.5. Problem Statement 13
1.6. Research objective 15

2. METHOD AND MATERIALS **16**

2.1. Segmentation 18
2.2. Pre-processing 18
2.3. Geometry of the discs 19
2.4. Mesh 21
2.5. Material properties 22
2.6. Constraints 23
2.7. Boundary conditions 24
2.8. Screw-rod fixation points 25
2.9. Displacement controlled analysis 30
2.10. Rod model 33
2.11. Spine model comparison 38
2.12. Automation 41

3. RESULTS **42**

3.1. The spine model 43
3.2. Curve reduction 45
3.3. Vertebral displacement 46
3.4. Compression and tension in the IVD's 47
3.5. Stress distribution in the IVD's 49
3.6. Vertebral reaction moments 50
3.7. Rod 52
3.8. Spine model comparison 54

4. DISCUSSION **57**

4.1. Spine model 57
4.2. Correction 59
4.3. Output parameters 60
4.4. Rod model 62
4.5. Comparison study 63
4.6. Assumptions and limitations 64
4.7. Future outlook and potential use 64

5. CONCLUSION **66**

REFERENCES **67**

APPENDICES **71**

Appendix A: Pre-processing 3Matic 71
Appendix B: Part names 72
Appendix C: Material study 73
Appendix D: Properties of the IVD's mesh 76

APPENDIX E: PROPERTIES OF THE VERTEBRAE MESH 77
APPENDIX F: STEP SPECIFICATIONS OF THE ANALYSIS 78
APPENDIX G: WORKFLOWS OF CODE 79
APPENDIX H: CALCULATE ROM IN MATLAB 81
APPENDIX I: PYTHON CODES 84

Acronyms

AF	Annulus Fibrosus
AIS	Adolescent Idiopathic Scoliosis
CAD	Computer Aided Design
Cobb angle	Angle between most superior- and inferior tilted vertebra
CT	Computed Tomography
DICOM	Digital Imaging and Communications in Medicine
DOF	Degree of Freedom
FEA	Finite Element Analysis
FEM	Finite Element Methods
IVD	Intervertebral disc
NP	Nucleus Pulposus
Region	Area of the spine, either cervical, thoracal, lumbar or sacral
RP	Reference point
Segment	An IVD with adjacent vertebrae (e.g., IVD L2L3 + vertebrae L2 and L3)

1. General Introduction

1.1. Epidemiology

Scoliosis is a condition in which the human spine has a deformed lateral curvature of at least ten degrees [1], [2]. Typically, the spine has curves along the sagittal plane at the cervical, thoracic, and lumbar regions. However, in scoliotic spines the curvatures also occur along the coronal plane, giving the spine a more complex three-dimensional shape (Figure 1.1 (A)).

The most common type of scoliosis in children is called adolescent idiopathic scoliosis (AIS), which affects between 0.4-5.2% of the worldwide population of whom approximately 0.1-0.3% requires operative treatment [3]. If the origin of the scoliosis is not identifiable, the condition is described as idiopathic. The term adolescent indicates patients over the age of ten when skeletal maturity occurs [1] and accounts for approximately 90% of idiopathic scoliosis in youth [2]. Progression of AIS is more common in women than in men [4], [5]. During the growth spurt, the spinal curvature tends to progress without the appropriate treatment. Although the underlying cause of AIS is still unknown, its progression is assumed to be the result of a biomechanical process. The Hueter-Volkmann principle explains that the imbalanced scoliotic adolescent spine endures unequally divided forces on the shifted vertebrae. Compression forces on the concave side of the spine slow down the development of growth plates, where tension forces on the convex side of the spine lead to more rapid enlargement of the growth plates. This causes a feed-forward mechanism that leads to an increasing structural deformity in the vertebral column [5].

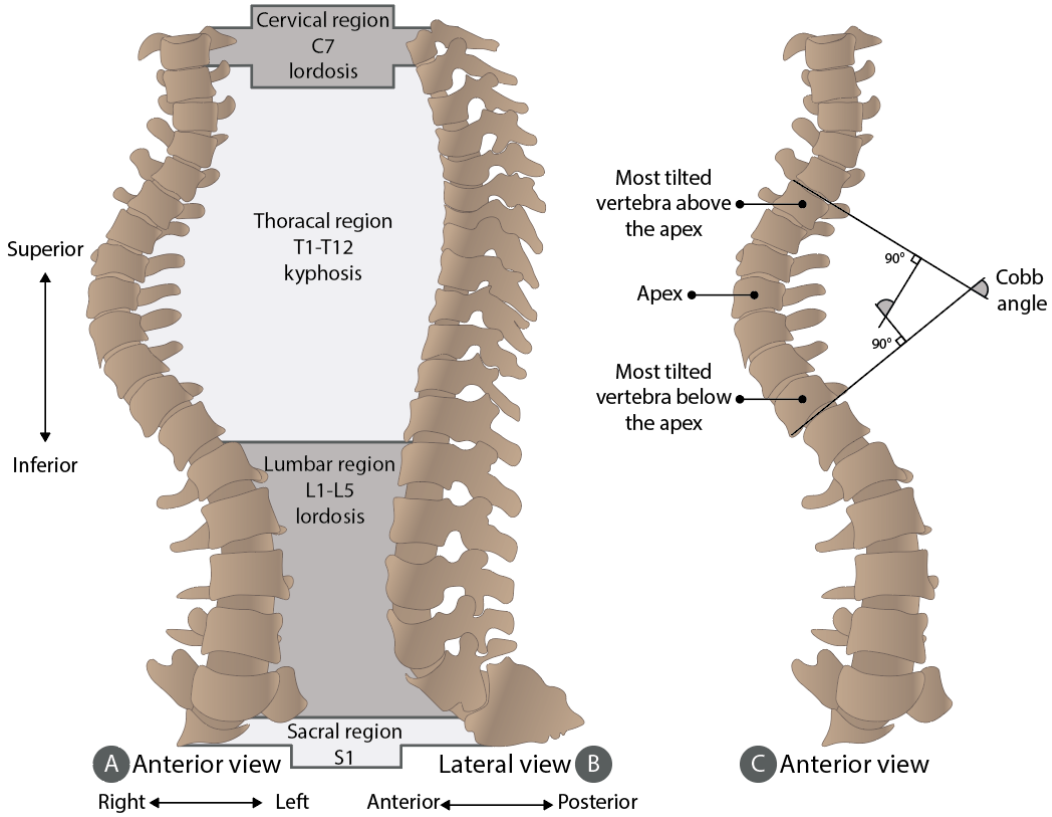


Figure 1.1 (A) Coronal plane of the spine, (B) Sagittal plane of the spine (adjusted) [5], (C) Measurement of the Cobb angle (adjusted) [6]. This image is based on a CT scan that does not include the cervical vertebrae C1-C6.

1.2. Curvature angle measurement

The Cobb's angle is a widely used indicator to quantify the severity of scoliosis. It is measured by determining the differences between the superior and inferior tilted vertebrae from standing coronal radiographs [1][4][6]. Figure 1.1 (C) shows the measurement of the Cobb angle with its corresponding tilted vertebrae. Lines are drawn parallel to the endplates of the most tilted vertebrae. These two lines intersect in the Cobb angle. The apex of a curve is at the farthest lateral level vertebra, or, -if two non-level vertebrae make up the apex, - it is considered the farthest lateral disc of that motion segment. In case of a Cobb angle over 45°, surgical intervention is often recommended to stop curve progression and reduce spinal deformity [1] [7].

1.3. Morphology

The spine functions as primary support of the body and generally consists of seven lordotic cervical- (C1-C7), twelve kyphotic thoracic- (T1-T12), and five lordotic lumbar (L1-L5) vertebrae, from cranial to caudal. Inferior to the L5 vertebra lies the fused kyphotic sacrum (S1) that ends in the coccyx [8].

Vertebrae

The vertebrae of the cervical-, thoracic- and lumbar regions show differences in morphology, based on their function. Cervical vertebrae are relatively small and lightweight to increase the amount of motion of the neck, thoracic vertebrae contain facets that articulate with the ribs, and lumbar vertebrae are fairly larger to bear the bodyweight [9]. The vertebral body is the anterior site of each vertebra where the intervertebral discs are positioned. The vertebral arch is formed by the bilateral pedicles and forms the foramen through which the spinal cord is led. Each vertebra has processes, see *Figure 1.2C* and *D*, that function as attachment sites for muscles and ligaments.

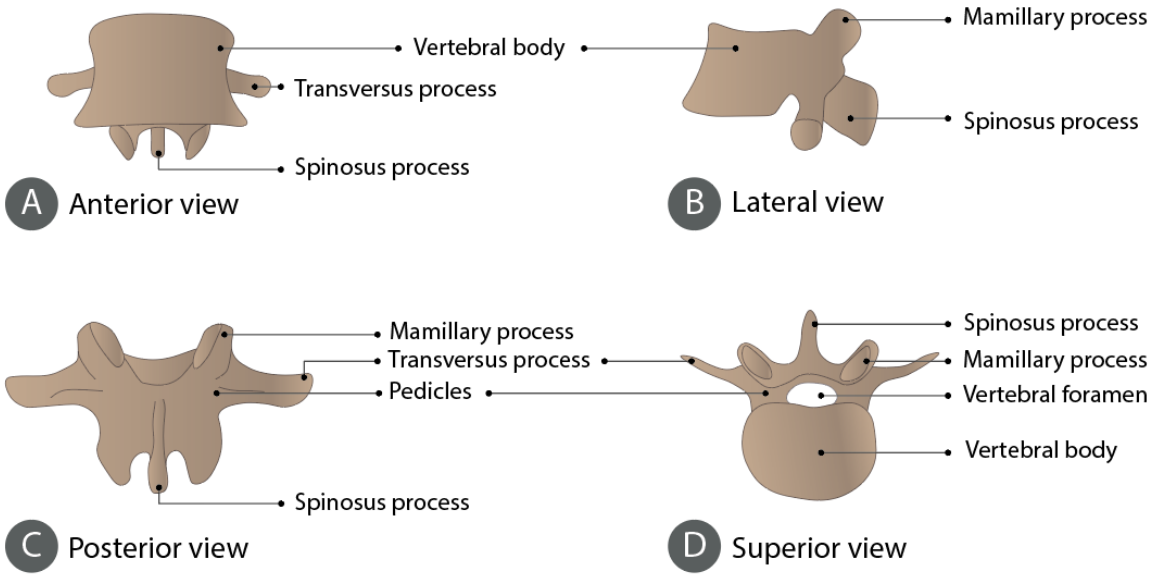


Figure 1.2: Morphology of vertebrae

Intervertebral discs

Vertebrae are separated by adjacently located intervertebral discs (IVD). IVD's act as shock-absorbers when impact increases the compression on the vertebrae. They also provide flexibility to the otherwise stiff spinal column [19]. The IVD-vertebra thickness ratio found in cervical- and lumbar region is higher compared to the thoracal region, resulting in a higher range of motion in these areas [10]. IVD's generally consist of three main parts; the nucleus pulposus (NP) which is the center of the disc, enclosed by the annulus fibrosus (AF) and two vertebral cartilaginous endplates as visualized in *Figure 1.3*A and B. The NP is composed of a gel-like structure. The AF is build-out of multiple fibrocartilage layers, - or laminae, - that consists of collagen type I and collagen type II. This annulus matrix is stiffer than the nucleus and can withstand compressive forces. The soft nucleus acts as a shock damper and helps to absorb the impact exposed by the body. It keeps adjacent vertebrae separated during regular body movements. IVD's are superiorly and inferiorly encased by cartilaginous endplates that assemble the NP and AF with the adjacent vertebral bodies [11].

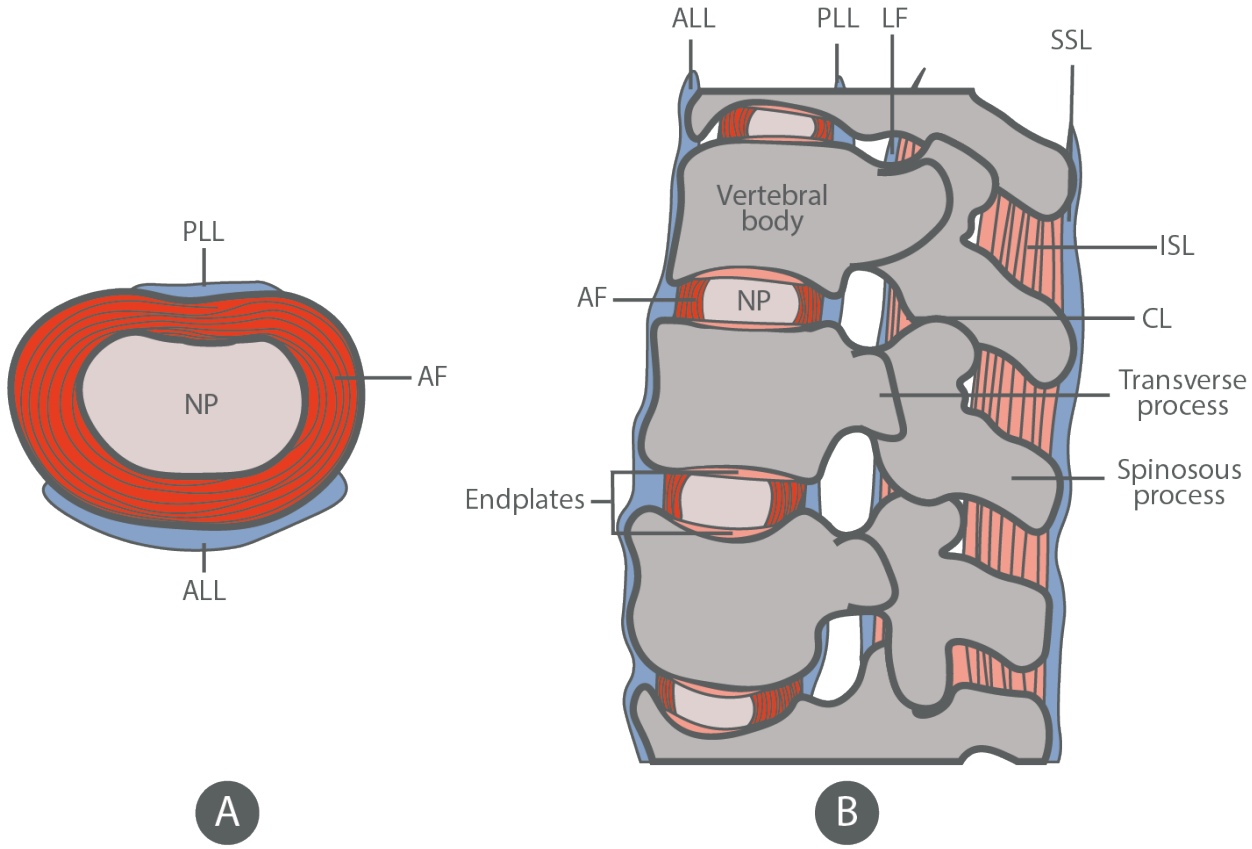


Figure 1.3: Intersection of intervertebral disc (A) based on [10], Lateral view of segments with accessory ligaments.

NP = Nucleus pulposus, AF = Annulus fibrosus, ALL = Anterior longitudinal ligament, PLL = Posterior longitudinal ligament, LF = Ligamentum flavum, SSL = Supraspinous ligament, ISL = Interspinous ligament, CL = Capsular ligament

1.4. Spinal Fusion

Spinal fusion is a surgical procedure used to treat adolescent idiopathic scoliosis (AIS). The goal is to stop the progression of the curvature, by decreasing the Cobb angle, correcting the deformity, and maintaining the corrections with rigid fixation [12]. Preoperative, clinical radiographs are used to determine the shape of the spine and to decide the type of surgical intervention. At the UMC Utrecht hospital, spinal fusion is done via a posterior incision, where the patient lies in the prone position. The orthopedic surgeon makes an incision in the patient's back along the spine, at the level of the proposed fixation. Connective tissue and muscles are placed aside to reach the vertebrae. In case the spine is still quite stiff, the ligamenta flava can be excised [13], see *Figure 1.3*B.

Subsequently, holes are drilled in the vertebrae, where the metal pedicle screws are placed through the pedicle and the vertebral body up to the cortex to anchor the rods to the spine (*Figure 1.4*A). These screws can either push or pull the rods placed atop, on both sides of the spine. The first rod is attached to the screws to correct the overall shape of the spine (*Figure 1.4B* and *Figure 1.4C*). The second rod reinforces rigid fixation and corrects the rotation per vertebra (*Figure 1.4D*). As the spinous processi lost its muscle attachment function, this part of the bone is then removed and used as an autograft. The bone graft is postero-bilateral placed on top of the posterior side of the vertebral column and will fuse the vertebrae throughout time [14]. Stiffness of the rods is important not only for initial correction but also to maintain the correction until the spine is fused. Thereafter, the loading via the rods will be reduced as the fused vertebral column will take over.

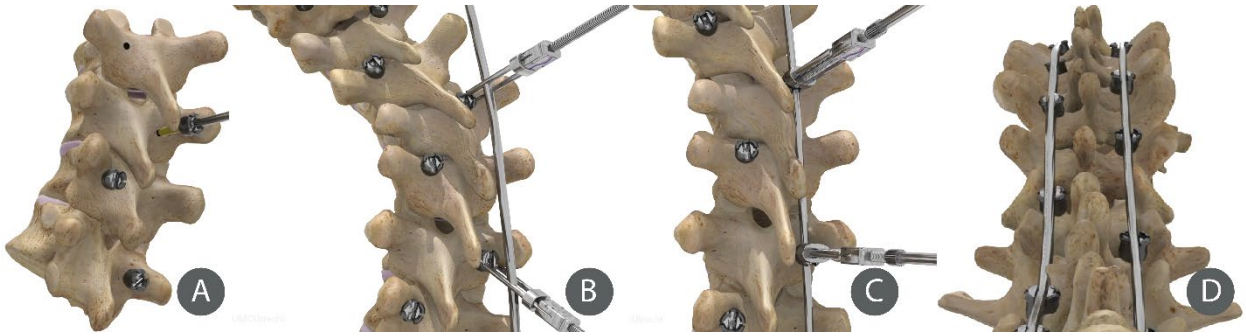


Figure 1.4: Spinal Fusion procedure, figures edited from Mobility Clinic UMC Utrecht [15], (A) insertion of the pedicle screws into pre-drilled holes, (B) placement of the first rod, (C) fixation of the first rod into the pedicle screws to correct the shape of the spine, (D) placement of the second rod to rotate the separate vertebrae

1.5. Problem Statement

Before the rods are placed along the spine during spinal fusion, the surgeon bends the two metal rods manually. There is no guidebook on how the rods should be bend. The degree of bending is entirely based on the experience and judgment of the surgeon and estimation of the desired curvature based on antero-posterior (AP) radiographs. Furthermore, it is dependent on the rod material, as the material's degree of elasticity determines to which extent the rod bends and corrects the spine to a more physiological shape.

Not only stiffness of the rod material plays an important role, but also the material properties of the spine influence the deformation. Besides the material properties of the vertebrae and intervertebral discs, the surrounding tissue, tendons, ligaments, and muscle also affect the stiffness of the entire spinal column. The influence of the surrounding tissues on the spine is dependent on the patient, as the abundancy and magnitude of these tissues can vary per person [15].

Interpatient variety impedes determining the optimal shape of the rod. Preoperative planning that includes assessment of the spine flexibility is helpful to analyze the response of spinal curves to corrective forces and can therefore help to determine the optimal shape for rods, enhancing the accuracy of the spine surgery.

But even if all stiffnesses are known, it is by far obvious how the rods should be pre-bent, as the shape and stiffness of the pre-bent rods in combination with the stiffness distribution of the spine together determine the final post-operative shape of the spine as visualized in *Figure 1.5.*

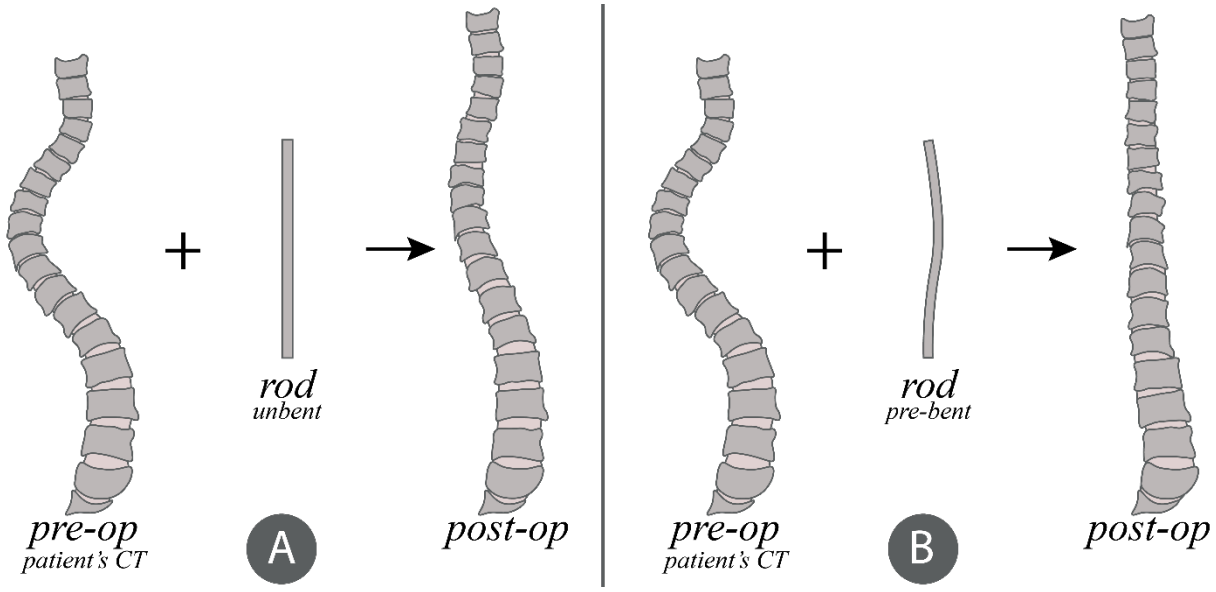


Figure 1.5: The importance of the shape of the rod. Both the stiffness of the rod and the spinal column determine the post-operative spinal shape. The pre-operative spinal shape is set, as well as the material properties and abundancy of spinal tissues. Hence, the pre-bent shape of the rod is important as it determines the post-operative shape of the spinal column. It is expected that the optimal rod must have a curvature in opposite direction of the main spinal curvature, as stated by the orthopedic surgeon. (A) Estimation of post-operative spinal shape after placement of an unbent rod, (B) Estimation of post-operative spinal shape after placement of pre-bent rod

The shape and stiffness of the rods are determined based on the spinal pre-operative geometry. However, the stiffness of the vertebrae and IVD's are relatively unknown. To predict a more accurate shape for the rods, a finite element (FE) model is used in this project. This model requires input parameters such as material properties for the vertebrae and IVD's to simulate their behavior during simulations. The unknown stiffness for this spine model is most likely determined by the stiffness of the IVD's, as this is the most flexible part of the spinal column. The material properties that will be used for this spine model are selected from literature articles, for example articles that used cadaver specimen- or FEM studies.

As the outcome of this model should be the predicted shape of the rods, this study approach will likely require and iterative process or optimalization.

1.6. Research objective

Prior to spinal surgery, preoperative flexibility assessments such as lateral-, or fulcrum bending can be performed to determine what segments should be included and estimate the magnitude of surgery correction. These assessments are mostly simple exercises for the patient that temporarily reduce the Cobb angle. Comparison of this reduced Cobb angle with the initial Cobb angle gives insight on the correction rate that can be achieved during surgical intervention. The outcome of these assessments is not very reliable, with ranges found in literature in a range of 32° - 68° [16], [17], due to the large variety of different assessment executions, the patient's degree of cooperation depending on their pain threshold and the effect of surrounding tissues that cannot be excluded.

The goal of this thesis is to predict the corrective forces needed on vertebrae to correct the pre-operative adolescent idiopathic scoliotic spine to the normal morphology. Pre-operative knowledge on desired corrective forces allows for better estimation of the rod-bending process during pre-planning. The research objective for this thesis is to:

Create a Finite Element Method (FEM) that determines the reaction forces on an adolescent idiopathic scoliosis spine that is straightened. The outcome (reaction forces on the spine) can subsequently be used to determine the pre-deformed shape of surgical rods used to correct the spine in scoliosis surgery.

The research question to this objective is;

- *How can the pre-bent rod shape be predicted for a corrected spine and how does this shape look?*

Accessory sub questions to this research objective are;

- *How can the Cobb angle of the spine model be minimized using FEA?*
- *How are stresses distributed on the spine after corrective rotations are applied?*
- *How can rotation of the vertebral bodies lead to an accessory pre-bent rod model?*

Finite Element Analysis (FEA) is the computational simulation of a complex phenomenon by simplifying it to a solvable solution. The geometry of the spine used in this study will be divided into smaller parts (elements), for which differential equations can be solved. By simplifying the geometry of the vertebrae and IVD's through meshing and excluding other tissue components such as ligaments, muscles and ribs, an approximation of the actual system is estimated. The advantage of FEA is that it allows for prediction of the deformation as a response to the applied forces in silico. It can also predict parameters that are not measurable in vivo, such as the forces and stresses between metal screws and rods and the biological tissue- or stresses in the biological tissue itself. Another benefit of FEA is the repeatability: with scripting, patient specific parameters can be used in a general model that returns valuable feedback, such as stress distribution and displacement magnitude.

In order to run such simulations, the stiffness of the spine is required as input value to mimic the behavior of vertebrae and IVD's. The stiffness is unknown and therefore an estimation is done based on values found in literature.

2. Method and Materials

With the rise of upcoming technology, medical procedures drastically improve. These techniques need testing before they can be practiced on patients. Finite Element (FE) models have been developed to study new methods in silico.

In this thesis, FEA will be used to explore a potentially new method to improve the accuracy of spinal fusion procedures. This chapter describes how a spine- and rod model were created and analyzed.

The project starts with the import of CT slices of an AIS patient. The DICOM slices are segmented in Materialise Mimics and pre-processed in Materialise 3-Matic software, where all segmented parts are prepared and meshed for FEA. With help of the Python Integrated Development Environment (IDE) PyCharm 2021.1.3, scripts are written to build (assemble, adding constraints and boundary conditions) and analyze the spine model in Abaqus 2020. Finally, two models will be produced; a spine model, which is an FE model of which the Cobb angle will be diminished, and a rod model associated with the corrected spine model. In theory, the undeformed spine model in combination with the deformed rod model should lead to the corrected spine shape, as visualized in *Figure 1.5*B.

Although generally two rods are used in spinal fusion, this thesis only describes the analysis of one rod. Firstly, because in spinal fusion, the first rod corrects the overall curvature of the spine, where the second rod improves the axial vertebral rotation. As the focus of this thesis mainly lies on the correcting the overall shape of the spine, the second rod was neglected. Secondly because this thesis is exploratory. The FE models used in this thesis are relatively inelaborate. The effect of adding the second rod is expected to be of more use once the FE models are of higher precision, including ligaments, muscles and even screws. Then, the effect of both rods on the spine and vice versa can be calculated more accurately.

The schematic overview in Figure 2.1 roughly describes the workflow of this chapter “Method and Materials”. All subjects in this figure will be elaborated in the following chapter.

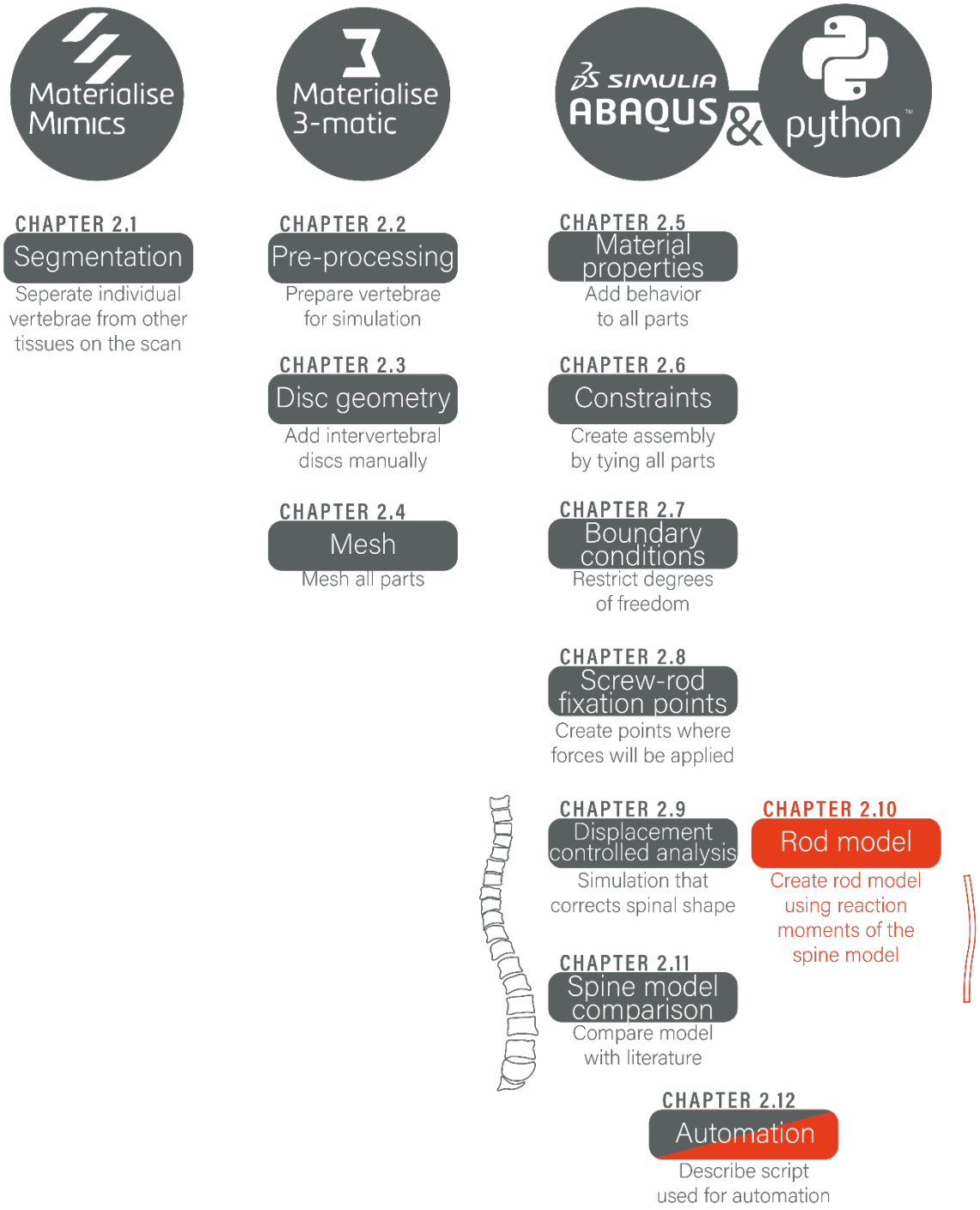


Figure 2.1: Simplified workflow of design process. All paragraphs within the chapter "Method and materials" are displayed below the software that was used for that particular paragraph. In this chapter, two models were created; the spine model (grey), and the rod model (orange).

2.1. Segmentation

Geometry of the spine is required to create an FEM model. An anonymized high-resolution preoperative Computed Tomography (CT) scan taken in 2014 of an AIS patient was obtained to extract the spine geometry. The Digital Imaging and Communications in Medicine (DICOM) file included 842 frames with a slice thickness of 1 mm. The spine had a double curvature, type 1C according to the Lenke classification [22], with a thoracic Cobb angle curvature of 55.7 degrees and a lumbar Cobb angle curvature of 42.4 degrees according to the Swedish orthopedic surgeons (*Figure 2.2B* and *C*). The apex of the thoracic curvature was located on T8 as pointed out in *Figure 2.2C*.

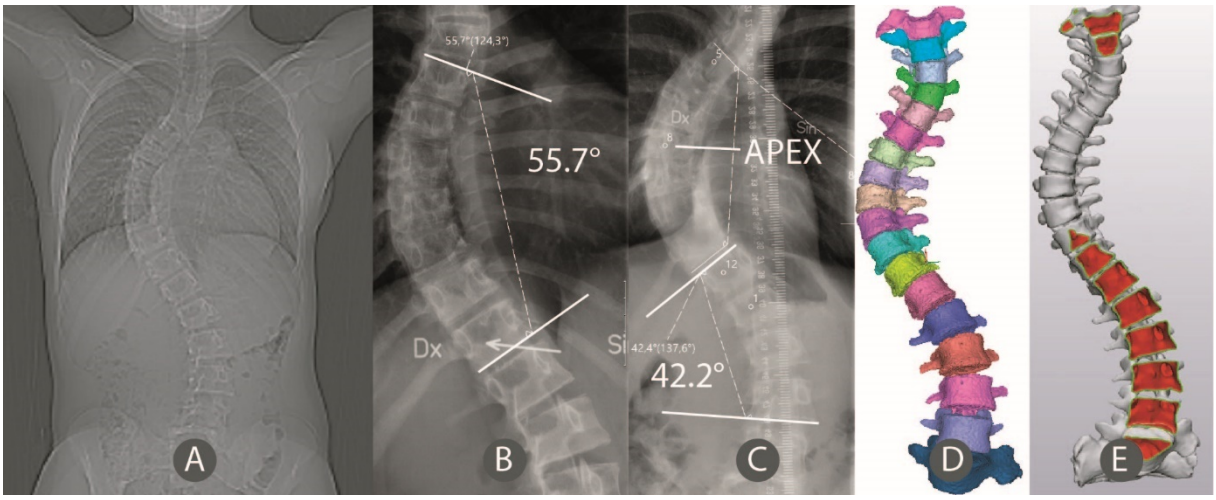


Figure 2.2. (A) Radiograph of the scoliotic spine, (B) Cobb angle measurement of the structural thoracic region, (C) Cobb angle measurement of the nonstructural thoracolumbar region, (D) Individually segmented vertebrae in Materialise Mimics, (E) Pre-processed spine in Materialise 3-Matic

The spine was segmented semi-automatically with a bone threshold gray value in Mimics Medical software package (version 23.0, Materialise, Leuven, Belgium) and separated with a split tool into individual vertebrae (*Figure 2.2D*). The masks were converted to parts after the smart fill tool filled all holes. *Figure 2.2E* shows an intersection of the model in Materialise 3-Matic, prepared for simulation.

2.2. Pre-processing

The parts were imported to the Computer Aided Design (CAD) program 3-Matic Medical software package (version 15.0, Materialise, Leuven, Belgium) to pre-process the parts. The surfaces of the imported parts were smoothed and wrapped. Smoothing reduces the amount of 'bumps' in the geometry created in the segmentation process, whereas the wrapping function is used to fill and wrap the remaining holes. Smoothing and wrapping both enhance the appearance of the parts, but most importantly it reduces artificial surface stresses by making the surface more homogeneous and increases the chance of running successful simulations. The number of triangles was reduced to make the model easier to work with. With the Fix Wizard, all parts were checked on possible errors.

2.3. Geometry of the discs

Intervertebral discs (IVD's) are not detectable by CT scans, and therefore needed to be created manually after the vertebral geometry was simplified. Smooth curves were drawn on the caudal and cranial surfaces of each vertebra, except for the sacrum S1 (*Figure 2.3B* and *Figure 2.4B*). The S1 bone was the most inferior bone of the spine and only got a smooth curve on its cranial site. The curve both attracted and attached to each surface. Surface sets were constructed by splitting the surfaces of the bone entities by the created curves (*Figure 2.3C*). Discs were formed by lofting all intervertebral surface sets *(Figure 2.3D→E* and *Figure 2.4 B→C*).

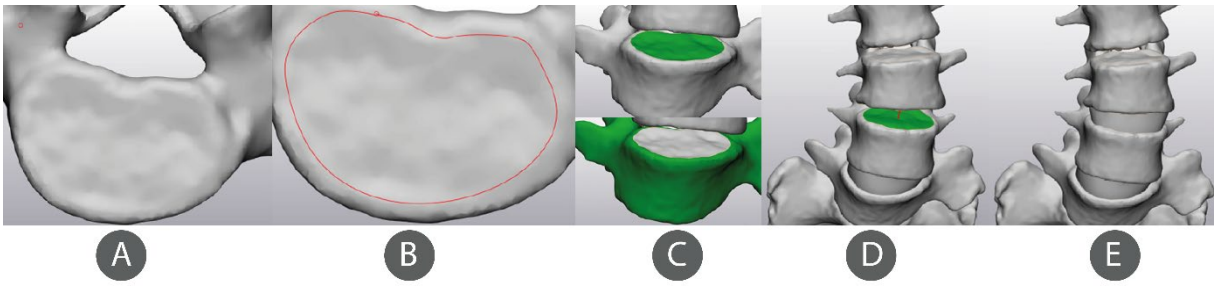


Figure 2.3: Lofting intervertebral space, (A) Smoothed and wrapped superior side of vertebral body, (B) Smooth curve on vertebral surface, (C) Most vertebrae are split in multiple surfaces, (D) Two selected surfaces, (E) Segments after lofting the intervertebral space

Subsequently, this disc needed to be divided in a nucleus pulposus (NP) and an annulus fibrosus (AF). Within each disc, a center of volume (CoV) was placed in the middle of the geometric structure to determine the midpoint of a sphere with a radius that was used to partition the disc into an AF and a NP (*Figure 2.4D*). The AF was shaped by subtracting the sphere from the disc (*Figure 2.4E*);

$$V_{AF} = V_{disc} - V_{sphere}$$

In the cranial direction, the volume and size of the discs decreased and therefore the radius of the sphere was chosen to be declining as well, to prevent breakage of the annulus fibrosus after subtraction. The radius of each sphere can be found in Appendix D. In some discs, the sphere was slightly moved to prevent the breakage of the outer AF part. Rupture of the AF leads to herniated discs, where the gel-like NP gets pushed out of the AF. Herniated discs lie not within the scope of this study and were for that reason not simulated. Free movement of the NP leads to a shift in the flexibility and would result in a different outcome.

The NP was created by intersection between the sphere and disc so that the overlapping portion remained (*Figure 2.4F*). All parts (original loft, sphere, and the CoV), except the subtracted and intersected parts were hidden, resulting in a remaining NP and enveloping AF. This modeling process is visually described in Appendix A. The endplates were excluded in the modelling process, because they have a relatively high young's modulus compared with IVD's (AF + NP) and therefore contribute less to the flexibility of the spine. The young's modulus of endplates and vertebrae are commensurate with a magnitude around 12000 MPa [18]. The main focus of this thesis is the spinal shape correction caused by the IVD's. The vertebrae are modeled for geometrical purposes and used to place screw-rod fixation points. The deformation of the bone itself lies not in the scope of this exploration. Modelling the endplates would lead to a higher computation time, more chance on computational error and thirty-six (two endplates per IVD) more parts that needed to be modeled.

This whole section was done manually because the occurrence of errors is high. For example, the superior segments are smaller than the inferior segments and require different settings. Due to the complex geometry that is based on real tissue, creating curves on surfaces, and lofting the intervertebral space regularly led to incomplete or misshapen discs. Therefore, this relatively labor-intensive part was done by hand.

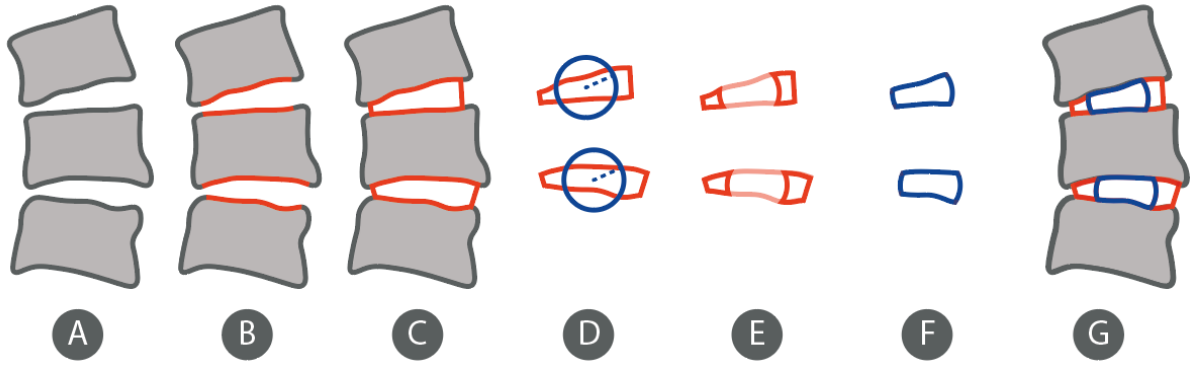


Figure 2.4: Modeling of the intervertebral discs with (A) the vertebrae, (B) surface sets created on superior and inferior vertebral body surfaces, (C) the intervertebral spaces lofted, (D) disc with sphere, (E) subtraction of disc and sphere to create AF (orange), (F) intersection of disc and sphere to create NP (blue), (G) complete segment with three vertebrae and two discs containing an AF (orange) and NP (blue).

Figure 2.4G shows two vertebral segments (L2-L4) with IVD's. The whole segmented spine (C7-S1) consisted of 19 vertebrae and 18 discs with both an AF and an NP, which ultimately resulted in a spinal geometry of 55 individual parts.

2.4. Mesh

Meshing is required to simplify the otherwise infinite continuous model into a finite number of equations that can be solved in the FE-code. All parts were meshed to prepare for Finite Element Analysis (FEA). As the IVD's are expected to be mostly responsible for the flexibility of the spine [19], the surface meshes for both the AF's and NP's were chosen to be finer than the surface meshes for the vertebrae. The meshing was done in Materialise 3-Matic instead of Abaqus to preserve the surfaces made in 3-Matic. As 3-Matic does not offer many elements type options, all parts were meshed using the linear element type tet4, also called type C3D4 (shown in *Figure 2.5*). This element type is tetrahedral shaped and has four nodal connectivity sites that make up the three-dimensional structure of the element.

The triangle edge lengths of the discs can be found in Appendix D. More cranial IVD's received a finer surface mesh than caudal discs due to their size. The target triangle edge length of the vertebrae was set to 10 [mm]. Before and after the application of a uniform surface mesh, the number of triangles of all parts was reduced whilst preserving the mesh quality. Subsequently, a volume mesh was applied to all parts to accommodate the objects with an interior structure.

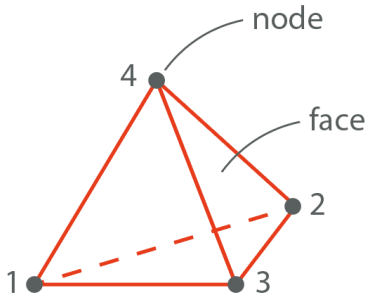


Figure 2.5: Element type tet4 consisting of nodes and four faces

2.5. Material properties

Material properties are added to the parts to determine how the parts must behave during simulation.

To do so, the surface- and volume meshed parts were imported from Materialise 3-Matic into Materialise Mimics [20] to add 'material property tags' to each part. The material assignment options in Mimics can be related to Hounsfield units, but do not provide options for different material behaviors, such as young's-modulus and Poisson's ratio. Therefore, the material properties were later adjusted in the Finite Element Method (FEM) software program Abaqus (Abaqus 6.14,2014, Dassault Systems, Vélizy-Villacoublay, France). The Complete Abaqus Environment (CAE) of Abaqus provides an environment for both the modelling and visualization process of FEA. All 55 parts, - with input (".inp") extension, - were imported into Abaqus as models (not parts) to ensure they retained their surface sets that were created in Materialise 3-Matic.

A small material property study was performed on the L4L5 lumbar IVD's to give insight on the best choice for the entire model. A comparison study of three frequently used material behaviors for IVD's are linear elastic (LE) models [21], hyper elastic (HE) models [15] and viscoelastic (VE) models [22] was performed and can be found in Appendix C.

Linear-elastic material properties were chosen instead of hyper- or visco elastic properties to reduce computational errors and computation time. There are several Finite Element Analysis articles concerning the spine that use homogenous isotropic linear-elastic material properties, which makes comparison with other studies reliable [23], [18].

Table 2.1: Linear-elastic material properties [18]

Component name	Young's Modulus [MPa]	Poisson's ratio υ[2]
Annulus Fibrosus[1]	4.2	0.45
Nucleus Pulposus	1	0.499 (nearly incompressible)
Vertebra (cortical bone)	12000	0.30

[1]ground substance, [2]value generally ranging between 0 and 0.5

Table 2.1 displays the linear-elastic material properties that were used for the simulation. The vertebrae have a significantly higher young's modulus than the two components of the IVD. The vertebrae were initially supposed to be modeled as rigid bodies. However, importing the parts as rigid bodies in Abaqus was no option as long as the in 3-Matic created surfaces were to be preserved.

The Poisson's ratio of the disc components was chosen to be nearly incompressible, meaning they do not change in volume as reaction to deformation.

The material properties of the AF are based on the ground substance. The AF consists of concentric collagen I layers arranged in opposite direction as shown in *Figure 1.3*. Due to large differences in AF geometry, the layers were not modelled and the properties of a ground substance were used instead.

Parts, materials, and sections of the objects were copied to one model called the spine model. Within this spine model, instances of the models were created to facilitate the assembly. Assemblation was fully automated in Python 3.9 using the integrated development environment PyCharm (JetBrains, version 2021.3.2). The code can be found in Appendix I.

2.6. Constraints

All 55 instances were connected through tie constraints. First, the superior surfaces of both the NP's and AF's were merged. The same was done for the inferior surfaces. The vertebrae' inferior surfaces were tied using node to surface constraints to the IVD's' superior surfaces and vice versa. Node to surface contact elements was chosen as it uses a less advanced contact algorithm and memory than surface to surface contact elements. The light and darker colors used in the IVD's in *Figure 2.6* show how the surfaces were merged and visualize the use of master- and slave surfaces.

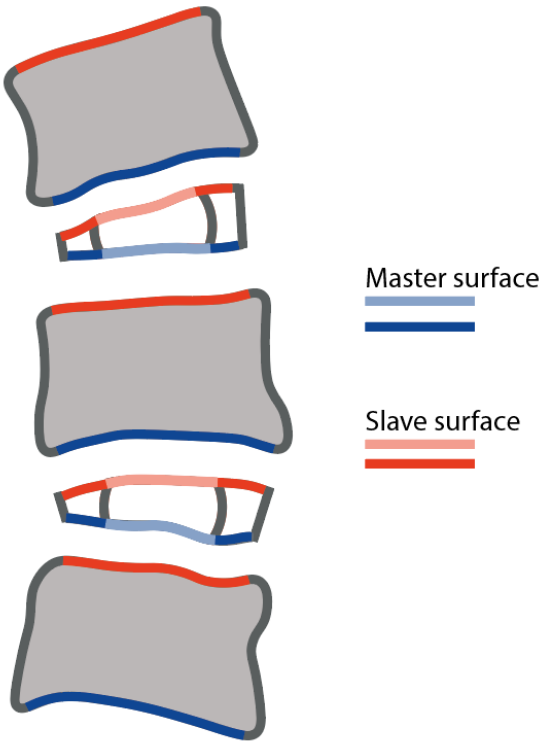


Figure 2.6: Location of the master- and slave surfaces. Blue lines indicate master surfaces, orange lines represent slave surfaces.

2.7. Boundary conditions

Boundary conditions are types of constraints that fixate the assembly in space, preventing the model from floating away after applying forces or deformations. *Figure 2.7* shows the locations and types of boundary conditions applied to the model. An encastre was applied to all nodes along the most inferior vertebra, - S1, restricting movement in all degrees of freedom (DOF).

The most superior vertebra, - C7, - is constraint in the transversal plane, but can translate along the z-axis. These boundaries were set in accordance with an orthopedic surgeon of UMC Utrecht. It mimics the situation during surgery, where the spine is posteriorly operated, and the upper and lower vertebrae are stabilized.

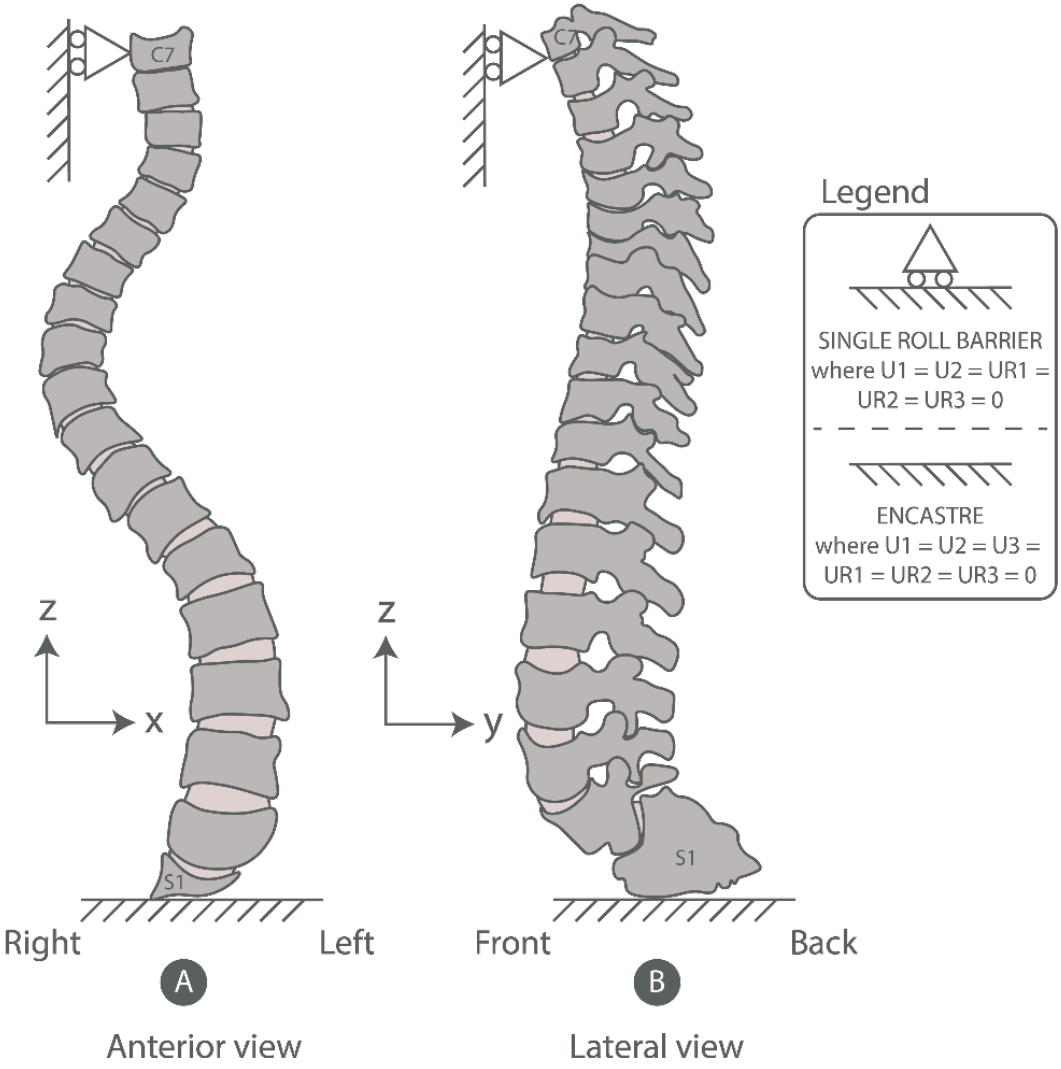


Figure 2.7: Boundary conditions of the spine model in (A) the anterior view and (B) the lateral view.

2.8. Screw-rod fixation points

Before FEA can start, some kind of input is required. In this case, the input parameters are displacement constraints, more specifically rotations (in degrees). These rotations will be applied to the vertebrae. As reaction to these vertebral rotations, the IVD's will deform, resulting in a different spinal shape. However, these rotations need to be applied to a point of engagement. This point will be called a reference point (RP). When connecting this RP to some nodes of the accessory vertebra, the entire vertebra reacts to the applied rotation on that single RP.

In order to apply these rotations to the vertebrae, the process according to *Figure 2.8* is required;

A. Determine location (reference point) where rotation should be applied
B. Add these reference points in the spine model
C. Connect the reference points with nodes of the accessory vertebra

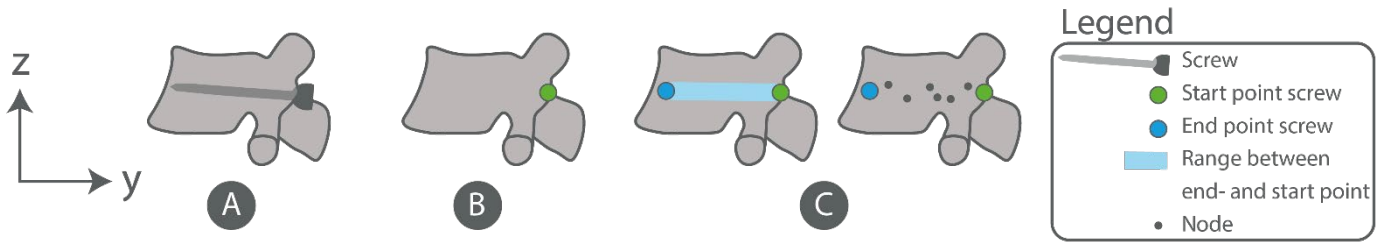


Figure 2.8: Engagement points for applying rotations on lateral vertebral view, with (A) screw placement used in spinal fusion, (B) Point of the screwhead used as starting point, and also as reference point, (C) the blue bar indicating the range between starting and end point of the screw, that is used to select nodes. These nodes will be coupled to the reference point (green), so that the applied rotation affect the entire vertebra.

Hereafter, the correction of the spinal shape can take place. By rotating the RP, the location of the vertebrae will change. The IVD's between the vertebrae will deform as reaction to the vertebral movement because of their low young's modulus, leading to deformation of the spinal curvature.

The three steps (A, B, C) from *Figure 2.8* are elaborated on the following pages.

A. Location of rotation points

The spinal fusion procedure was looked at more closely, to determine the location where rotations should be applied. During this surgery procedure, the vertebrae of the segments that are to be corrected receive screws. These screws are the points of engagement where the rod is placed. Eventually, the bilateral rods connect the screws altogether and maintain the corrected shape. In this thesis, the screws are chosen to be the location where the rotations will be applied. The rotation applied to these points will mimic placement of the rod.

Figure 2.9A visualizes the pre-operative surgery plan in accordance with the orthopedic surgeon at UMC Utrecht hospital. The region T3 up till T12 was recommended to be included in the procedure. The grey dots represent the screws that are inserted through the pedicle up to the cortex of each vertebral body. The most inferior and superior vertebrae received screws through bilateral pedicles to anchor the two rods. Screws were inserted alternately at the left- and right side of all vertebrae between T3 and T12. Spinal rods will be bilaterally attached to the predetermined pedicle. The dotted red line in *Figure 2.9A* shows how both rods are led through the screws.

Figure 2.9B visualizes the screw-rod fixation points that were created in the FEM analysis. More points were created than recommended to allow potentially future analyses that require more points. *Figure 2.9C* shows the screw-rod fixation points that were used for this thesis. As described before, only one rod was simulated and therefore only one side of the vertebrae was used for rotation. There were no screw points placed on the most inferior vertebra S1, because an encastre constraint was placed that restricts translation and rotation in any direction.

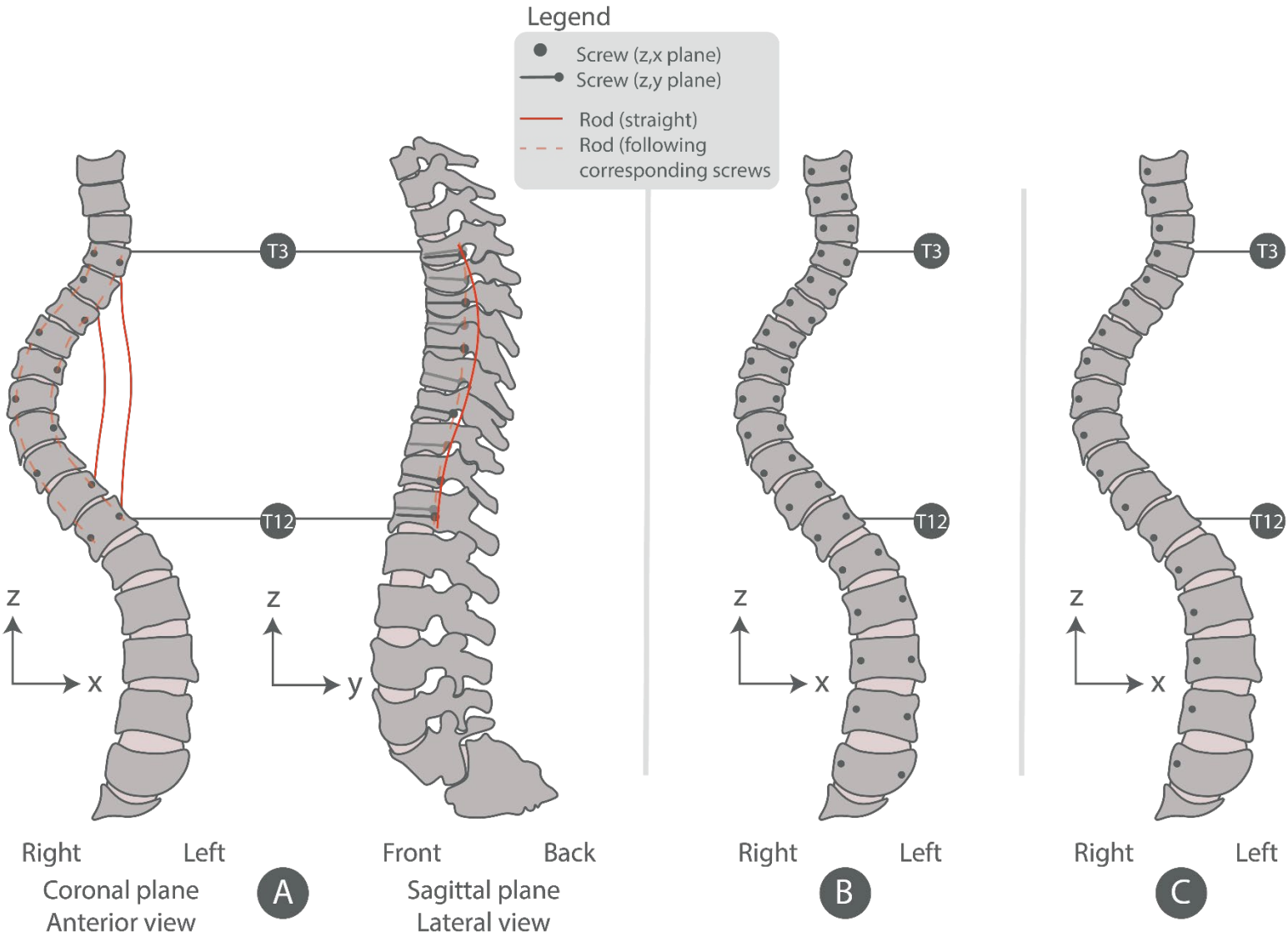


Figure 2.9: pre-operative surgery plan with (A) surgery planning according to the orthopedic surgeon, (B) the created screw-rod fixation points during simulation, (C) the screw-rod fixation points used for analysis

B. Add Reference Points

Once the location of the screw points was determined, the RP's could be created. In addition to the RP's, two points were created, within which nodes were selected that could be coupled with the RP's, so that for the simulation, the applied rotations would be administered to the entire vertebra.

To do so, coordinates of the desired locations were required. Therefore, the location of the starting point and end point of the screw were manually placed in Materialise Mimics. *Figure 2.8C*, as well as *Figure 2.10A* and *B* show how this was done. The green dot represents the location of the screwhead after placement. The blue dots pinpoint the location where the screw touches the vertebral cortex. One screw contains two points; an end and a starting point. Each vertebra accommodates two screws. This means a total of $2\ (green\ dots) * 2\ (blue\ dots) * n_{vertebrae} = 72$ points with cartesian coordinates (x, y, z) were created. These points were saved to an xml file and placed on the spine model in Abaqus. The yellow lines in *Figure* 2.10 represent the lines in Abaqus that connect the start- and endpoint of these vectors.

RP's were created that were stationed on the vertebrae to function as attachment point for applied forces and deformation. The RPs are displayed in *Figure 2.10C* and had the same coordinates as the starting points (green dots).

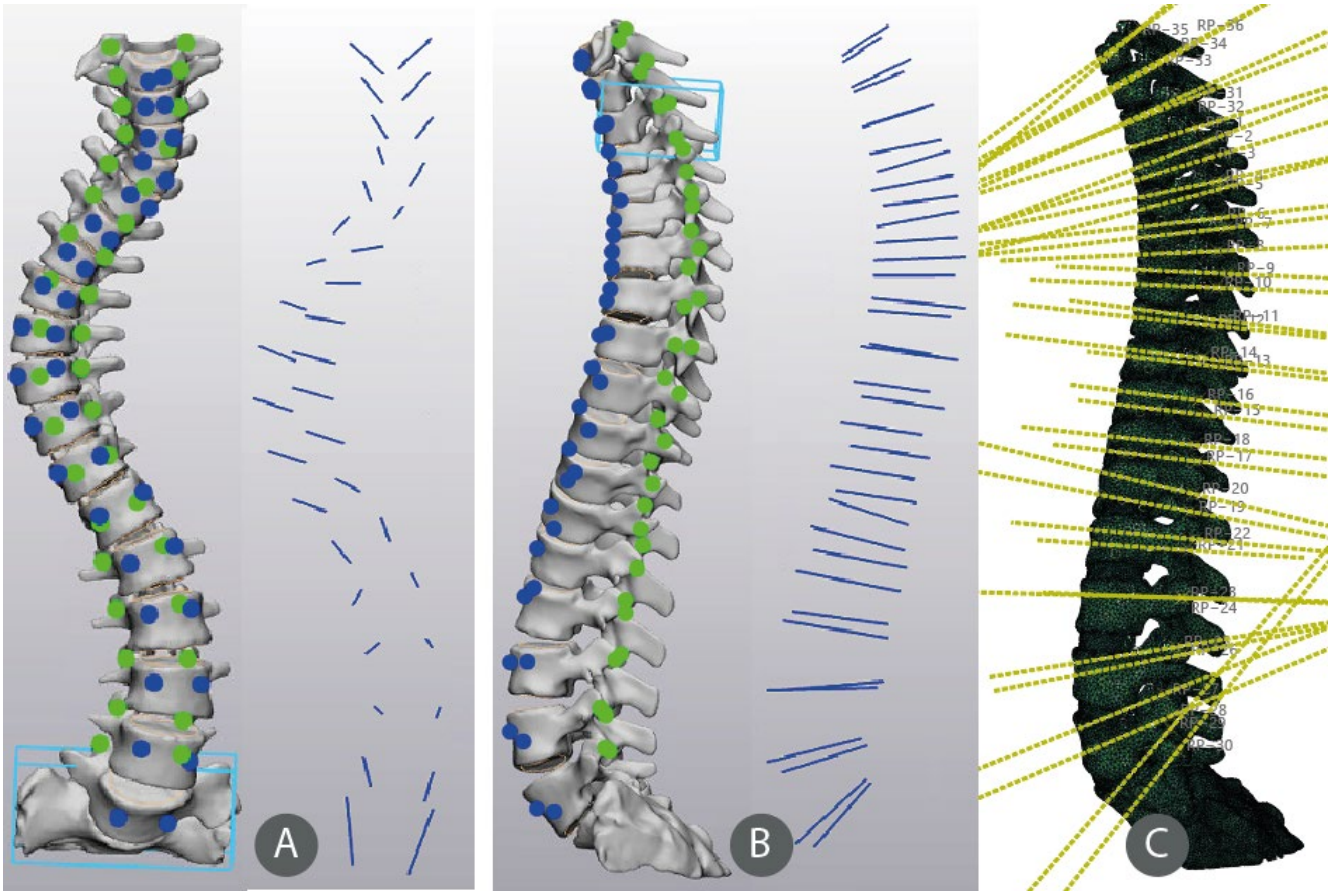


*Figure 2.10: Location of screw attachment points. (A, B) Starting point (green) and endpoints (blue) in Materialise 3-Matic in coronal and lateral view. (C) Reference points in Abaqus in lateral view. The yellow dotted lines are the vectors of the screws and the reference points are named "RP-n", with n as counting number up to 36 (number of vertebrae * 2 sides)*

C. RP coupling with vertebrae

To select the nodes that should be coupled with the RP's, a cylinder was drawn between the end- and starting points (the green and blue dots in *Figur*e 2.10). Subsequently, a radius of 1.5 mm was chosen to create the cylinder using Python (Appendix I).

This radius was chosen because to be 1.5 mm to select enough nodes for rotation as shown in *Figure 2.11C*. A larger radius would lead to overlapping nodes, as shown in *Figure 2.11B,* where a node from another vertebra is selected in the wrong set. The vertebra with overlapping node would rotate unintentionally together with the vertebra that must be rotated. A radius that is too small would lead to too few nodes that are selected as shown in *Figure 2.11A*. The reason why not all vertebral nodes were selected to couple with the RP's, is because this small node selection gives more control during rotation of the vertebrae.

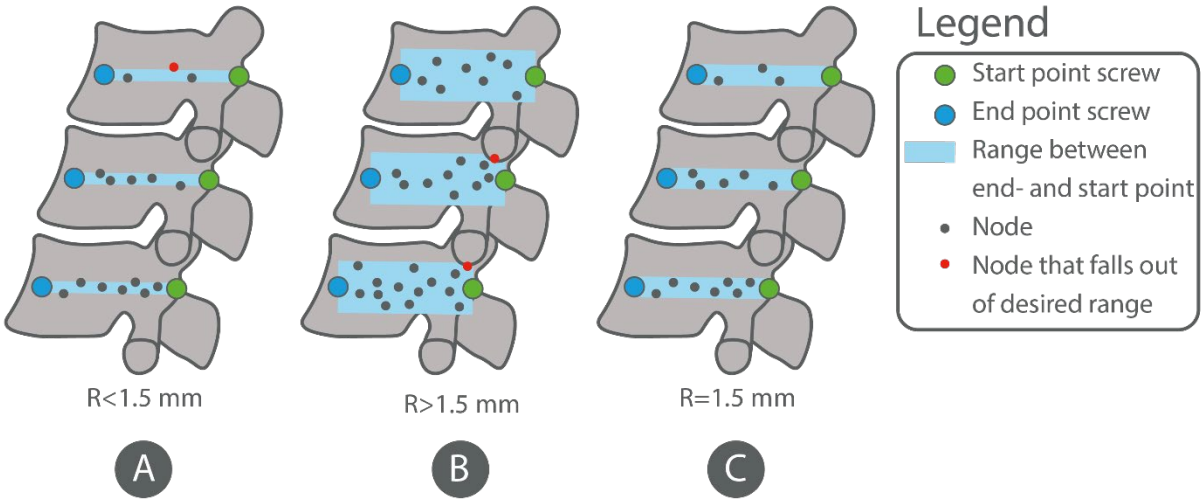


Figure 2.11: Selecting nodes for coupling constraint; (A) cylindrical radius too small: essential nodes fall out of range, (B) cylindrical radius too large: nodes from other vertebra are selected too, (C) cylindrical radius correct: enough nodes per vertebra and no overlapping nodes.

All nodes that fall within this cylinder were selected and coupled to the RP's (*Figure 2.12*).

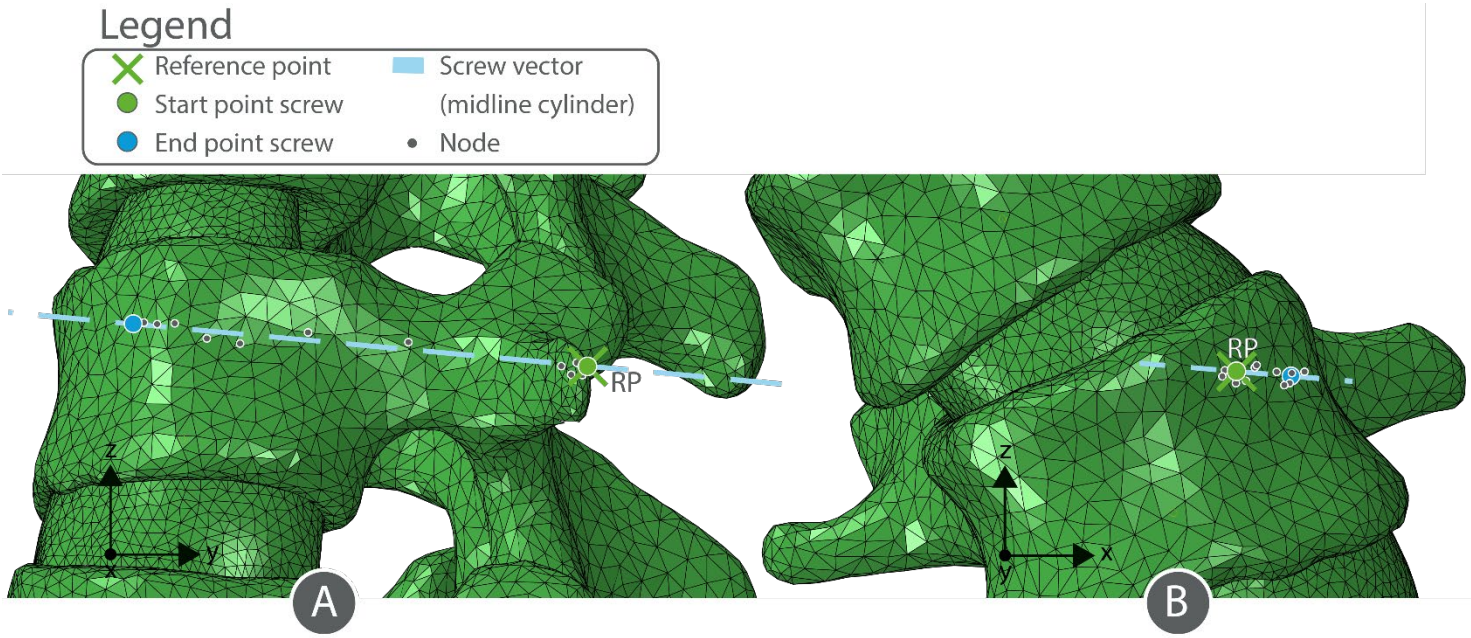


Figure 2.12: Selecting nodes for coupling constraint in the FE model. The blue dotted line is the midline of the cylindrical range. Note that the actual blue range shown in figure 2.11 is not visible in the model., (A) Lateral view (B) Coronal view

2.9. Displacement controlled analysis

The shape of the scoliotic spine was improved by applying step-by-step rotations to the vertebrae. The rotation was applied to the RP's that were coupled with vertebral nodes as described in the previous paragraph.

Although the orthopedic surgeon would only operate the segments T3-T12 as presented in *Figure 2.9*, the goal of this simulation was not to mimic surgery, but to aim for a most straightened spine. Therefore, also the vertebrae superior- and inferior to the recommended operation segments were adapted.

Each vertebra was rotated along two different axes; first along the y-axis (*Figure 2.13A*) to correct the coronal shape, and secondly an axial rotation was applied along the z-axis, to align the processus spinosi in the axial plane (*Figure 2.13B*). There was no rotation applied along the x-axis, because the desired lateral spinal curvature was not examined in this thesis.

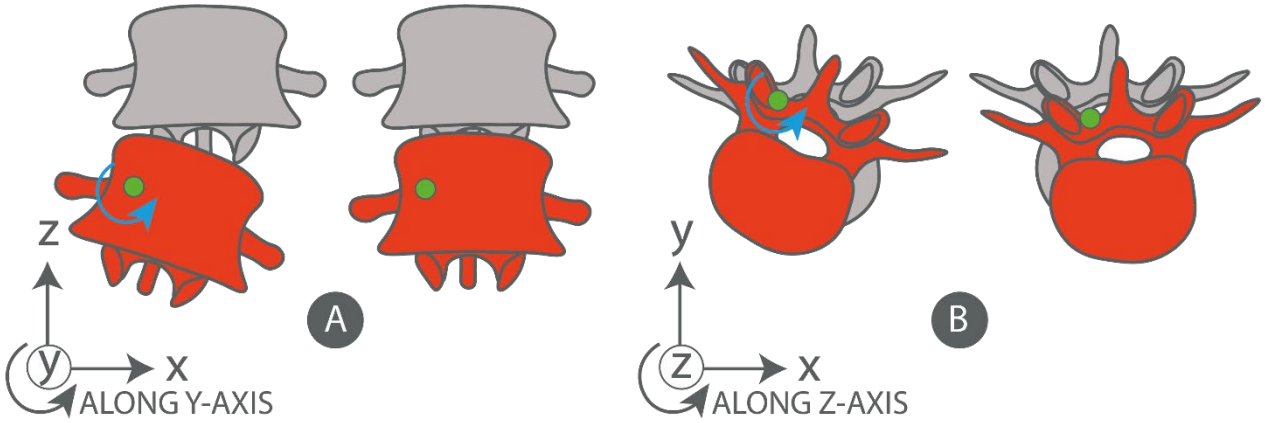


Figure 2.13: Vertebral rotation in two different directions. The orange vertebra is the rotating vertebra. The green dot is the reference point around where the rotation is applied. The blue arrow indicates the direction of rotation. (A) Rotation along the y-axis in coronal plane, (B) Rotation along the z-axis in axial plane

All vertebrae, -except for the encastred inferior S1, and C7 and T1-, were rotated along the y-axis and the z-axis. C7 and T1 seemed straight enough and were therefore excluded from the rotational correction procedure. The magnitude of each rotation was an estimation that improved the vertebral correction. In total, $16 * 2 = 32$ rotations were applied.

The deformation that occurred as a reaction to the rotation is mainly in the IVD's, as those are the most flexible parts of the structure. It should be recognized that each rotation has a consequence for the entire deformation of the spine. Hence, also the proximal and distal vertebra will rotate, so previous rotations of vertebrae can become enlarged or reduced. For this reason, all rotations were applied subsequently instead of all at once. Therefore, every single rotation was applied via 32 subsequent steps. Every FE model contains an initial step, or a "0^{th} step" as it were, wherein the boundary conditions and predefined constraints are described.

After the initial step, general static steps were created that each contained one rotation per vertebra. Upon recommendation of the orthopedic surgeon, the correction process took place from top (C7) to bottom (L5), as shown in *Figure 2.14*.

As each rotation affects previous rotations, the procedure can only be solved in an iterative procedure. However, there was decided not to go back to previous rotations due to time limitations, but fixate each vertebra in the horizontal direction with a single roll barrier, to prevent the model from undesired translocation during transition of subsequent segments. This can still lead to rotations of each individual vertebra and therefore does not necessarily lead to the ideal spinal shape. In the last final step, all roll barriers were replaced by encastres, as shown in the last spinal column of Figure 2.14. This is a necessary step, because only fixed ends can have reaction moments, which are required to determine the shape of the accessory rod model. A body with fixed support has no degrees of freedom (DOF) and therefore has six support reactions to retain equilibrium.

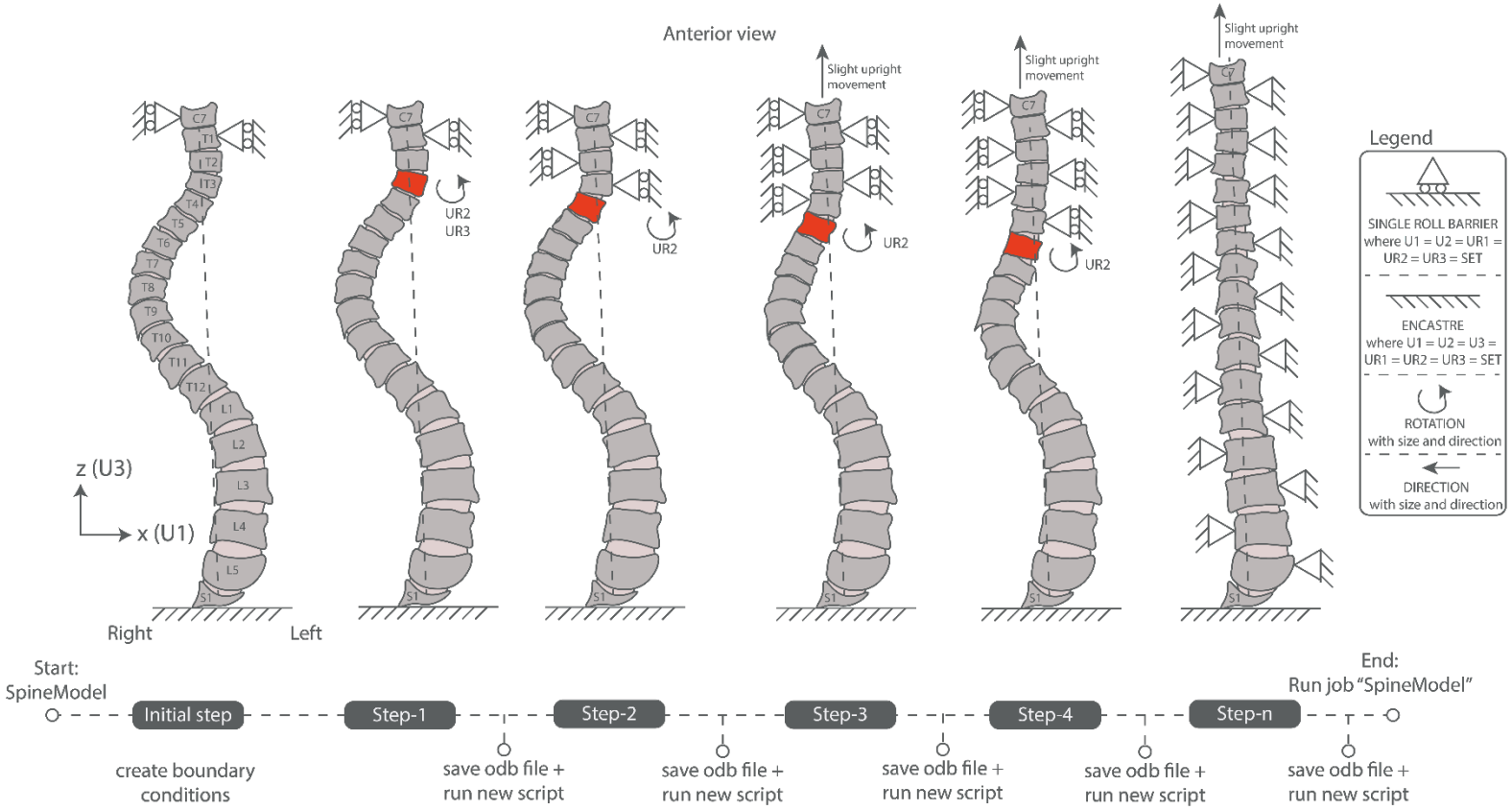
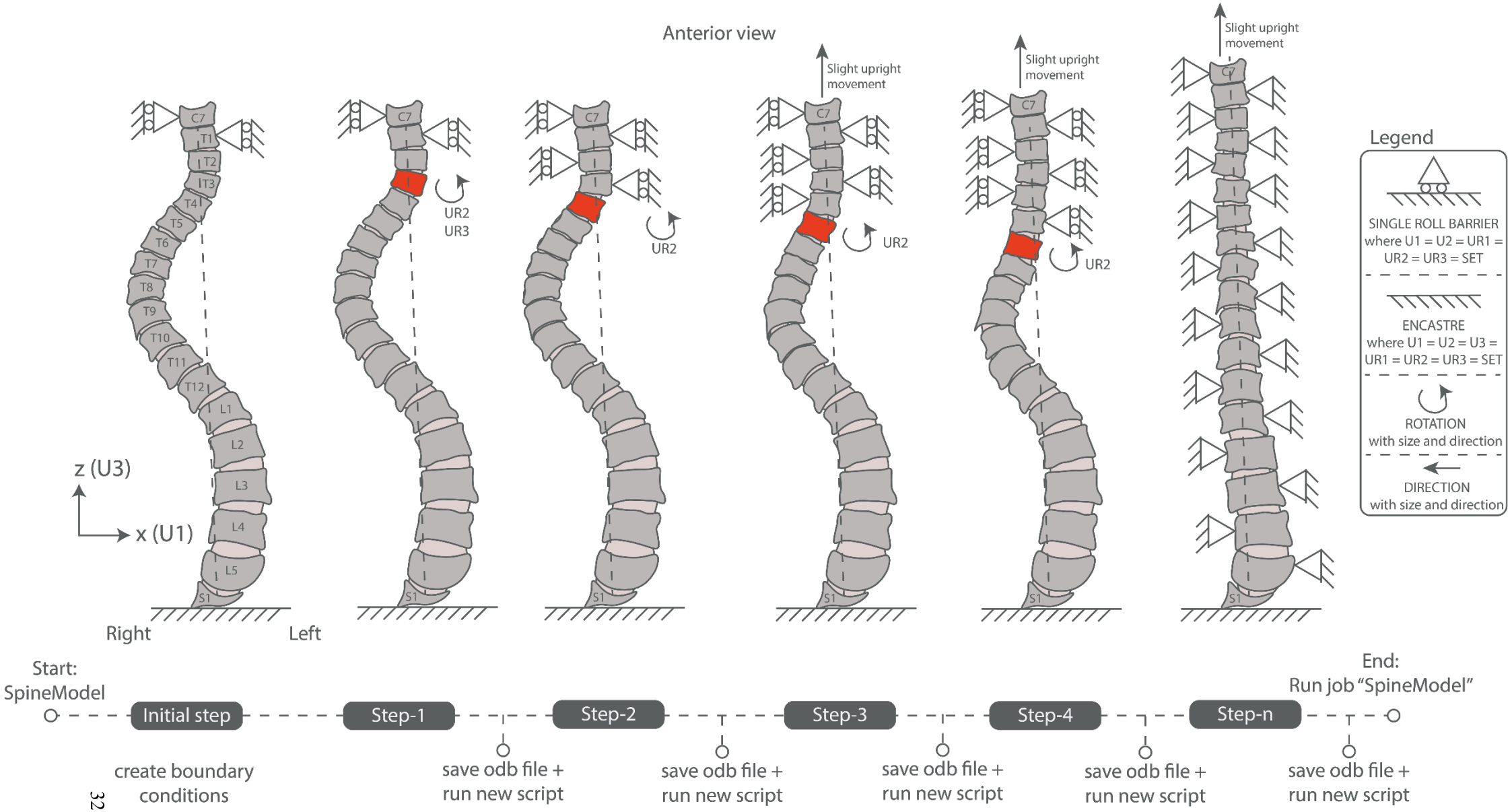


Figure 2.14: Idealized correction process of the spine shown in multiple steps. The orange vertebra is the vertebra that gets rotated during that step. A larger version of this figure can be found on the next page.


Anterior view
C7
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
L1
L2
L3
L4
L5
S1
z (U3)
x (U1)
Right
Left
UR2
UR3
Slight upright movement
Legend
SINGLE ROLL BARRIER
where U1 = U2 = UR1 = UR2 = UR3 = SET
ENCASTRE
where U1 = U2 = U3 = UR1 = UR2 = UR3 = SET
ROTATION
with size and direction
DIRECTION
with size and direction
Start:
SpineModel
Initial step
Step-1
Step-2
Step-3
Step-4
Step-n
End:
Run job "SpineModel"
create boundary conditions
save odb file + run new script

2.10. Rod model

The previous paragraphs lined out the modeling process of an AIS patient's spine. This chapter describes how an accessory rod model was achieved by transmitting the reaction moments, -that are calculated in the spine model that are required to deform the spine model to its new shape-, to the rod model (Figure 2.15).

For this to happen, the following steps were executed;

A. Build an FE model of a rod
B. Add reference points (RP's), so that reaction moments can be applied to them
C. Run the analysis

A. Building a rod model

In essence, the rod model is nothing more than a metal solid bar, that requires a radius, length and material properties.

The rod was modeled in Abaqus according to the following steps visualized in *Figure 2.16*. First, a circle with a radius R_{rod} of 3 mm was sketched. Subsequently, a three-dimensional deformable body was created by extruding the sketch with a length L_{rod} $[mm]$ of 196 mm and a volume V_{rod} of 5545 mm^3. The length was based on the number of vertebral segments that should be included according to the orthopedic surgeon, as visualized in *Figure 2.9*, namely segments T3-T12. This length is a trade-off between enough post-operative flexibility for the patient and stability for the corrected spine. Rods covering more vertebral segments will reduce the patient's mobility by limiting the freedom of movement. Constraining too few vertebral segments after surgery can lead to under-correction and results in a Cobb angle that is quite large [12].

The length of the rod was determined by calculating the absolute z-value between the inferior and superior reference point of the selected area in the spine model (T3-T12), see *Figure 2.16A*;

$$L_{rod} = |T12_z - T3_z|$$

$$L_{rod} = |-701.1928 - 505.0691|$$

$$L_{rod} = 196.1237\ mm$$

After extrusion, material properties were assigned. Frequently used metals for spinal are Titanium Ti6Al4V (Ti) and Cobalt-Chromium (CoCr) based alloys [24]. These biomaterials have their own set of material properties that can be beneficial or disadvantageous in certain surgery cases. Stainless steel (SS) used to be the gold standard, but is hardly used since the emergence of Ti and CoCr rods for spinal fusion. Ti alloys have beneficial biocompatible and biofunctional characteristics. They are corrosion resistant and show less artifacts in MRI studies than SS and CoCr [12]. The high Young's modulus of CoCr result in better stability over time after spinal fusion, because it hardly leads to post-operative deformation [25]. Higher rod stiffness allows for smaller rod diameters and therefore less material is needed. The fracture rate of Ti is higher than CoCr as presented in *Table 2.2* [26]. However, CoCr shows relatively more distortions on MRI than Ti [27]. Moreover, CoCr is more expensive than Ti and SS.

Analyses of the rod model were run with both Ti and CoCr properties.

The Young's modulus E of the rod material is generally higher than the elastic modulus of the intervertebral discs, respectively, ~11-21 GPa (*Table 2.2*) and ~1-4.2 MPa (*Table 2.1*). Hereafter, the part was meshed. The mesh contained were 8820 elements of type C3D8R and 11032 nodes.

This large difference in stiffness between rod and spine changes the physiological force loading of the spine to the segments fused by pedicle screws and rods. The instrumented spinal segments have to bear less weight, in comparison with non-fused segments. These non-fused segments bear a relatively high force, which can lead to adjacent segment degeneration (ASD) [28].

Table 2.2: Material properties of the rod model [29], [30]

	CoCr	*Ti*
Young's Modulus E [MPa]	210.000	110.000
Poisson's ratio υ	0.29	0.35
Fracture rate [%]	2.7	8.6

B. Add reaction moments to the reference points

Earlier, placement of reference points (RP's) on the spine model was described. The placement of RP's on the rod model is comparable. First, an axis was created on the left side of the cylinder in the z-direction to guide the reference points along the rod, as shown in *Figure 2.16B*. RP's were placed on the rod model with equal z-coordinates of the RP's on the spine model, as shown in *Figure 2.15B* and *Figure 2.16A*.

Each RP was then bound with a node set consisting of six nodes, as shown in *Figure 2.15C* through coupling constraints, so that these nodes were able to react on the applied moment. Coupling constraints inhibit the motion of a surface to the motion of a single point, - in this case a RP [31].

The whole model was automated using Python, except for creation of the node sets required for the coupling constraints, which were constructed manually.

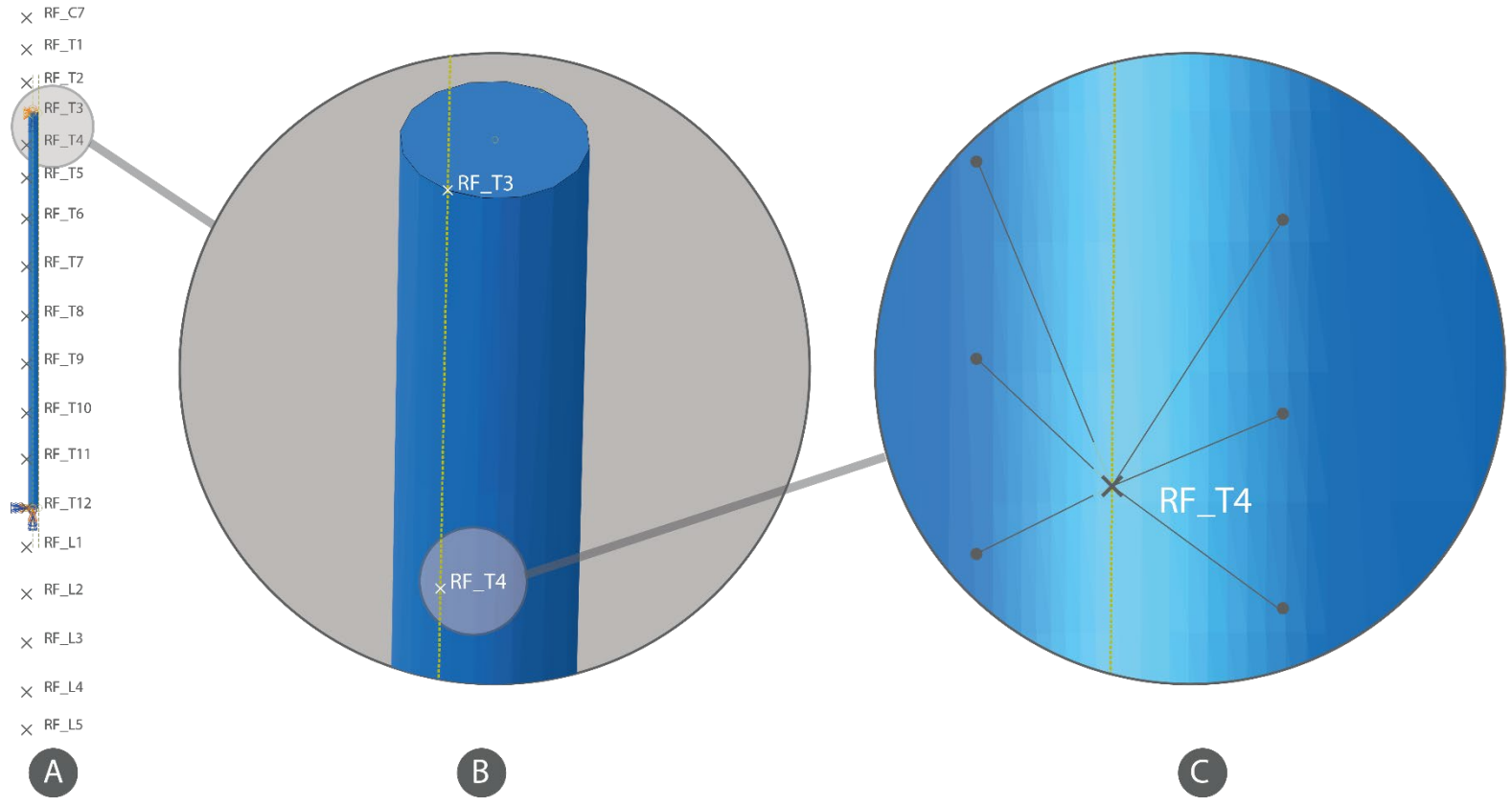


Figure 2.15: Creating the rod model and placement of reference points (RP's). (A) The rod model occupies RFT3 up till RFT12. Boundary conditions are shown at superior- and inferior surfaces, (B) RP points are placed along a longitudinal axis (yellow dotted line), (C) Each RP is coupled to six nodes (grey dots).

C. Preparing for analysis

Before analysis was run, boundary conditions were applied. Sets of the superior, inferior and lateral surfaces of the rod were created to function as attachment sites for boundary conditions. The inferior surface set was encastered in all directions and the superior surface set was given a z-asymmetric boundary condition, meaning it was restricted in all directions, except for the longitudinal z-direction and the rotation along x- and y axis (U1=U2=UR3=0). These boundary conditions are similar to the boundary conditions used in the spine model, see *Figure 2.16A*. Thereafter a mesh was generated.

The reaction moments gathered from the last step of the spine model (*Figure 2.16A*: *last step*) were applied to the rod model (*Figure 2.16B*: *initial step*). Only the reaction moments along the y-axis at the recommended surgery region shown in *Figure 2.9* were used. The reactions along the z-axis were not applied to the rod as rotations in z-direction only lead to axial torsion and do not affect the shape desirably of a rod that is extruded in the z-direction.

Boundary conditions were applied to vertebra T3 and T12 and therefore the accessory reaction forces were not implemented on these vertebrae. Moments were applied on vertebra T5, T7, T8, T9 and T11 in five subsequent steps. The moments in T4, T6 and T10 were zero, because they were not or hardly rotated in the spine model.

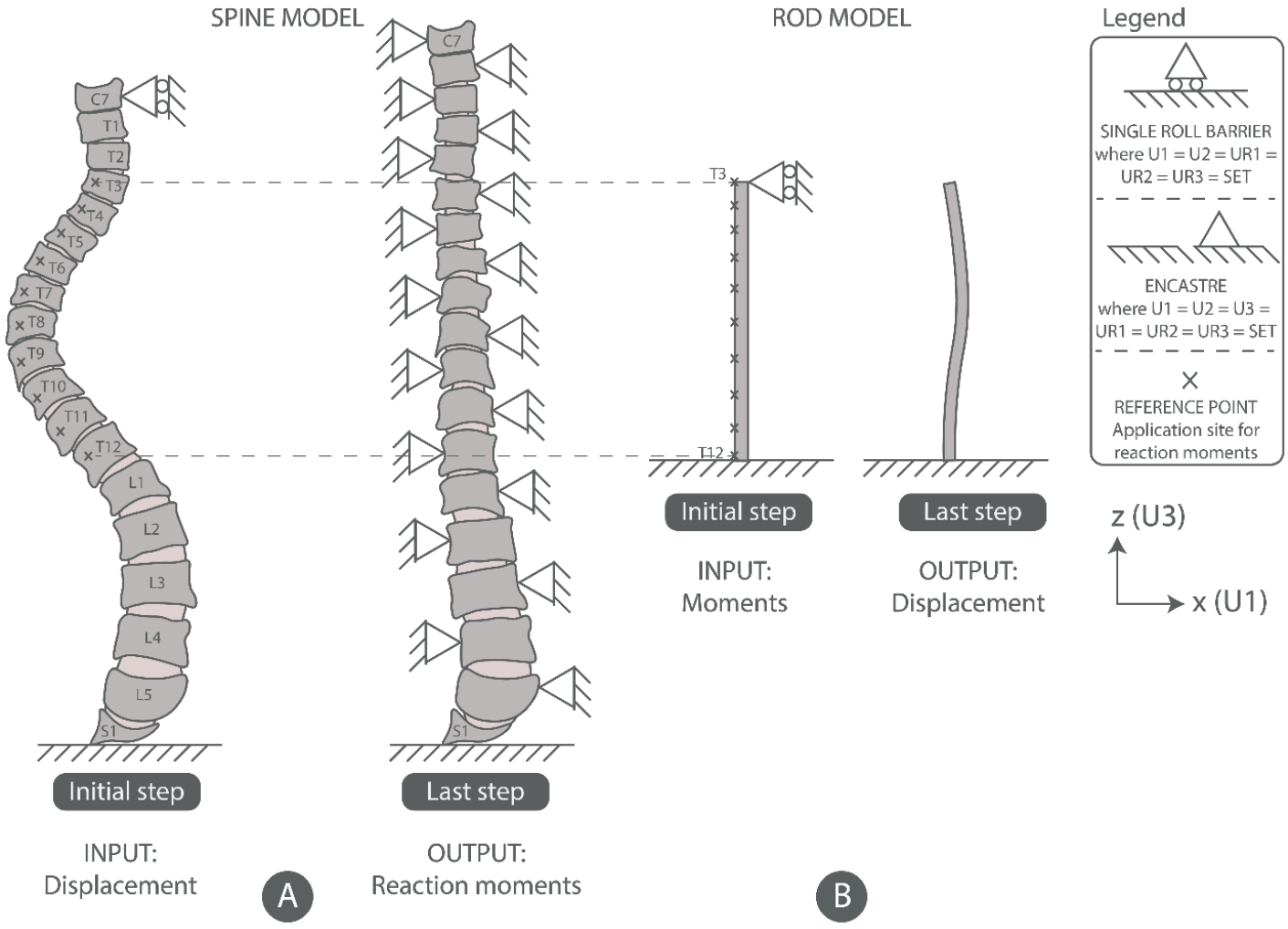


Figure 2.16: Application of the reference points (RP's) on the spine model to the rod model, where the z-coordinates are equal. (A) Initial and last step of the rod model. The RP's of the initial model and the reaction moment of the last step model are transferred to the initial step of the (B) rod model,

which results in deformation of the shape. The dotted lines show that the z-coordinates of the RP's on both the spine- and rod model match.

FEA of the rod model with these reaction moments will result in a deformed rod (*Figure 2.17B*) that in theory can be used in the spinal fusion procedure of the patient (*Figure 2.17A*).

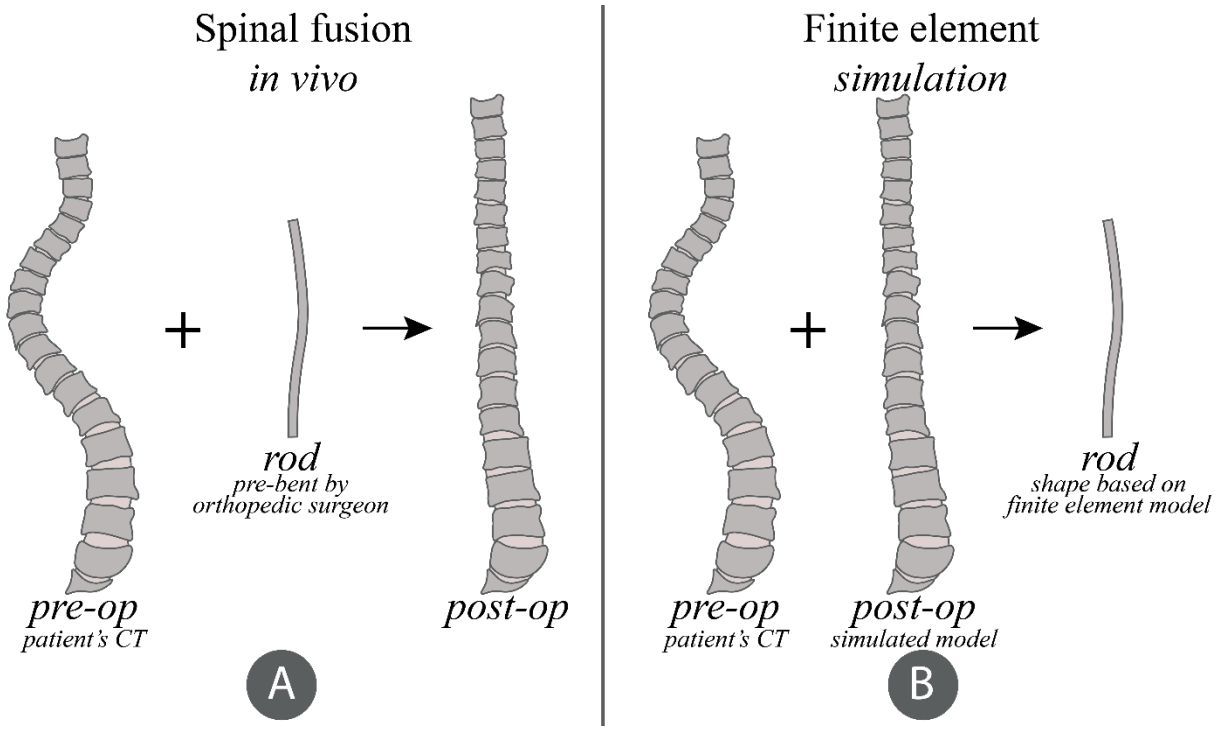


Figure 2.17: The shape of the rod is mainly responsible for the post-operative shape of the spine. (A) Spinal fusion procedure, where the initial spinal shape and a pre-bent rod determine the post-operative shape, (B) Finite element simulation, where the desired shape of the spine is formed, whereafter an accessory rod is built

2.11. Spine model comparison

FEA comparison with articles was performed to give an idea of the validity of the model. Model comparison is important to decide to what extent the model reflects real world results. Most spinal FEM studies are focused on applied motion in different directions. The model in this study was therefore compared with;

1. Stresses in the IVD's reaction to different movements (Figure 2.18);
    a. Flexion and extension (positive and negative moment along x-axis)
    b. Left- and right lateral bending (positive and negative moment along y-axis)
    c. Left-and right axial rotation (positive and negative moment along z-axis)
2. Range of motion

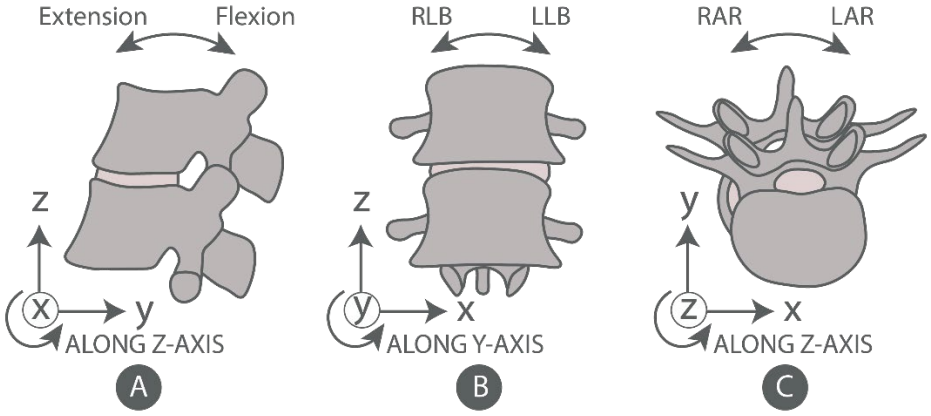


Figure 2.18: Six different movements (A) Flexion and extension, (B) Right lateral bending (RLB) and left lateral bending (LLB), (C) Right axial rotation (RAR) and left axial rotation (LAR)

To do so, the FE spine model that was used in this study was adapted slightly. The changes were the removal of the roller barrier boundary condition placed on the most superior vertebra (C7) and placement of moments on the superior surface of C7. The boundary conditions used in this model are comparable with the ones used by Xiao et al, except for the roller barrier since they only modeled two segments [15].

Different motions

Moments in different directions were applied to mimic motions that can be used for model comparison. Unlike forces, moments cannot be applied on a single element or node, due to lack of rotational degree of freedom. Therefore, a reference point (RP) was created in the middle of the surface that functioned as point of engagement, as shown in Figure 2.19*A*. The entire superior surface of vertebra C7 was connected with the RP through a kinematic coupling constraint so that all nodes of that surface were able to react on the applied moment.

The magnitude of each moment was chosen to be 10 Nm, the same value used by Zhang et al. The model was run six times, each time the direction of the moment was adapted as shown in Figure 2.19*B*. The chosen movements are three types of motions in opposite directions, according to *Table 2.3*.

Table 2.3: Applied moment in different directions

	Along x-axis	Along y-axis	Along z-axis
Flexion	10.000 Nm	-	-
Extension	-10.000 Nm	-	-
Left lateral bending (LLB)	-	10.000 Nm	
Right lateral bending (RLB)	-	-10.000 Nm	
Left axial rotation (LAR)	-	-	10.000 Nm
Right axial rotation (RAR)	-	-	-10.000 Nm

In addition, an axial load was applied of 300 N, based on the patient's body weight. The bodyweight was estimated to 45 kg [32], given that the patient was presumably adolescent [4], [26]. The axial force was obtained using the formula as used by Zhang et al [18]:

$$F_{axial} = \frac{2}{3} * bodyweight * 10$$

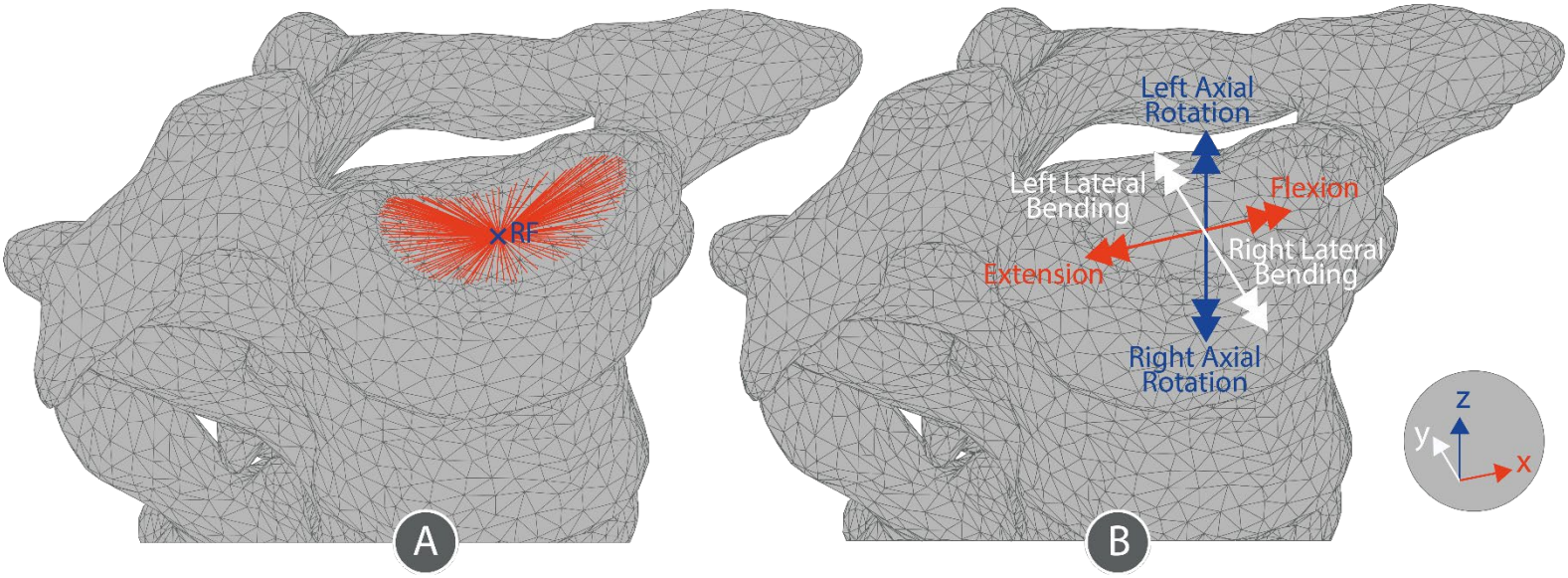


Figure 2.19: Coupling constraint with (A) Centroid reference point on C7 vertebra that is coupled with the superior surface and (B) applied moments in different directions, where the double arrows indicate a moment along the accessory axis

Range of motion

The range of motion (RoM) of AIS patients is generally lower than in patients without AIS [18]. Low RoM values means the spine is relatively stiff, which is not desirable. Therefore, a pre-operational measurement of RoM can give insight on the current flexibility of the spine.

The RoM was calculated by measuring the angle between two nodes before and after analysis, see *Figure 2.21*. Each time a different plane and accessory coordinates were used to determine angle α. By dividing the scalar products $\overrightarrow{AB}$ and $\overrightarrow{CD}$ by the magnitude of the vectors $|\overrightarrow{AB}|$ and $|\overrightarrow{CD}|$, angle α, or ROM, was calculated [33]. The MATLAB code including coordinates can be found in Appendix H.

$$\alpha = \text{acos}\left(\frac{\overrightarrow{AB} * \overrightarrow{CD}}{|\overrightarrow{AB}| * |\overrightarrow{CD}|}\right) * \frac{\pi}{180}$$

With $AB = \binom{b1}{b2} - \binom{a1}{a2}$ and CD = $\binom{d1}{d2} - \binom{c1}{c2}$. With 1, 2 being the accessory x, y, or z coordinate for that motion. For example, in Figure 3.15, $a1$ is the a_y coordinate and $b2$ represents the b_z coordinate in this zy plane.

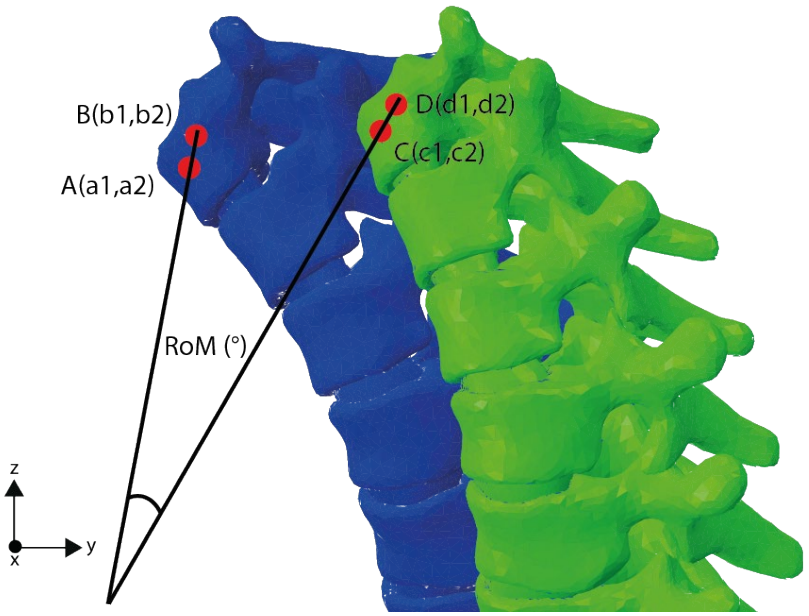


Figure 2.20: Range of motion measurement of flexion model in zy plane. The green model is the pre-analysis model and the blue one is the post-analysis model

2.12. Automation

Creating and running the model in Abaqus was completely automated. This means that, - when using the same input (.inp) files with accessory names, - this script can be applied to other spine models. Automation of FEA saves time and improves repeatability and reliability. Furthermore, parameters can be adjusted with more easily than creating a model by hand. For example, with this script, the material properties or loading conditions can be adjusted by simply changing one line of code. Workflows of the spine model and placement of the RP's can be found in Appendix G. All code used for the FE models can be found in Appendix I.

3. Results

This chapter elaborates the results of the spine- and rod model. *Figure 3.1* gives an overview of the paragraphs that will be discussed.

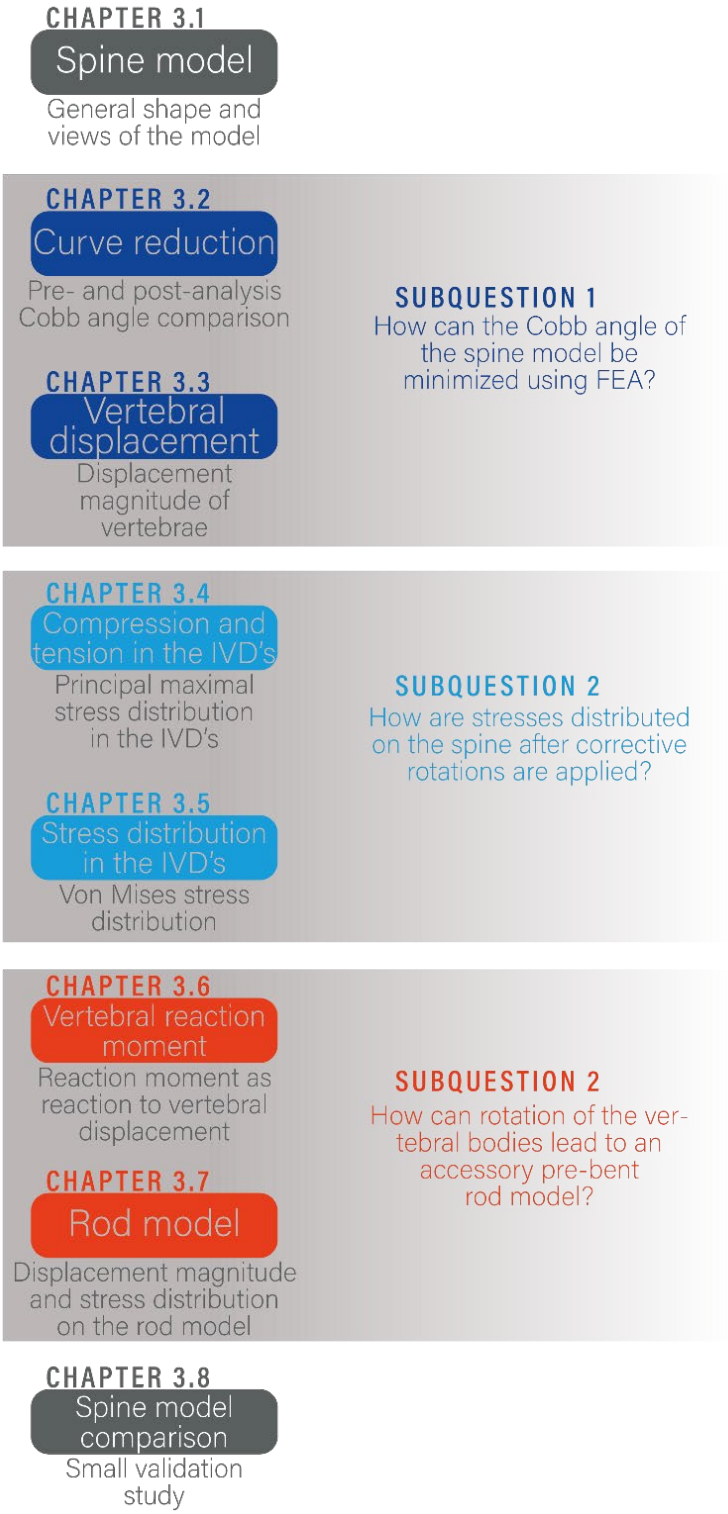


Figure 3.1: Simplified overview of the results. The sub questions from the research objective are assigned to accessory chapters. The dark blue chapters describe the overall curvature correction. The bright blue chapters describe the effect of the correction on the intervertebral discs. The orange chapters line out how the rod model reacted on the applied reaction moments.

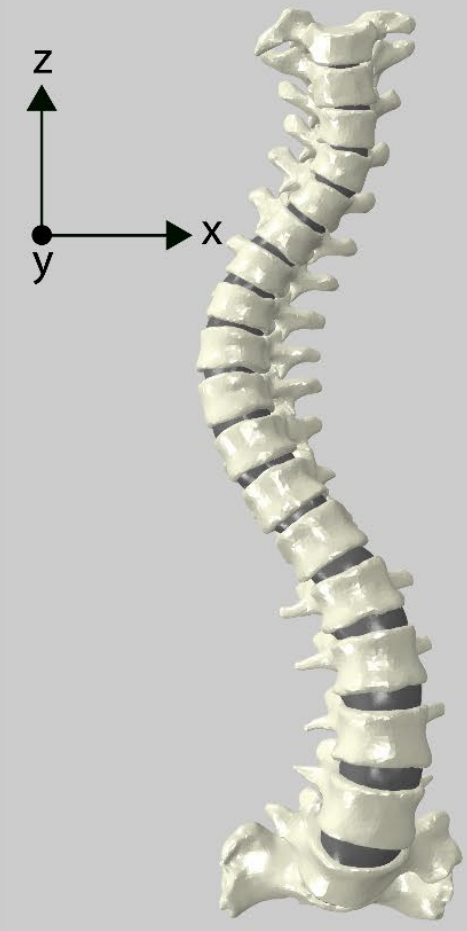
3.1. The spine model

The spine model includes the nineteen vertebral segments C7-S1 with the accessory eighteen IVD's (C7T1-L5S1). The latter are divided in two models: eighteen NP's and eighteen AF's. The assembly consists of 55 individual instances that has a total volume of 573943.44 mm^2, 316733 linear tetrahedral elements of type C3D4 and 97458 nodes. Information about the spine model can be found in *Table 3.1*. The number of sets and surfaces in the tables show the importance of automation of such models. Manually adding such amounts enlarges the change on both human and computational errors. The entire assembly of the spine model is visualized in *Figure 3.2* on the following page.

Table 3.1: Model specifications of the FE spine model in Abaqus. Sets, surfaces and constraints can be overlapping between vertebrae, AF and NP and are therefore not written down individually.

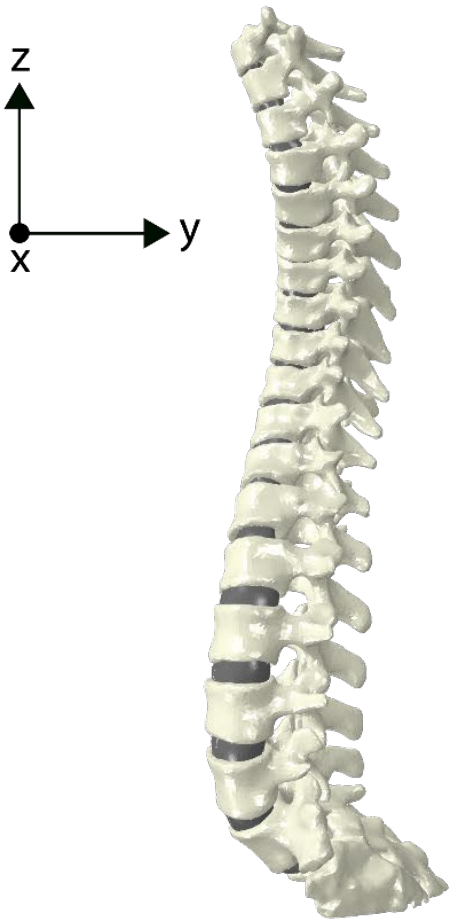
	Vertebrae	Annulus fibro-sus	Nucleus pulpo-sus	Total
Instances	19	18	18	55
Sets	-	-	-	437
Surfaces	-	-	-	218
Constraints	-	-	-	37
Volume [mm^3]	531884.88	26310.71	15747.83	573943.44
Nodes	73276	16968	7214	97458
Elements	252013	45108	19612	316733

Coronal plane
Anterior view



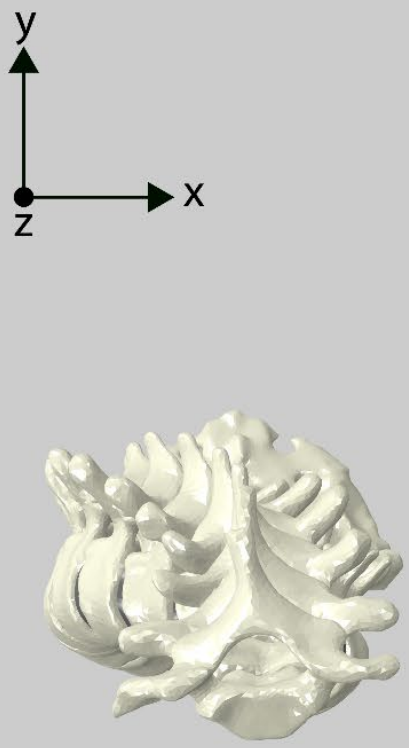
Right --- Left

Sagittal plane
Lateral right view



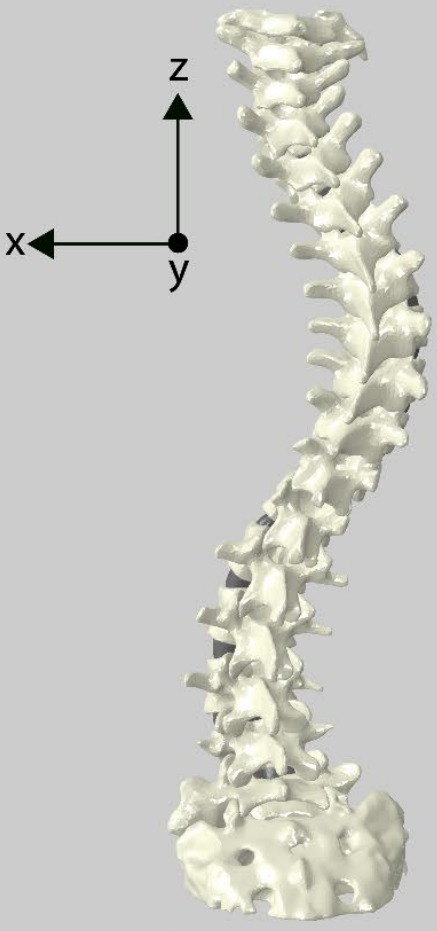
Anterior --- Posterior

Transverse plane
Superior view

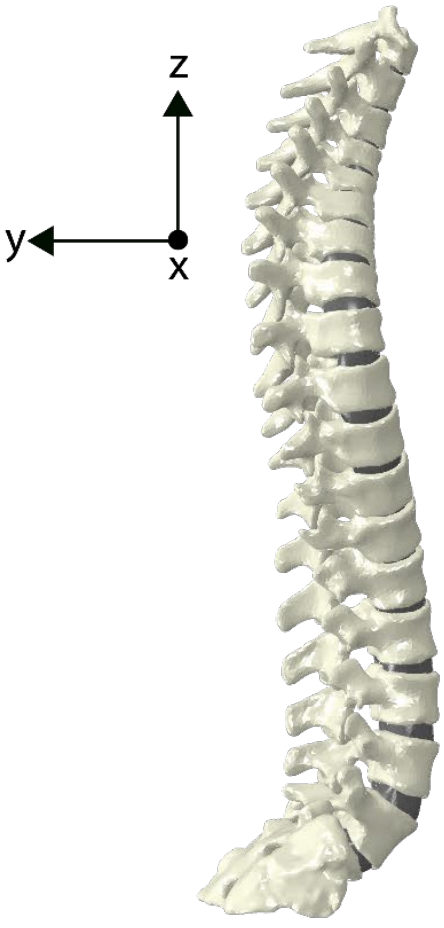


Right --- Left

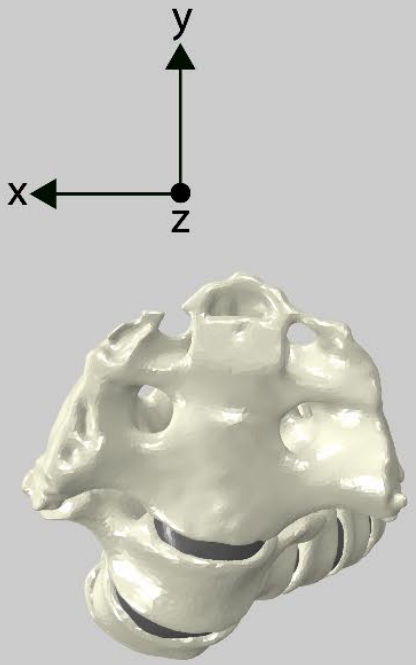
Posterior view



Lateral left view



Inferior view



3.2. Curve reduction

The Cobb angle of the spine declined after analysis, but a perfectly straight spine was not achieved. The initial Cobb angle of 58° was determined by Swedish orthopedic surgeons. The Cobb angle after FEM was estimated to be 39°, as shown in *Figure 3.4*. This means the magnitude of the curvature was reduced by 19° and a curve reduction of 33% was accomplished.

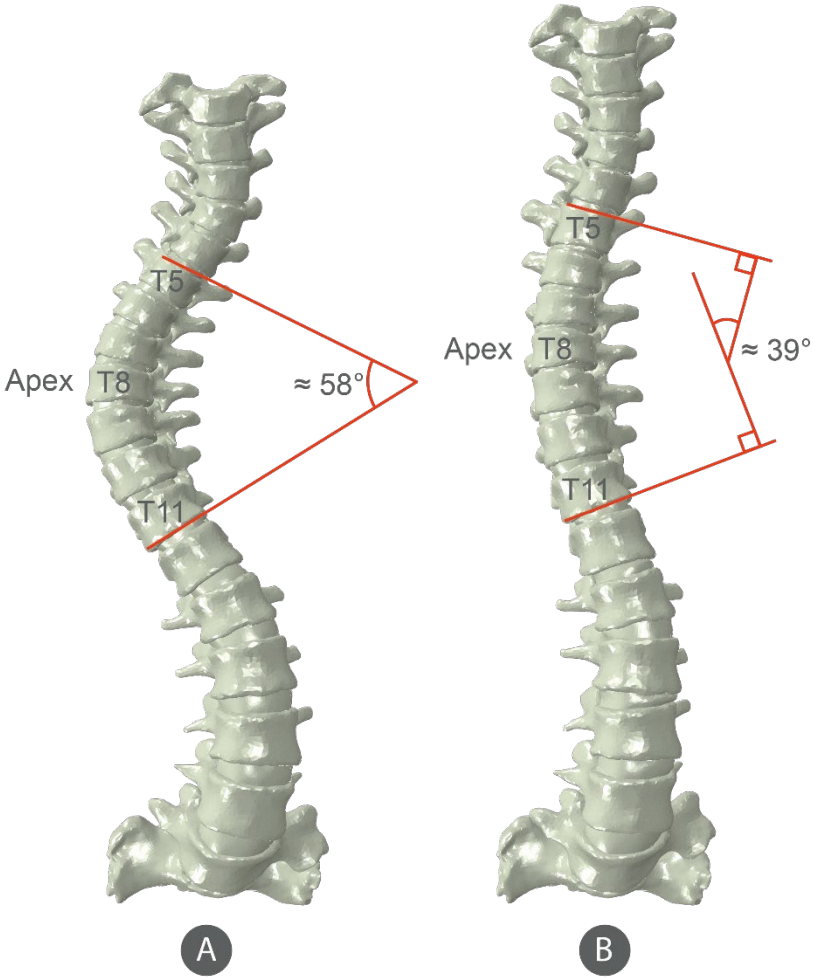


Figure 3.4: Curve reduction after analysis, (A) Initial Cobb angle of the thoracic curvature, (B) Post-analysis Cobb angle of the thoracic curvature

3.3. Vertebral displacement

Figure 3.5 shows the displacement magnitude of all nodes after analysis. The spine was elongated with a length of 1.385E+02 mm.

The overall magnitude (*Figure 3.5B*) shows the largest nodal displacement occurs in the superior vertebrae C7-T3 and around the apex at T8 and adjacent vertebrae. The upper segments only show displacement in the u3 or z direction, also as result of vertebral rotation and the roller barrier applied to C7.

There is hardly any displacement visible in *Figure 3.5D* because the applied rotations were focused on the zx-, or coronal plane and not in the zy- or sagittal plane. Although almost every vertebra was rotated along the y and/or z axis, hardly any displacement is found in the lumbar region. During the vertebral rotation procedure, vertebrae were only rotated along the zy-axis and not translated, which means the elongation in the u3 or z direction is merely a result of the applied rotations.

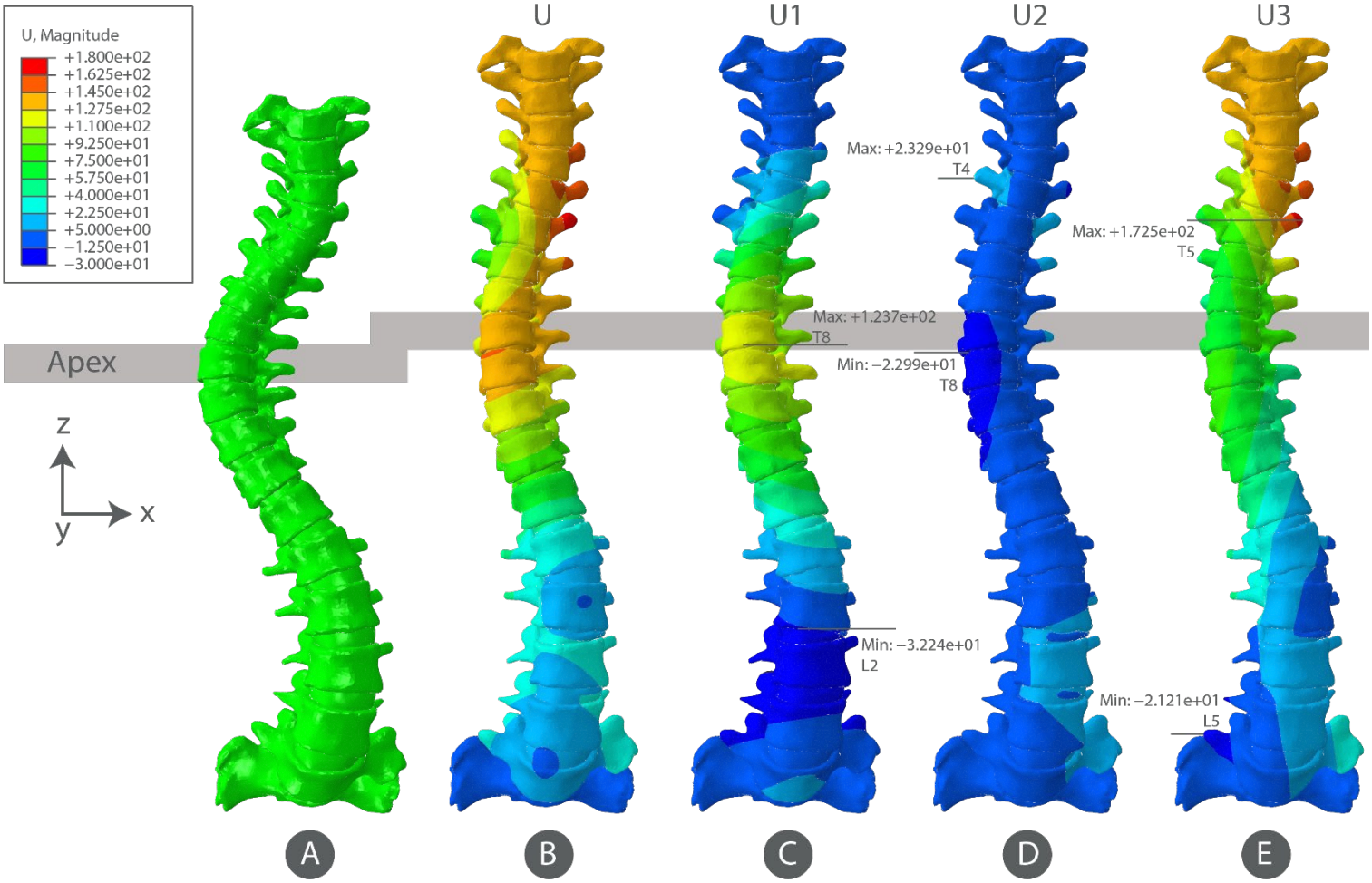


Figure 3.5: Displacement magnitude of the model after a displacement-controlled analysis with (A) initial shape, (B) overall displacement magnitude, (C) in u1 or x direction, (D) in u2 or y direction, (E) in u3 or z direction.

3.4. Compression and tension in the IVD's

The absolute principal stress represents a combination of normal forces that is initiated by the pushing interaction contact of the IVD's and the shear stress that is induced by sliding of IVD's over one another. The principal stress can be negative, indicating compression (blue) or positive, indicating tension (red) as represented in *Figure 3.6.*

The concave curves of the AF's are red colored and indicate tension. This is a result of the correction forces. The principal stress peak of over 45 MPa is found in the convex curvature at the thoracic region, around the apex. The concave part of the IVD's is stretched as a reaction to correction of the spine, whereas the convex side is compressed to compensate for the deformation as shown. IVD T4T5 contains both the AF with lowest stress and NP with highest stress.

The principal stresses of the NP's are relatively low in comparison with the AF's. Therefore, the scale for the NP's was adjusted to gain a better view on internal pressure in *Figure 3.7.*

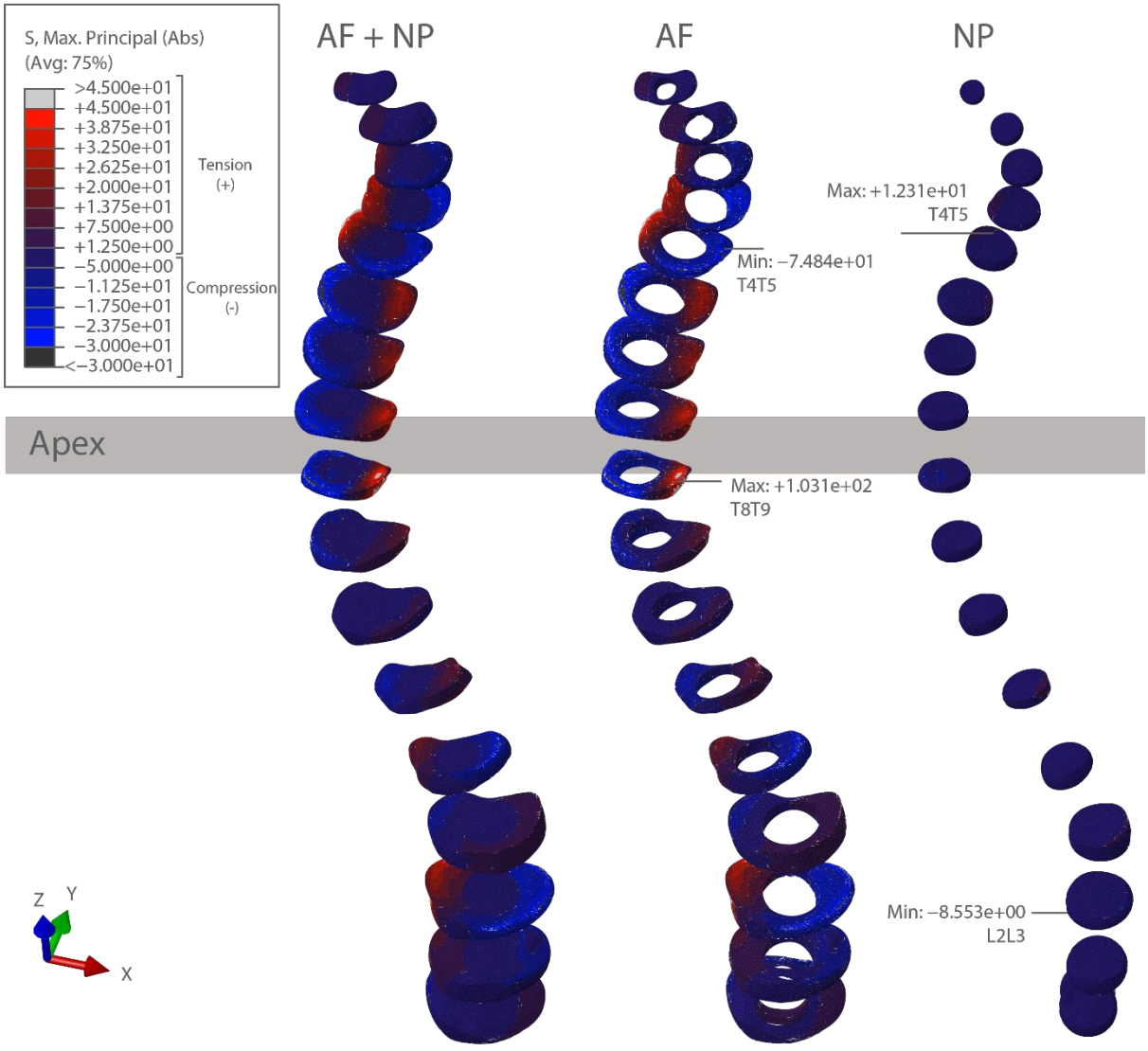


Figure 3.6: Principal stress in the IVD's

The compression and tension in the NP's as shown in Figure 3.7 are somewhat similar to the stress distribution in the AF's in *Figure 3.6.* The NP's have a Poisson's ratio of nearly 0.5 (*Table 2.1*), which means the volume stays constant. The internal pressure shifts from the convex- to the concave curvature.

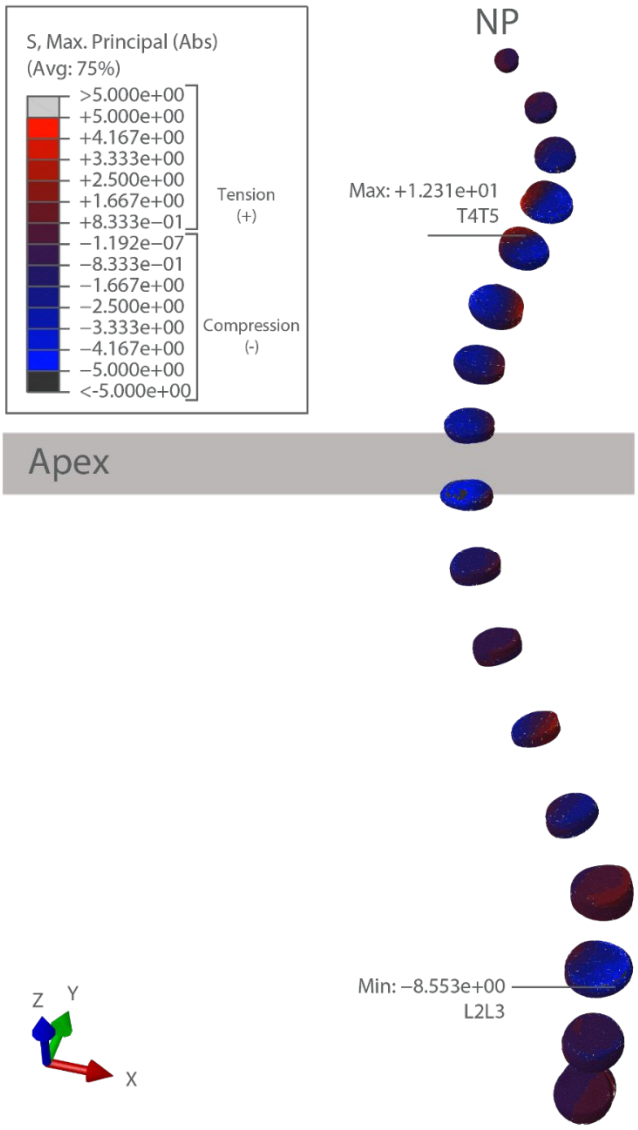


Figure 3.7: Principal stress in the NP's

3.5. Stress distribution in the IVD's

Figure 3.8 shows that the stress distribution in all NP's is very low. The AF's show highest stress distributions in the thoracic region. The lateral sides of each AF show higher stress peaks than the medially sides. The stresses found in the L2L3 AF remarkably high in comparison with adjacent IVD's. *Figure 3.8A* shows the L2L3 IVD lies on the transition from the main thoracic curvature to the thoracolumbar curvature.

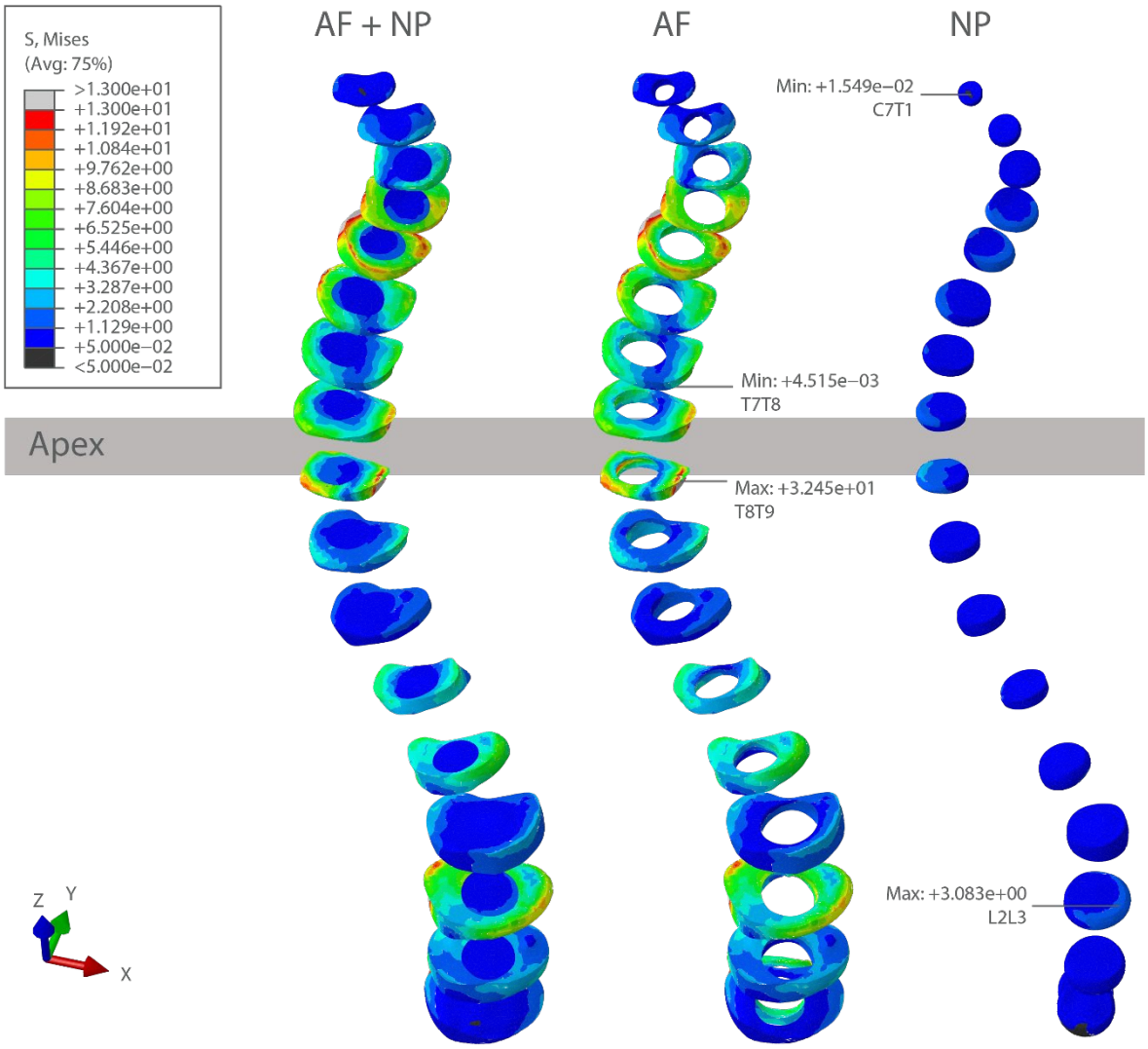


Figure 3.8: Von Mises stress in the IVD's

3.6. Vertebral reaction moments

Reaction moments were requested as history output for each reference point (RP). The value of the reaction moments at the last step (step 32) was used as input value for the new rod model. In the last step, all vertebrae were fixed in their current position to retain static equilibrium for the spinal shape at t=32 as shown in *Figure 3.9.* Remember that the steps described in the method were subsequent rotations instead of time measurements. There were two directions in which rotations were applied, thus also two directions along reaction moments were found; reaction moments along the y-axis and reaction moments along the z-axis. These reaction moments are presented in graphs (*Figure 3.9* and *Figure 3.10*), with on the horizontal axis the subsequent rotations. Higher segments are rotated in earlier rotations than the lower segments, for example; T3 is rotated in step 2 (rotation along y-axis) and 3 (rotation along z-axis) and L3 is rotated in step 27 (along y-axis) and 28 (along z-axis). The vertical axis represents the magnitude of the moment in Nm.

Reaction moment along y-axis

Figure 3.9 is a graph with the reaction moments along the y-axis. Right to the graph, the accessory motion is shown. Lower segments (~T12-L4 around step 25-32)) result in higher reaction moments than higher segments (thoracic region). Vertebrae C7, T1, T4, T6 and T10 do not have a reaction moment along the y-axis, as no rotation was applied in this direction. The vertebrae C7 and L5 were initially restricted in rotational directions and therefore do have a reaction moment. Vertebra L2 has a very high peak of 68 Nm in the positive direction, whereas all the other lumbar vertebra have negative reaction moments.

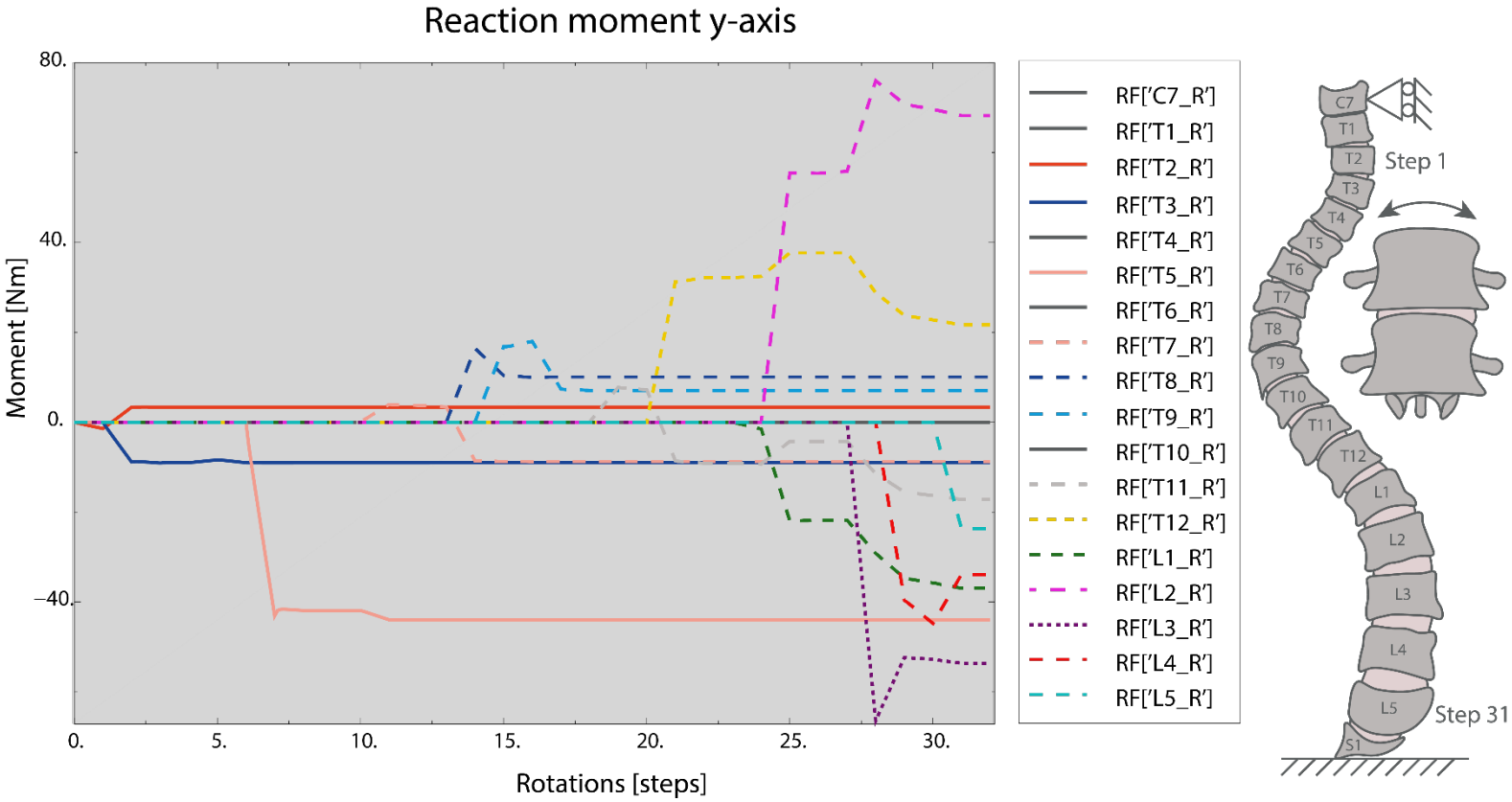


Figure 3.9: Reaction moments along the y-axis

Reaction moment along z-axis

Figure 3.10 shows the moments along the z-axis. The axial rotation motion is displayed right to the graph. C7, T1, T2 and L5 do not have a reaction moment along the z-axis, as no rotation was applied in this direction. In constrast to the reaction moment in y-axis in *Figure 3.9*, *Figure 3.10* shows the higher segments (~T3-T8) have higher reaction moments along the z-axis. T5 has the highest moment of almost 10 Nm. A low negative moment was found in T4 of almost -15 Nm. The values of the reaction moments that will be used as input parameters of the rod model at the last timestep (t=32) along the y- are displayed in Table 3.2.

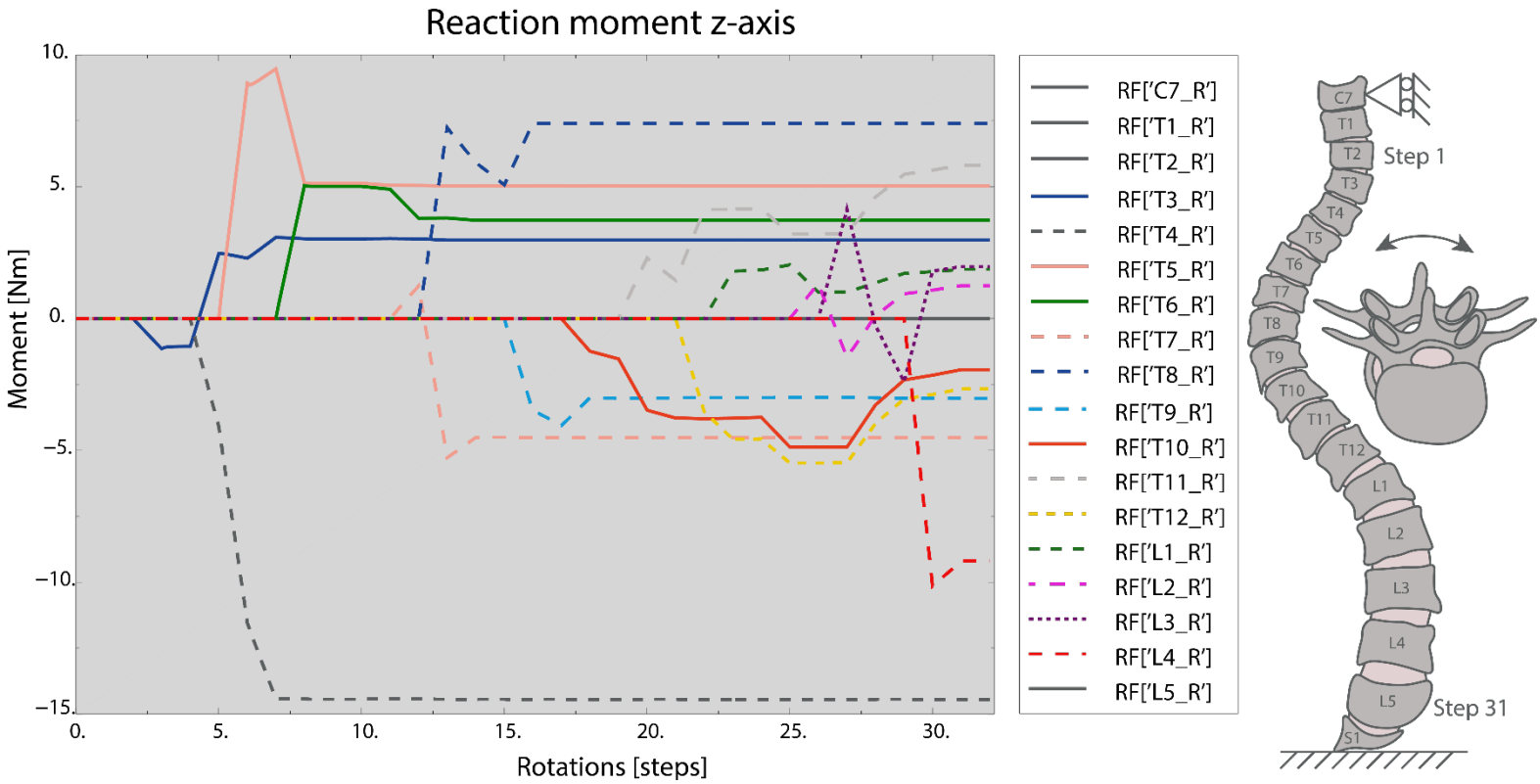


Figure 3.10: Reaction moments along the z-axis

Table 3.2: Moments at vertebral levels T3-T12

Vertebral level	Moment along y-axis at $t = 32$ [Nmm]	Moment along z-axis at $t = 32$ [Nmm]
T3	-9008,12	2974,593
T4	0	-14439,4
T5	-44000,9	5028,173
T6	0	3737,016
T7	-8749,82	-4506,57
T8	10097,31	7404,022
T9	7019,351	-3027
T10	0	-1937,91
T11	-17206,7	5817,062
T12	21659,62	-2670,64

3.7. Rod

Figure 3.12 visualizes the output results of Cobalt chromium (CoCr) and Titanium (Ti) rods after application of the reaction moments along the y-axis. The stress distribution in both rods is comparable and located at the same regions. The main displacement was found in the U1, - or x-direction. A small displacement is seen in the U3, - or z-direction. The location of the spinal apex (T8) is not corresponding with the apex of the rod model (T6-T7). The general shape of the rod is interesting, as the overall curvature of the rod is mirrored with respect to the curvature of the initial spine shape. This is expected as the rod has to bend the spine to a straight position.

Table 3.3: Maximum values with rod diameter of 6.0 mm

	CoCr	Ti
Max Von Mises stress [MPa]	9147	8904
Max displacement magnitude [mm]	4.213	8.130
Computation time [s]	517	648

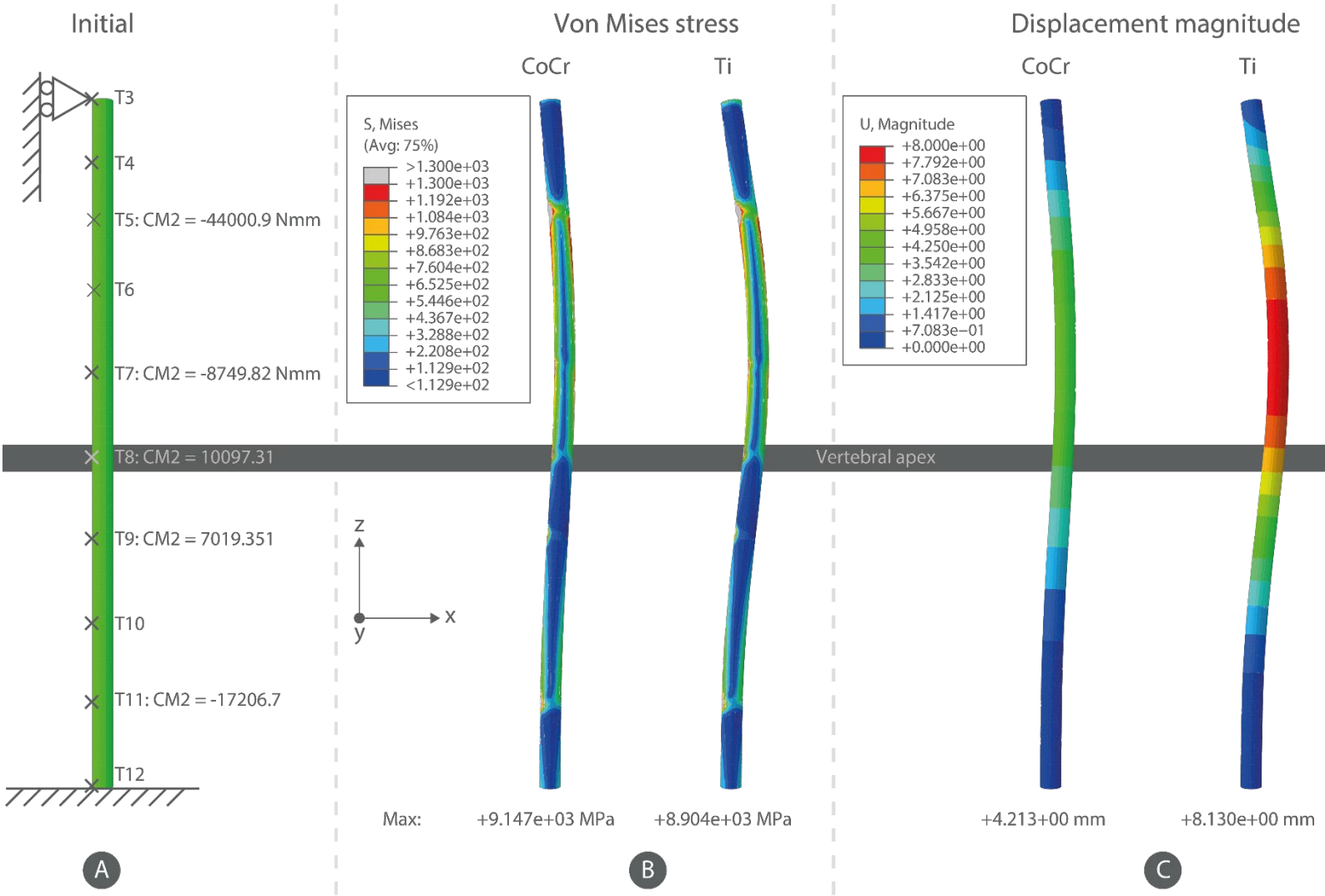


Figure 3.11: Rod model, (A) Initial shape with boundary conditions, (B) The stresses on the CoCr and Ti rod after analysis, (C) the displacement magnitude on the CoCr and Ti rod after analysis.

If height of the rod in comparison with the spine is visualized in *Figure 3.12*. The rod would be placed within the segments T3-T12. The curvatures of both are mirrored. The young's modulus of the rod is much higher than the young's modulus of the IVD's. Hence the smaller curvature of the rod in comparison with the spine model. The apex of the rod is located somewhat higher than the apex of the thoracic curvature, as shown in *Figure 3.12*.

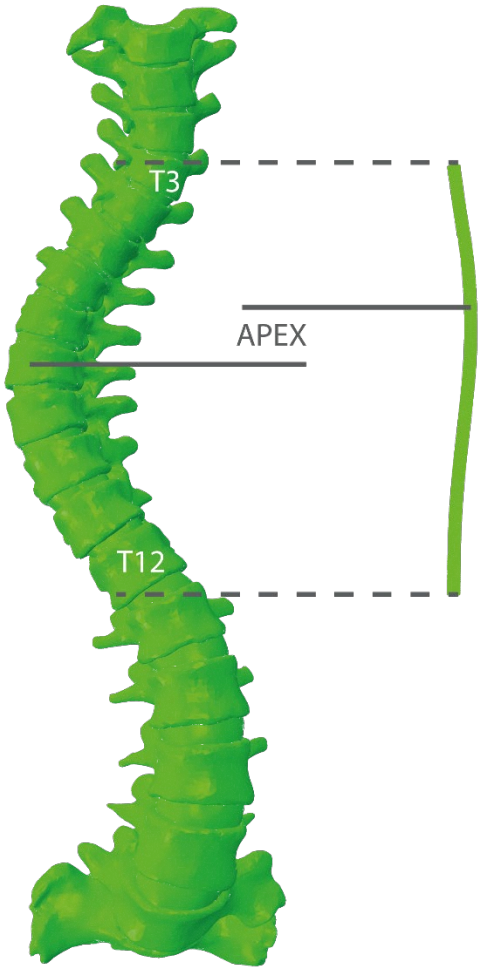


Figure 3.12: The magnitude of the rod in comparison with the spine. The titanium rod model was chosen for this comparison.

3.8. Spine model comparison

Figure 3.13 shows different types of motion in anterior- lateral and posterior view of the comparison study. The largest stress concentration is found in the upper section of each model under different loading conditions. The stress in intervertebral discs gradually decreases towards the lower segments of the spine.

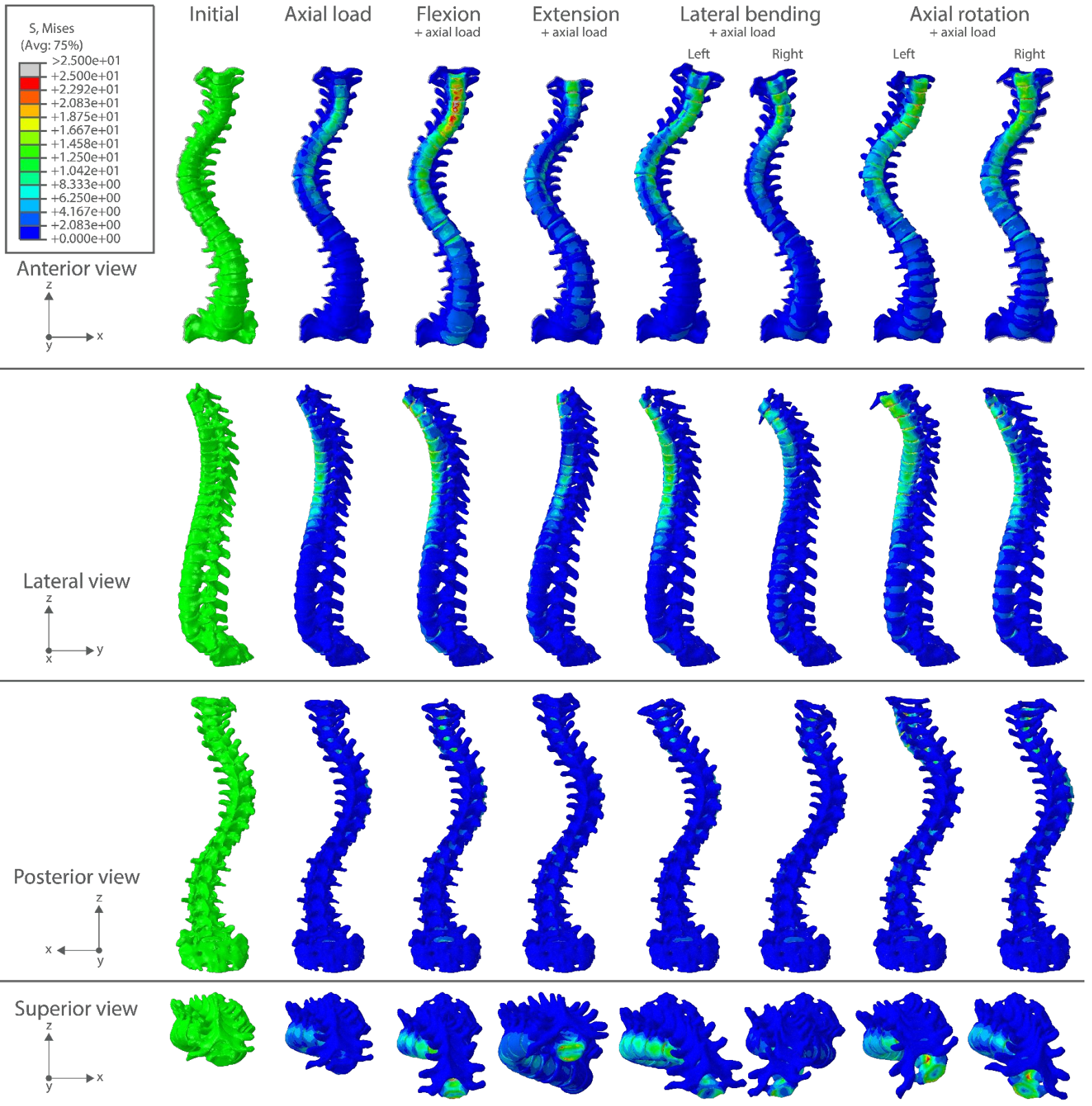


Figure 3.13: Frontal and lateral view of flexion, extension, lateral bending and axial rotation of the spine model

Extension results in lowest stress concentration and both axial rotation and flexion results in the largest stress distributions. Both right- and left axial rotation show higher Von Mises stresses on intervertebral discs, with a peak stress of around 6-8 MPa below the apex (*Figure 3.13*) and average IVD stresses around 1.8 MPa (*Figure 3.14*), whereas the intervertebral discs of other models below apex level show average Von Mises stresses of less than ~1.4 MPa, which is in accordance with the findings of Zhang et al (2021).

Of the lumbar IVD's L1L2, L2L3, L3L4 and L4L5, the average stress of all nodes per IVD was calculated, see Figure 3.14. All motions included an axial load of 300 N in negative direction. The L3L4 IVD was subjected to the highest stresses with an average of 1.36 MPa in all directions. Left- and right rotation were the motions that resulted in highest stress concentrations, respectively 1.75 MPa and 1.63 MPa.

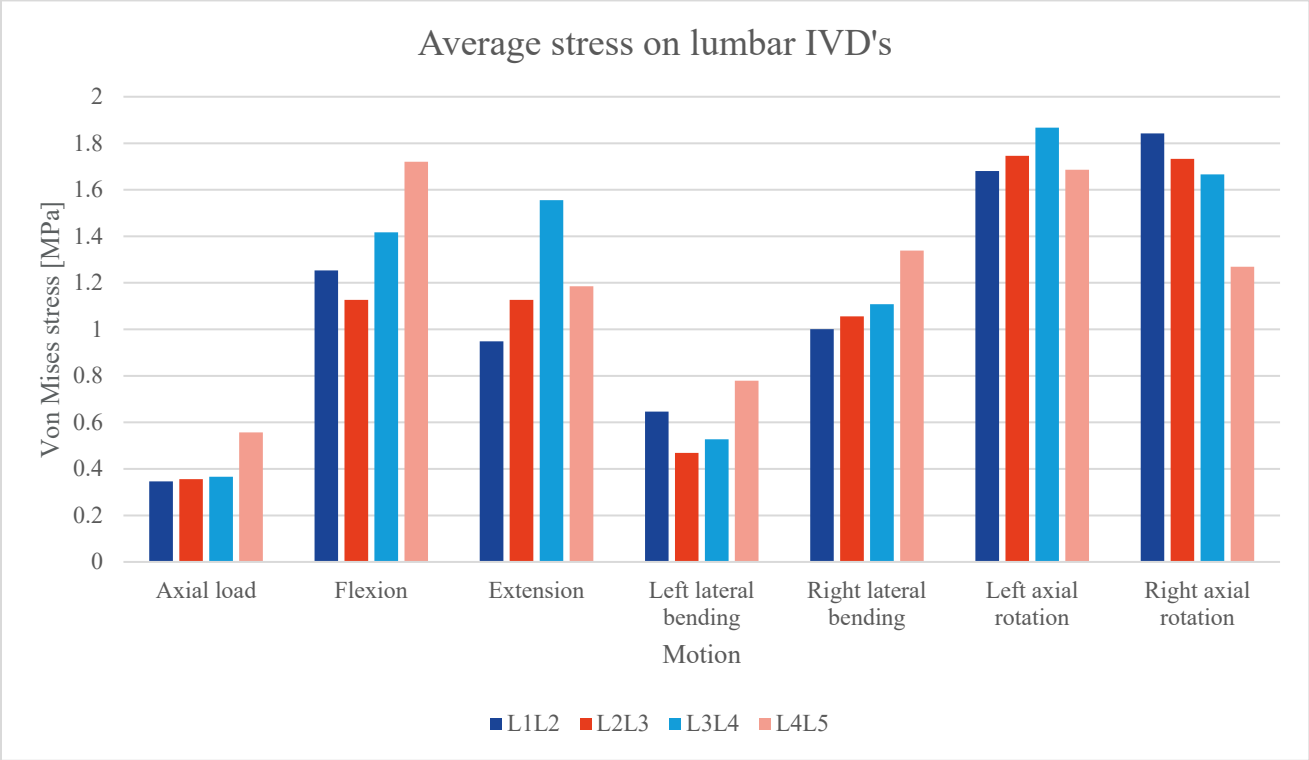


Figure 3.14: Average stress on intervertebral discs L1L2, L2L3, L3L4 and L4L5 in different motions

Table 3.4: Average stress values for IVD's in different motions, compared with the values found by Zhang et al.

		Flexion	*Extension*	*LLB*	*RLB*	*LAR*	*RAR*
L1L2	This study	1.25	0,948441	0,646026	1,000519	1,680807	1,842148
	Zhang et al. [18]	0.42	0.44	0.8	0.59	0.74	0.62
L2L3	This study	1,127005	1,126325	0,468821	1,056042	1,746022	1,733247
	Zhang et al.	0.34	0.28	0.5	0.43	0.63	0.67
L3L4	This study	1,417348	1,555269	0,527165	1,108071	1,867461	1,665857
	Zhang et al.	0.33	0.22	0.43	0.4	0.46	0.59
L4L5	This study	1,720833	1,185407	0,779064	1,338891	1,686393	1,269377
	Zhang et al.	0.41	0.17	0.34	0.33	0.43	0.57

The range of motion (RoM) was measured in all different directions and visualized in Figure 3.14. The ROM in the axial rotation was largest and the RoM for left lateral bending was the smallest. The values found are displayed in Table 3.5. The RoM in axial rotation is higher in this study compared to results of Zhang et al. Note that Zhang et al. only included five vertebral segments in his study (L1-L5). Application on the most superior vertebra results in a lower moment than done in this study, because the arm/distance d is lower, see *Figure 3.16.* When the distance d, - or moment arm -, increases, the moment M increases equally if the applied force F is the same, as shown *Figure 3.16.*

$$M = F * d$$

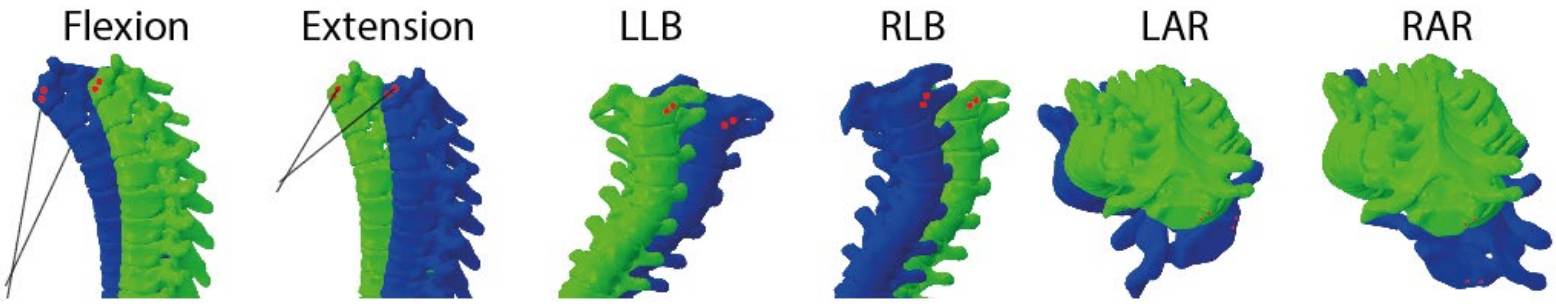


Figure 3.15: Range of motion calculation of flexion, extension, left- and right lateral bending (LLB and RLB), and left- and right axial rotation (LAR and RAR). The initial model is green and the blue model is after analysis.

Table 3.5: The range of motion of different directions. Comparison with measurements from Zhang et al. LLB = left lateral bending, RLB = right lateral bending, LAR = left axial rotation, RAL = right axial rotation

	Flexion [°]	Extension [°]	LLB [°]	RLB [°]	LAR [°]	RAR [°]
This study	18.4783	34.5552	14.1629	27.8371	37.0981	35.0418
Zhang et al. [18]	16.8	14.5	24.5	27.3	13.7	17.3

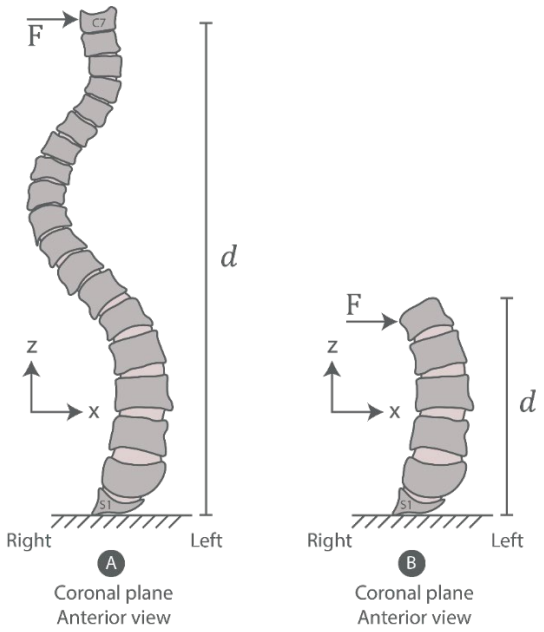


Figure 3.16: Different moments, (A) Model used in this thesis, (B) comparable studies that only use few segments

4. Discussion

This chapter discusses the results presented in the previous chapter. It reviews the geometry and quality of the spine model, the correction after analysis, discusses the rod model and elaborates on the comparison study.

4.1. Spine model

Geometry

In this paper, the FE geometry of the vertebrae is based on high resolution CT-imaging of an AIS patient and can therefore be assumed to be a realistic parameter to use for the geometry. Intervertebral discs (IVD's) are hardly detectable on CT scans and were for this reason created manually. The IVD's, that consist of a nucleus pulposus (NP) and annulus fibrosus (AF) were simplified according morphology described in articles [9]. The NP was given a spherical shape instead of the more realistic ellipse shape. This was computationally easier to manage, given that spherical primitives require less input parameters than ellipse shaped primitives. The shape of IVD's affect the flexibility of the spine. Therefore, the simplifications that were made are expected to effect the spinal flexibility and limit the reliability of the spine model [34].

Excluded tissue

The mobility of the spine is not determined by the bony vertebrae. Instead, the spinal flexibility is mainly regulated by the material properties and geometry of the IVD's, ligaments, and muscles [34], [35]. In the current spine model, the latter are not included, as shown in *Figure 4.1A*. All the mobility in this FEM model is ensured by the IVD's, which diminishes the reliability of the model. The human spine can be compared to a tensegrity structure, balanced by compressive forces. When relaxed, the spine is deformable and compliant, but by pulling the intervertebral ligaments together, the spine becomes stiff and prevents lateral bending to withstand high loads and external forces. Therefore, inclusion of muscles and ligaments into a FEM model such as shown in *Figure 4.1B* can provide far more realistic results [5].

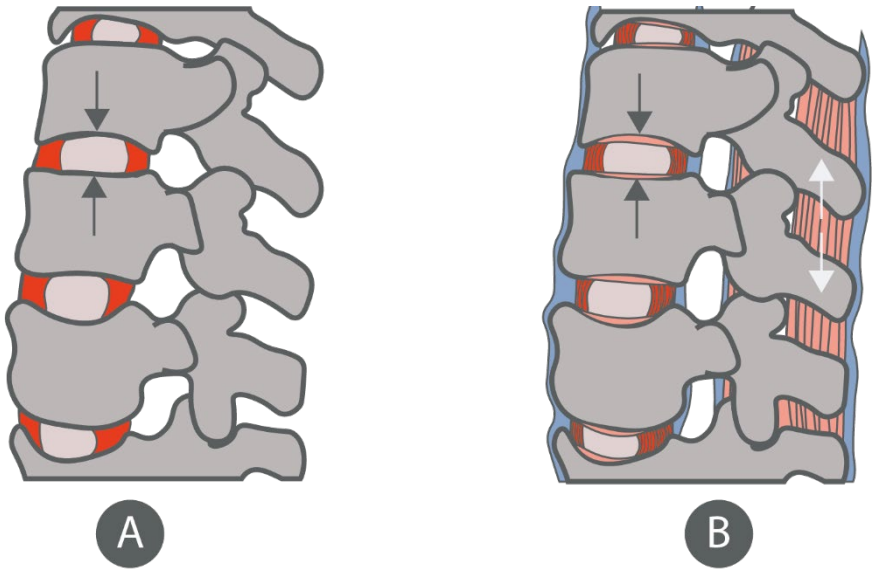


Figure 4.1: Intervertebral discs under compression and ligaments under tension (A) Simplified FEM model where the IVD's are compressed, (B) More realistic situation, where IVD's are under compression (grey arrows) and the ligaments under tension (white arrows). Names of the structures can be found in Figure 1.3

Nucleus location

The nucleus was modeled as center of gravity of the intervertebral disc, as visualized in *Figure 4.2*A. There is not much information available about the location of the nucleus pulposus in AIS patients, but Yeung et al. (2021) assumed that the nucleus can migrate to the opposite direction of highest compression force, which is the convex curve, shown in *Figure 4.2*B. The question remains as to whether migration of the nucleus pulposus towards the convex curve is an effect or a cause to enhance scoliotic behavior [35]. More research can lead to an answer to this question and possibly track down a feasible cause of AIS. Given that the location of the nucleus effects the spinal shape [36] and that the choice of the location of the nucleus in this study was not based on scientific evidence, decreases the accuracy of this study.

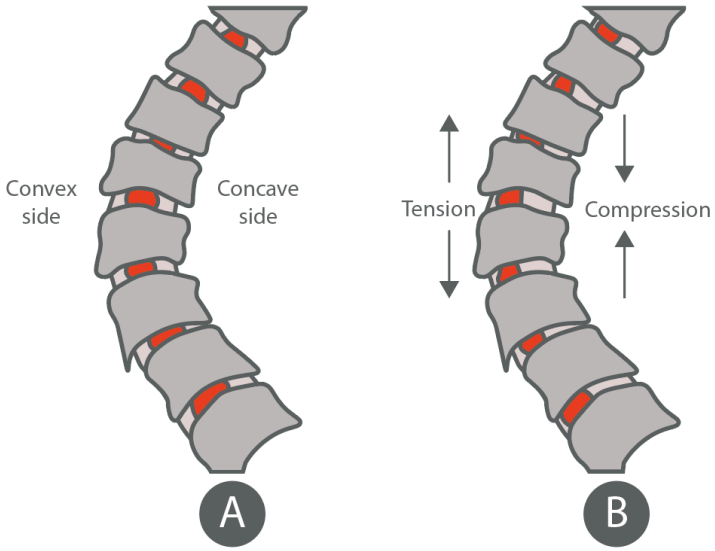


Figure 4.2: Location of nucleus pulposus with (A) the modelled spine and (B) a presumably more realistic situation where the nucleus is shifted to the convex side of the scoliotic curvature

Material properties

Due to the large number of parts and elements of the FE model, the material properties of the vertebrae and IVD's were chosen to be homogeneous isotropic and linear elastic. For the AF, a ground substance was used. Although poor estimations are likely when using an oversimplified FEM model, simple material properties were used to prevent large computational time. For future studies, hyper- or visco-elastic properties should be considered, as the gel-like and fiber structure of the NP and AF in reality show visco-elastic behavior [21] [9].

Material properties of individual structures such as the IVD's and vertebrae are often estimated values based on cadaver studies. As most of these studies use cadavers of elderly people [37], chances are the spinal structures are affected by old age ailments, such as disc degeneration, stiffening of the joints, and shortening of tendons. This results in measuring different material properties that are not always representative for adolescents. The AF material properties are based on the ground substance, instead of a collagen fiber composite. This results in an elastic modulus of the AF which falls well outside the range of moduli as experimentally determined by Skaggs et al. [38].

4.2. Correction

A curve reduction of 33% was accomplished in the FEA, resulting in a Cobb angle of 39°, that was initially 58° (*Figure 3.4*). Studies report a broad variety of different curve reductions [16], [17], [39], [40]. It is difficult to determine a proper correction rate, as there are many parameters that influence the success of the surgery procedure. A Cobb angle of 39° lies below the surgery threshold of 45° [1], [7]. It is difficult to estimate whether this percentage is a proper reduction. Some studies report much higher curve reductions after surgery, up to 63% for main thoracic curves [17], [39], others report similar curve reductions with a range of 18% to 36% for main thoracic curves [16], [40]. Although the threshold below 45° was reached, the patient was still left with a post-analysis Cobb angle of 39°. This means the patients still has quite a scoliotic curvature of moderate severity (Cobb angle range of 26°-45°). Due to time limitations within the context of the thesis, the correction was insufficient to decrease the Cobb angle to a mild scoliotic severity (Cobb angle range of 10°-25°) [41]. Optimization of the method and the automation process might result in higher curve reductions.

Every spine is different and not all surgery procedures are executed similarly. Articles that mention a correction rate do not always mention the metal type, rod diameter, curve flexibility, spinal length, Lenke type or included segments, which complicates comparison with this study [39], [40], [42]. Besides, orthopedic surgeons have their own set of instrument preferences and treatment methods.

Perfect correction is hard to accomplish, even in FEM models. *Figure 4.3* shows that by rotating the vertebrae in the most desirable position, the intervertebral space becomes unrealistically large. All vertebrae and intervertebral discs are interconnected, and do not allow large deformations if the flexibility of the spine is rather low. This FEM model was run with linear elastic material properties for the intervertebral discs. The use of more realistic material properties, for example visco-elastic properties, can lead to results with a higher reliability.

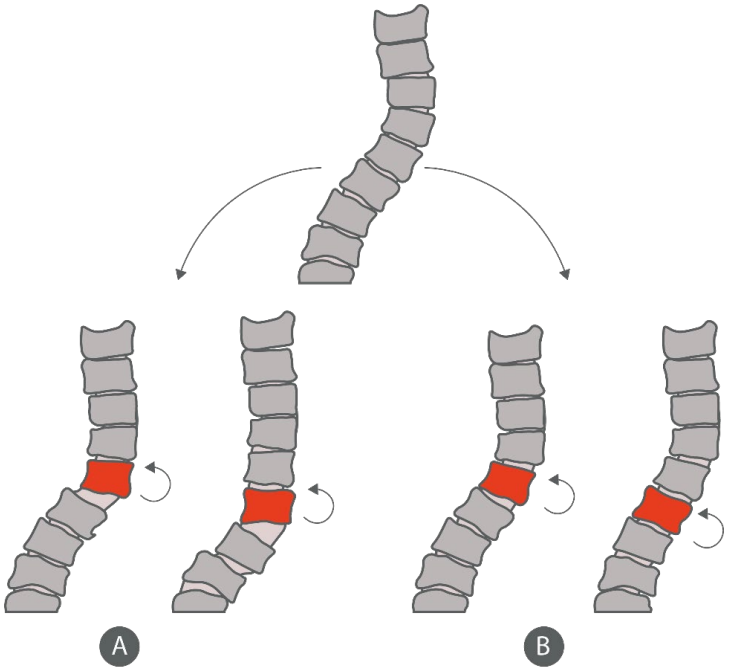


Figure 4.3: Vertebral rotation with (A) Ideal rotation where each vertebra is perfectly rotated, (B) Actual situation where the vertebra cannot be rotated optimally due to tissue limitations.

4.3. Output parameters

Vertebral displacement

The post analysis shape of the spine model was analyzed in three different directions u1 (x), u2 (y) and u3 (y). The displacement magnitude U in *Figure 3.5A* is a combination of a displacement in the u1 or x direction of around 140 mm and a smaller displacement in the u3 or z direction around 90 mm that can be explained by elongation of the spine as result of vertebral rotation. *Figure 3.5C* shows the direction magnitude is most extant in the vertebrae ≈T3-T12, which is in accordance with the pre-operative surgery plan of the orthopedic surgeon as shown in *Figure 2.9*. Thus, the chosen segments of the thoracic region that were to be included in surgery result in the highest displacement magnitudes. Although the lumbar region was corrected as well in the FE simulation, their displacement magnitude is relatively small. To leave the patient with enough flexibility, the spinal rod should not include a larger region than recommended by the orthopedic surgeon.

Compression and tension in the IVD's

Correction of the spinal shape by rotating vertebrae leads to pressure in the IVD's on the concave side of the curve because of redirected force distribution of the vertebrae. Higher pressure leads to compression, meaning the density of the IVD increases and the volume decreases. The pressure is highest in the convex curvature where the IVD's are pushed to one another. *Figure 4.4* visualizes this phenomenon. The rescaled NP's in *Figure 3.7* show equal compression-tension distribution as in the AF in *Figure 3.6.* Compression in the NP's can eventually lead to a shift towards the concave curvature side, as shown in *Figure 4.2*, to distribute the gravitational forces more evenly.

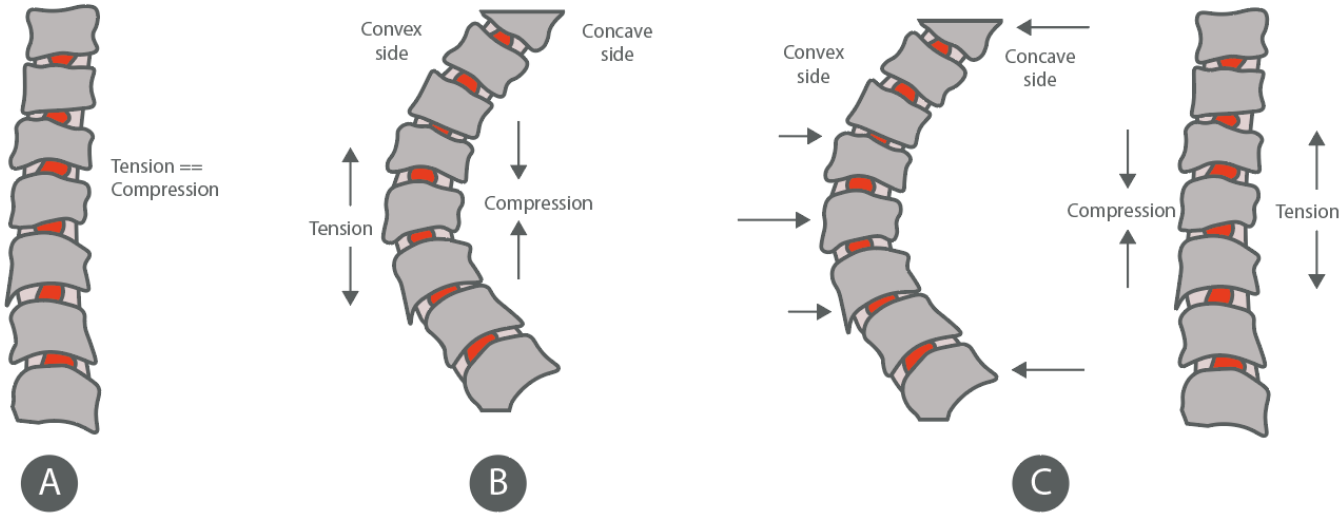


Figure 4.4: Compression and Tension, (A) Ideal situation where tension and compression in the IVD are equal, (B) Scoliotic spine where the concave side of IVD's is constantly under compression and the convex side under tension and this becomes the new normal, (C) Correction of the spine where compression and tension are reversed due to correctional forces.

Stress in the IVD's

Low stress distribution in NP's in comparison with the AF's in *Figure 3.8* can be explained by the lower young's modulus the nuclei have in comparison with the enclosing annuli. As most correction is performed in the thoracic area, the largest stresses in the AF's are also found here. As the applied rotations on the vertebrae were along the x-axis, most pressure was administered to the sides. The lumbar region required less correction as it slightly followed the displacement of the superior region. IVD L2L3 is subjected to a relatively large stress because it is located in between the two curvatures. This is probably the case because the superior and inferior vertebrae are both rotated in a different direction.

Vertebral reaction moments

Because the input was mainly applied moments along the y- and z-axis, the reaction moments are also found along the same axes. Note that only the reaction moments on the y-axis were used as input for the rod model, because moments along the z-axis do not affect the shape of the rod, but only lead to torsion, which is not desirable.

Vertebra L2 has a very high moment of 68 Nm in the y-direction, as displayed in *Figure 3.9* to compensate for the negative moments in L3 and L4. This vertebra lies directly in between the thoracic curvature and the thoracolumbar curvature and therefore is expected to show a higher reaction moment.

The segments between T3 and T8 show higher reaction moments along the z-axis, possibly due to the larger Cobb angle at the curvature. Vertebral rotation is said to induce scoliosis. Regions with a higher Cobb angle (thoracic region) are prone to more axial rotation [35].

4.4. Rod model

Pre-operative prediction of the shape of the rod used in surgery has various advantages. By pre-operatively determining the shape of the metal rod used in spinal fusion, the surgical duration can be reduced. Also, by computing the optimal shape of the rod, the quality of surgical outcome can be improved.

The shape of the current rod model was only based on the zy plane. The ultimate shape in the zx plane was not determined. Due to lack of knowledge on the exact shape models of the entire lateral spine curvature, it was hard to determine an ideal shape in the zx plane.

More knowledge on the optimal average shape of the entire spine (cervical, thoracal and lumbar segments), would give more information on how to deform an AIS spine in the lateral plane and can be useful to predict rod shapes in all directions.

The displacement magnitude of 8 mm as presented in *Table 3.3* found in the Titanium rod is twice as high as the displacement found in the Cobalt chromium rod, because the E-modulus of CoCr is approximately twice as high as the E-modulus of Titanium.

The location of the apex of the Rod model is not equal to the apex of the Spine model, as can be seen in *Figure 3.12*. The reason for that may be that all connecting vertebral bodies exert a force as reaction to the movement away from the apex.

Higher reliability of the spine model leads to higher reliability of the rod model. An improved spine model that includes visco-elastic material properties, inclusion of ligaments and muscles will provide reaction moments that are closer to nature and will result in a more accurate rod model.

4.5. Comparison study

Comparison of FEM models is an important aspect of computational studies to estimate how realistic the physical situation was modeled.

The spine is a large complex structure in the human body and therefore hard to compare between patients. It is an interconnected system that consists of bones, intervertebral discs, nerves, muscles, tendons, and ligaments. Deviations can be found not only in different Lenke types- or curvatures, but for example also in vertebral size, the size and effect of muscles and ligaments, the condition of the intervertebral discs and the location of the nucleus pulposus within the intervertebral disc. These are many parameters that influence the behavior of the spine when subjected to applied loads and deformations. All these differences are in- or excluded in FEM studies, depending on the type of study and model requirements.

Only vertebral structures visible on the CT scan were included in the FEM model. Ligaments and muscles were excluded. For the vertebral segments and IVD's, linear elastic material properties were used to save computation time and reduce errors, while other studies chose to assess less segments but include more components to create a model closer to nature.

No comparable studies were found that that included more than five segments [15], [18], [22]. The stresses in IVD's found in other studies were somewhat similar to the ones found in the validation study in this thesis. The IVD stresses found by Zhang et al. were almost twice as high. An explanation might be that this model included more segments superior to the ones assessed and the applied moment in the current model was applied to the vertebra C7, whereas Zhang et al. applied moment with the same magnitude on the vertebra L1 [5].

In comparison with similar studies, the left lateral bending was submitted to lower stresses. Zhang et al. reported a relatively high stress in the L1L2 IVD in left lateral bending. This difference can be explained by the type of curvature. The frontal lumbar curvature in the FEM model of this thesis is lateral right, where Zhang et al. described a curvature to the lateral left.

The stresses are somewhat larger than in comparable studies. This can be explained as the moment with the moment arm if the applied force F stays the same, as was shown Figure 3.16.

4.6. Assumptions and limitations

The bony material was assumed to be homogeneous isotropic elastic material, because deformations in the vertebrae were not the focus in this thesis. For the annulus fibrosus, a homogeneous ground substance was modeled instead of ligament fibers that encases the nucleus pulposus.

The nucleus and annulus were modeled manually and were not retrieved from the CT scan. The magnitude of both was estimated and are not based on in vivo measurements.

The endplates were neglected. Young's modulus is significantly higher than the young's modulus of intervertebral discs. The focus of this thesis lies on the flexibility of the spine and therefore the behavior of IVD's is more relevant than vertebrae and endplates. Likewise, modeling the endplates would add 36 extra models to the assembly which increases computation time.

Ligaments, tendons, and muscles were not modeled due to computational matters. Most studies that do include ligaments and muscles in their FEA model, only assess a few segments and hardly ever the entire spine.

4.7. Future outlook and potential use

In this thesis, CT scans are used to determine the geometry of the patient-specific spine. However, they are not routinely used for AIS patients. Instead, radiographs are used to determine the curve magnitude and shape. Radiographs are quicker and less harmful in terms of radiation for the patient than CT scans that use radiation. Besides, CT scans are performed in supine position, while the Cobb angle is determined in standing position where gravitational forces are included. Therefore, it would be preferable to find a method in the future that can generate the shape of the spine based on radiographs.

The challenge is that radiograph only take photos in one plane at the time, meaning that for a three-dimensional model, two separate photos should be taken. Between the two radiographs, the patient must turn 90 degrees before the second radiograph can be taken. It is hardly possible to take the two radiographs from the same distance as the patient moves freely. This difference in distance to the x-ray generator leads to beam distortion due to divergence, resulting in misrepresenting images on the radiograph. Therefore, the anteroposterior and lateral radiograph do not match, and anatomical geometry is hard to compile. To get useful radiographs for spine FEM, a solution for the problem described above should be found. Rotating platforms as used can reduce the movement of the patient during rotation, but might still lead to slight distortions on radiographs [19], [42]. Another option would be the use of an EOS biplanar system that captures radiographs in two planes at the same time. This system is still very expensive [43].

The model was meshed in Materialise 3-Matic to ensure all the surfaces were included when imported into Abaqus. Due to few available mesh options in Materialise 3-Matic, only tetrahedral elements could be selected, which is not always the most suitable choice for every model. However, as the surfaces were supposed to transfer coupled with the parts, there was no other option. Comparison with other element types could give insight on the best option and lead to better results. To do so, the model should not be meshed in Materialise 3-matic, but in Abaqus.

More research to the morphology of intervertebral discs can improve knowledge on the origin of AIS. The nucleus tends to shift away from the curvature in scoliotic patients. Whether this is the cause, or a result of scoliosis is not clear.

Although FEM studies never mimic a perfect in vivo situation, they can predict aspects that are not or hardly measurable invasively. Better computers and higher complexity of models will lead to quicker and better results in the future. Automation will help to create patient-specific models more easily with the use of parameters accessory to that patient.

Patient-specific FEM models that predict the most optimal spinal shape and the accessory rod shapes can be helpful for surgery pre-planning in the future. Finite element spine- and rod models can help orthopedic surgeons determining the most advantageous rod shape, predicting the amount of correction that should be accomplished,

This will improve the treatment quality and decrease surgery time. A shorter operation time means less time under general anesthesia, shorter operations and therefore also reduces the costs. In order to accomplish this in the future, more research is necessary to improve FEM models and decline the labor intensity by focusing more on automatization.

5. Conclusion

Adolescent idiopathic scoliosis is a deformation to the spine that can be improved by spinal fusion. With this exploratory research, a Finite element method was composed to predict the shape of a spinal rod to reduce the Cobb angle of an AIS patient as much as possible during surgical procedures. Pre-operative prediction can improve the quality and reduce the time of spinal fusion. Improvements to the FEM Spine- and Rod model, such as automation, generalization, and inclusion of ligaments and muscles in the model, will lead to a better surgical outcome.

References

[1] M. A. Asher en D. C. Burton, 'Adolescent idiopathic scoliosis: natural history and long term treatment effects', *Scoliosis*, vol. 1, nr. 1, p. 2, dec. 2006, doi: 10.1186/1748-7161-1-2.

[2] M. R. Konieczny, H. Senyurt, en R. Krauspe, 'Epidemiology of adolescent idiopathic scoliosis', *J Child Orthop*, vol. 7, nr. 1, pp. 3–9, feb. 2013, doi: 10.1007/s11832-012-0457-4.

[3] S. Negrini *e.a.*, 'Braces for idiopathic scoliosis in adolescents', *Cochrane Database Syst Rev*, nr. 6, p. CD006850, jun. 2015, doi: 10.1002/14651858.CD006850.pub3.

[4] M. Burkus, Á. T. Schlégl, I. O'Sullivan, I. Márkus, C. Vermes, en M. Tunyogi-Csapó, 'Sagittal plane assessment of spino-pelvic complex in a Central European population with adolescent idiopathic scoliosis: a case control study', *Scoliosis*, vol. 13, nr. 1, p. 10, dec. 2018, doi: 10.1186/s13013-018-0156-0.

[5] T. H. Smit, 'Adolescent idiopathic scoliosis: The mechanobiology of differential growth', *JOR SPINE*, vol. 3, nr. 4, p. e1115, 2020, doi: 10.1002/jsp2.1115.

[6] J. Wang, J. Zhang, R. Xu, T. G. Chen, K. S. Zhou, en H. H. Zhang, 'Measurement of scoliosis Cobb angle by end vertebra tilt angle method', *Journal of Orthopaedic Surgery and Research*, vol. 13, nr. 1, p. 223, sep. 2018, doi: 10.1186/s13018-018-0928-5.

[7] M. N. Choudhry, Z. Ahmad, en R. Verma, 'Adolescent Idiopathic Scoliosis', *Open Orthop J*, vol. 10, pp. 143–154, 2016, doi: 10.2174/1874325001610010143.

[8] P. Roussouly en J. L. Pinheiro-Franco, 'Sagittal parameters of the spine: biomechanical approach', *Eur Spine J*, vol. 20, nr. S5, pp. 578–585, sep. 2011, doi: 10.1007/s00586-011-1924-1.

[9] M. Kurutz, 'Finite element modelling of human lumbar spine', in *Finite Element Analysis*, IntechOpen, 2010.

[10] U. F. O. Themes, 'Applied anatomy of the lumbar spine', *Musculoskeletal Key*, 5 juni 2016. https://musculoskeletalkey.com/applied-anatomy-of-the-lumbar-spine/ (geraadpleegd 4 juli 2022).

[11] G. Pattappa, Z. Li, M. Peroglio, N. Wismer, M. Alini, en S. Grad, 'Diversity of intervertebral disc cells: phenotype and function', *J Anat*, vol. 221, nr. 6, pp. 480–496, dec. 2012, doi: 10.1111/j.1469-7580.2012.01521.x.

[12] H. Yoshihara, 'Rods in spinal surgery: a review of the literature', *Spine J*, vol. 13, nr. 10, pp. 1350–1358, okt. 2013, doi: 10.1016/j.spinee.2013.04.022.

[13] M. J. Geck *e.a.*, 'Comparison of surgical treatment in Lenke 5C adolescent idiopathic scoliosis: anterior dual rod versus posterior pedicle fixation surgery: a comparison of two practices', *Spine (Phila Pa 1976)*, vol. 34, nr. 18, pp. 1942–1951, aug. 2009, doi: 10.1097/BRS.0b013e3181a3c777.

[14] S.-I. Suk, J.-H. Kim, S.-S. Kim, en D.-J. Lim, 'Pedicle screw instrumentation in adolescent idiopathic scoliosis (AIS)', *Eur Spine J*, vol. 21, nr. 1, pp. 13–22, jan. 2012, doi: 10.1007/s00586-011-1986-0.

[15] Z. Xiao, L. Wang, H. Gong, D. Zhu, en X. Zhang, 'A non-linear finite element model of human L4-L5 lumbar spinal segment with three-dimensional solid element ligaments', *Theoretical and Applied Mechanics Letters*, vol. 1, nr. 6, p. 064001, 2011, doi: 10.1063/2.1106401.

[16] L. M. R. Rodrigues, F. H. Ueno, A. O. Gotfryd, T. Mattar, E. N. Fujiki, en C. Milani, 'Comparison between different radiographic methods for evaluating the flexibility of scoliosis curves', *Acta Ortop Bras*, vol. 22, nr. 2, pp. 78–81, 2014, doi: 10.1590/1413-78522014220200844.

[17] A. Hamzaoglu, U. Talu, M. Tezer, C. Mirzanli, U. Domanic, en S. B. Goksan, 'Assessment of curve flexibility in adolescent idiopathic scoliosis', *Spine (Phila Pa 1976)*, vol. 30, nr. 14, pp. 1637–1642, jul. 2005, doi: 10.1097/01.brs.0000170580.92177.d2.

[18] Q. Zhang, T. Chon, Y. Zhang, J. S. Baker, en Y. Gu, 'Finite element analysis of the lumbar spine in adolescent idiopathic scoliosis subjected to different loads', *Computers in Biology and Medicine*, vol. 136, p. 104745, sep. 2021, doi: 10.1016/j.compbiomed.2021.104745.

[19] S. Berger *e.a.*, 'Patient-specific spinal stiffness in AIS: a preoperative and noninvasive method', *Eur Spine J*, vol. 24, nr. 2, pp. 249–255, feb. 2015, doi: 10.1007/s00586-014-3623-1.

[20] 'Mimics Innovation Suite | Medical Image Analysis Software'. https://www.materialise.com/en/medical/mimics-innovation-suite (geraadpleegd 1 augustus 2022).

[21] Y.-H. Ahn, W.-M. Chen, K.-Y. Lee, K.-W. Park, en S.-J. Lee, 'Comparison of the load-sharing characteristics between pedicle-based dynamic and rigid rod devices', *Biomed. Mater.*, vol. 3, nr. 4, p. 044101, dec. 2008, doi: 10.1088/1748-6041/3/4/044101.

[22] J. L. Wang, M. Parnianpour, A. Shirazi-Adl, A. E. Engin, S. Li, en A. Patwardhan, 'Development and validation of a viscoelastic finite element model of an L2/L3 motion segment', *Theoretical and Applied Fracture Mechanics*, vol. 28, nr. 1, pp. 81–93, okt. 1997, doi: 10.1016/S0167-8442(97)00032-3.

[23] S. C. Dogru en Y. Z. Arslan, 'Effect of Model Parameters on the Biomechanical Behavior of the Finite Element Cervical Spine Model', *Applied Bionics and Biomechanics*, vol. 2021, p. e5593037, jun. 2021, doi: 10.1155/2021/5593037.

[24] M. R. Etemadifar, A. Andalib, A. Rahimian, en S. M. H. T. Nodushan, 'Cobalt chromium-Titanium rods versus Titanium-Titanium rods for treatment of adolescent idiopathic scoliosis; which type of rod has better postoperative outcomes?', *Rev. Assoc. Med. Bras.*, vol. 64, pp. 1085–1090, dec. 2018, doi: 10.1590/1806-9282.64.12.1085.

[25] A. Angelliaume *e.a.*, 'Titanium vs cobalt chromium: what is the best rod material to enhance adolescent idiopathic scoliosis correction with sublaminar bands?', *Eur Spine J*, vol. 26, nr. 6, pp. 1732–1738, jun. 2017, doi: 10.1007/s00586-016-4838-0.

[26] J. S. Smith *e.a.*, 'Assessment of Symptomatic Rod Fracture After Posterior Instrumented Fusion for Adult Spinal Deformity', *Neurosurgery*, vol. 71, nr. 4, pp. 862–868, okt. 2012, doi: 10.1227/NEU.0b013e3182672aab.

[27] F. Ahmad, C. Sidani, R. Fourzali, en M. Wang, 'Postoperative magnetic resonance imaging artifact with cobalt-chromium versus titanium spinal instrumentation: presented at the

2013 Joint Spine Section Meeting. Clinical article', *Journal of neurosurgery. Spine*, vol. 19, sep. 2013, doi: 10.3171/2013.7.SPINE1359.

[28] E. Dekelbaum, 'Alternative Spinal Fusion Fixation Rod Materials: Polyetheretherketone, Nitinol and Silicon Nitride', p. 2.

[29] 'American Elements'. [Online]. Beschikbaar op: https://www.americanelements.com/

[30] J. Litak *e.a.*, 'Metallic Implants Used in Lumbar Interbody Fusion', *Materials (Basel)*, vol. 15, nr. 10, p. 3650, mei 2022, doi: 10.3390/ma15103650.

[31] Smith, Michael, *ABAQUS/Standard User's Manual, Version 6.9*. 2009.

[32] K. Albertsson-Wikland *e.a.*, 'Swedish references for weight, weight-for-height and body mass index: The GrowUp 1990 Gothenburg study', *Acta Paediatrica*, vol. 110, nr. 2, pp. 537–548, 2021, doi: 10.1111/apa.15477.

[33] 'Calculate the angles of two lines, online calculator and formula'. https://www.red-crab-software.com/en/Calculator/Angles-Of-Lines (geraadpleegd 4 augustus 2022).

[34] V. Deviren, S. Berven, F. Kleinstueck, J. Antinnes, J. A. Smith, en S. S. Hu, 'Predictors of flexibility and pain patterns in thoracolumbar and lumbar idiopathic scoliosis', *Spine (Phila Pa 1976)*, vol. 27, nr. 21, pp. 2346–2349, nov. 2002, doi: 10.1097/00007632-200211010-00007.

[35] E. Ameri, H. Behtash, B. Mobini, en A. Daraie, 'Predictors of curve flexibility in adolescent idiopathic scoliosis: a retrospective study of 100 patients', *Acta Med Iran*, vol. 53, nr. 3, pp. 182–185, 2015.

[36] K. H. Yeung, G. Man, A. Hung, T. P. Lam, J. Cheng, en W. Chu, 'Morphological changes of intervertebral disc in relation with curve severity of patients with Adolescent Idiopathic Scoliosis - a T2-weighted MRI study', *Stud Health Technol Inform*, vol. 280, pp. 37–39, jun. 2021, doi: 10.3233/SHTI210431.

[37] S. K. Eswaran, A. Gupta, en T. M. Keaveny, 'LOCATIONS OF BONE TISSUE AT HIGH RISK OF INITIAL FAILURE DURING COMPRESSIVE LOADING OF THE HUMAN VERTEBRAL BODY', *Bone*, vol. 41, nr. 4, pp. 733–739, okt. 2007, doi: 10.1016/j.bone.2007.05.017.

[38] D. L. Skaggs, M. Weidenbaum, J. C. Iatridis, A. Ratcliffe, en V. C. Mow, 'Regional variation in tensile properties and biochemical composition of the human lumbar anulus fibrosus', *Spine (Phila Pa 1976)*, vol. 19, nr. 12, pp. 1310–1319, jun. 1994, doi: 10.1097/00007632-199406000-00002.

[39] Z.-Q. Chen *e.a.*, 'Using precisely controlled bidirectional orthopedic forces to assess flexibility in adolescent idiopathic scoliosis: comparisons between push-traction film, supine side bending, suspension, and fulcrum bending film', *Spine (Phila Pa 1976)*, vol. 36, nr. 20, pp. 1679–1684, sep. 2011, doi: 10.1097/BRS.0b013e31820e6265.

[40] M. A. Halanski, C. M. Elfman, J. A. Cassidy, N. E. Hassan, S. A. Sund, en K. J. Noonan, 'Comparing results of posterior spine fusion in patients with AIS: Are two surgeons better than one?', *J Orthop*, vol. 10, nr. 2, pp. 54–58, jun. 2013, doi: 10.1016/j.jor.2013.03.001.

[41] 'Mild, Moderate or Severe Scoliosis', *Scoliosis 3DC®*. https://scoliosis3dc.com/scoliosis-resources/mild-moderate-severe-scoliosis/ (geraadpleegd 26 november 2022).

[42] P. Büchler *e.a.*, ‘Axial suspension test to assess pre-operative spinal flexibility in patients with adolescent idiopathic scoliosis’, *Eur Spine J*, vol. 23, nr. 12, pp. 2619–2625, dec. 2014, doi: 10.1007/s00586-014-3386-8.

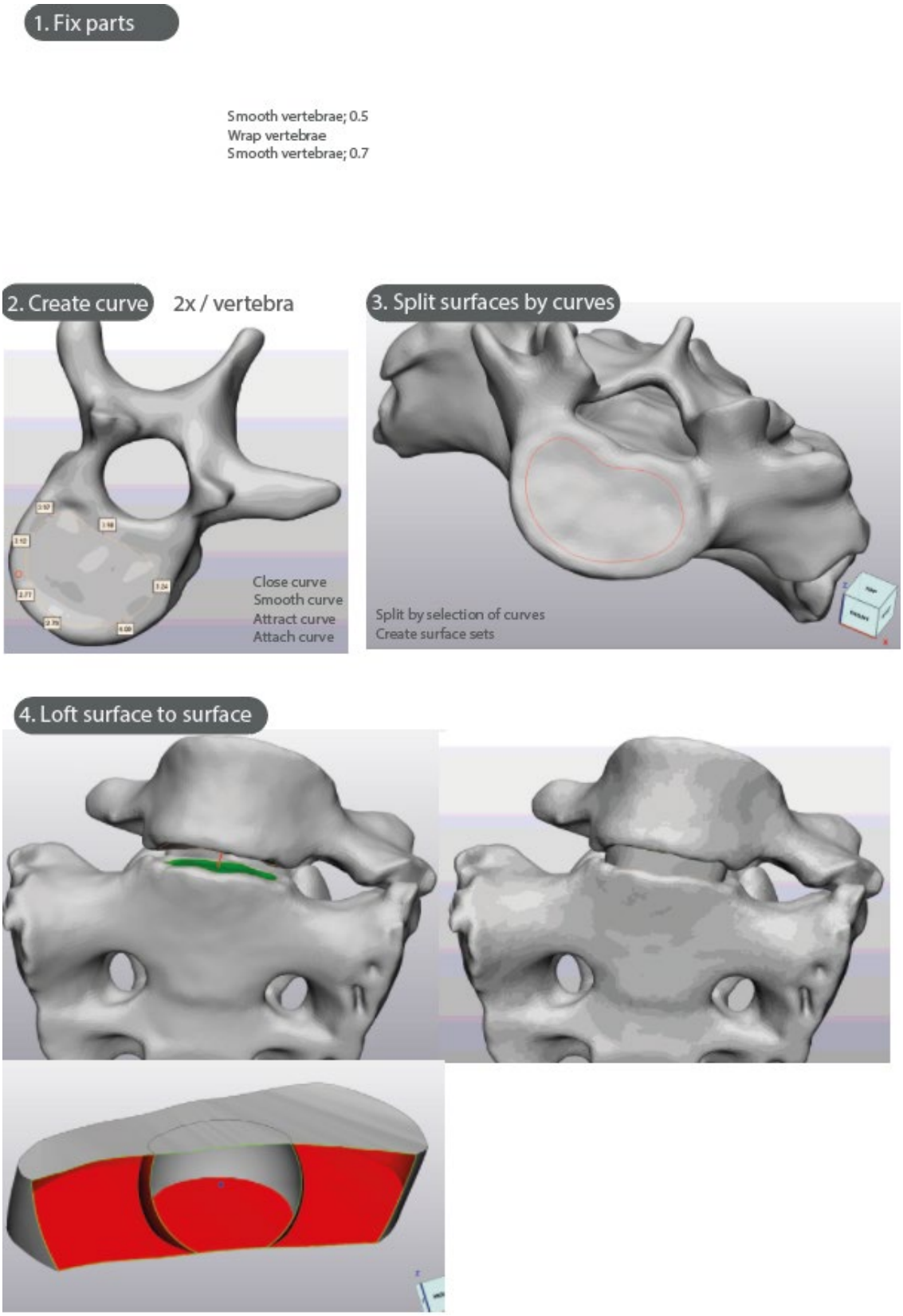
[43] E. Melhem, A. Assi, R. El Rachkidi, en I. Ghanem, ‘EOS® biplanar X-ray imaging: concept, developments, benefits, and limitations’, *J Child Orthop*, vol. 10, nr. 1, pp. 1–14, feb. 2016, doi: 10.1007/s11832-016-0713-0.

[44] V. K. Goel, B. T. Monroe, L. G. Gilbertson, en P. Brinckmann, ‘Interlaminar Shear Stresses and Laminae Separation in a Disc: Finite Element Analysis of the L3-L4 Motion Segment Subjected to Axial Compressive Loads’, *Spine*, vol. 20, nr. 6, pp. 689–698, mrt. 1995, doi: 10.1097/00007632-199503150-00010.

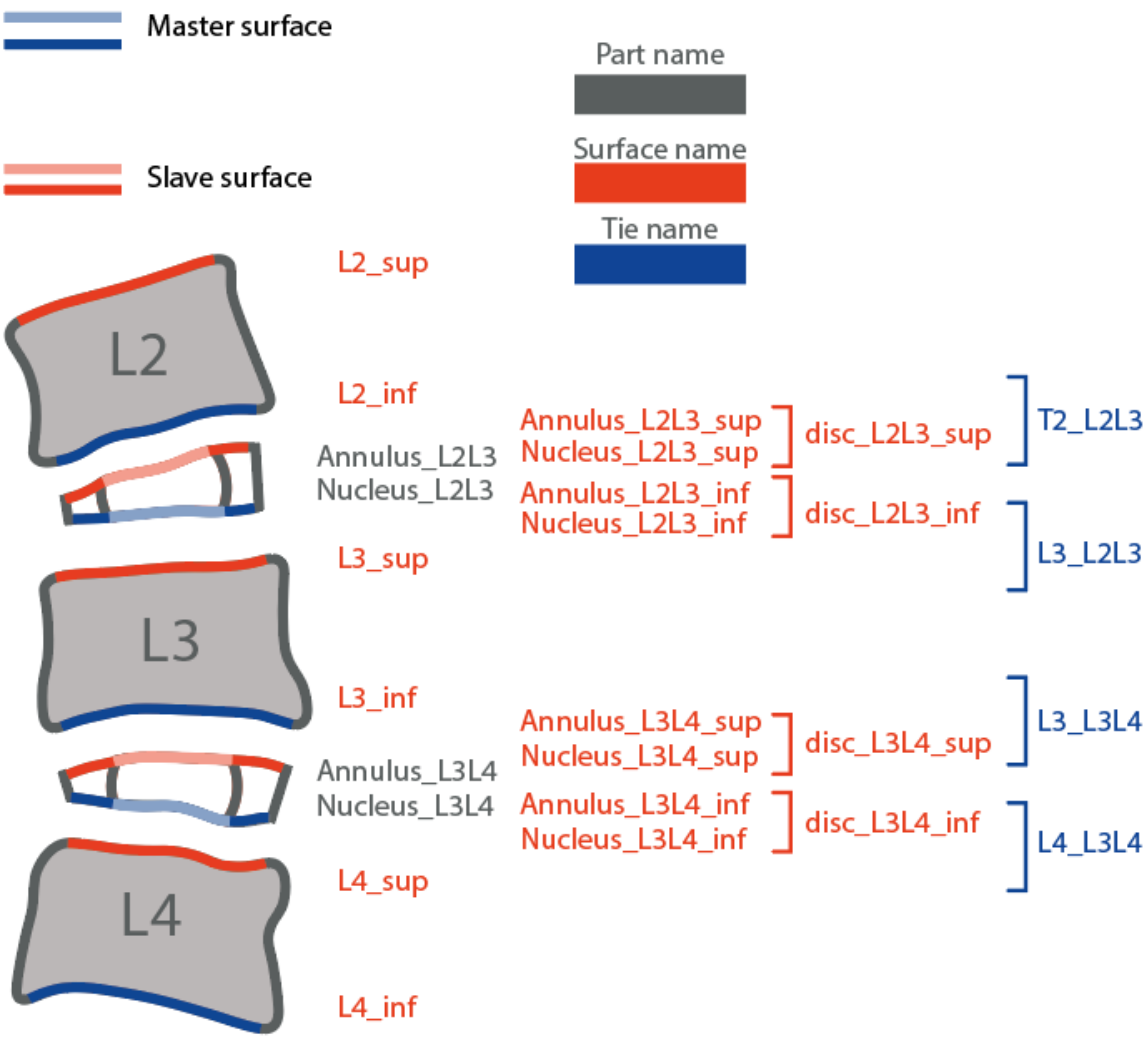
[45] J. L. Wang, M. Parnianpour, A. Shirazi-Adl, A. E. Engin, S. Li, en A. Patwardhan, ‘Development and validation of a viscoelastic finite element model of an L2/L3 motion segment’, *Theoretical and Applied Fracture Mechanics*, vol. 28, nr. 1, pp. 81–93, okt. 1997, doi: 10.1016/S0167-8442(97)00032-3.

Appendices

Appendix A: Pre-processing 3Matic



Appendix B: Part names



Appendix C: Material study

Material analysis was performed on one IVD that consisted of a NP and AF. The chosen LE properties are shown in Table 5.1 below. The AF was modelled solely as annulus ground substance, as opposed to layers of collagen fiber. The nucleus was modelled having a Poisson's ratio close to 0.5, meaning the volumetric strain is equal to zero and therefore acts as a nearly incompressible material. The material properties were obtained from articles concerning finite element analyses [18], [21], [44].

Table 5.1: Linear-elastic material properties

Component name	Young's Modulus [MPa]	Poisson's ratio
Annulus Fibrosus (ground substance)	4.2	0.45
Nucleus Pulposus	1	0.499 (incompressible)
Vertebra (cortical bone)	12000	0.30

For the hyper elastic model, Moonley-Rivlin coordinates were obtained from Xiao et al., as shown in table 5.2. The three parameter Moonley-Rivlin model is regularly mentioned in spinal FEA and is mainly suitable for describing shear moduli. The three parameters are described in table 3.2. The material properties follow isotropic behavior [15].

Table 5.2: Hyper-elastic material properties

Component name	C10	C01	D1
Annulus Fibrosus	0.56	0.14	0.01
Nucleus Pulposus	0.25	0.02	0.01

Table 5.3: Visco-elastic material properties in time domain, using Prony series.

Component name	Young's Modulus [MPa]	Poisson's ratio	Shear (G)	Bulk (k)	Time (τ)
Annulus Fibrosus	4.2	0.45	0.399	0.399	3.45
			0	0.3	100
			0.361	0.149	1000
			0.108	0.15	5000
Nucleus Pulposus	1	0.499	0.638	0	0.141
			0.156	0	2.21
			0.12	0	39.9
			0.0383	0	266
			0	0	500

The AF and NP were both given different material properties. The NP was given visco-elastic material properties, as it consists of a gel-like structure. The AF was given linear-elastic material properties. The used material properties are stated in table 5.3. The AF was modeled solely as the AF ground substance, as opposed to a collagen fiber composite. For the viscoelastic material properties, a Prony Series for FE models was following the formula as described by Wang et al. (1997) [22]. Prony series follow an equation that represents the relation between shear modulus and time.

$$G_R(t) = \frac{G(t)}{G_0} = 1 - \sum_{i=1}^{n} g_i \left(1 - e^{-\frac{t}{\tau_i}}\right)$$

Boundary conditions

The following boundary conditions for this material study were chosen;

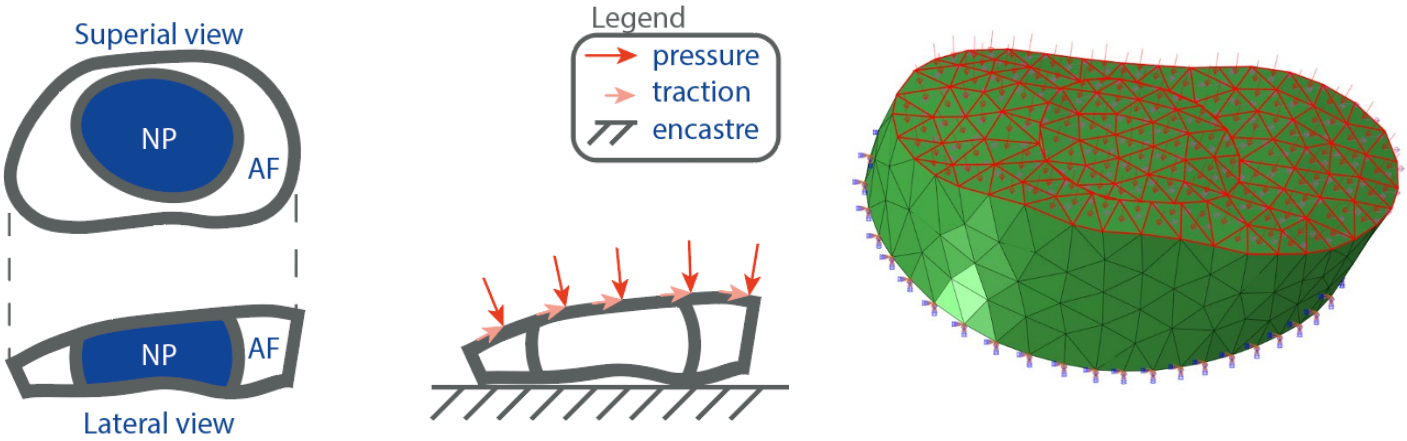


Figure 5.1: The used boundary conditions and forces for the model.

Under physiological loading conditions, IVD's are exposed to tension-, compression- and shear stresses. The entire inferior node surface of the disc was encastred to fix the intervertebral disc in space see Figure 5.1.

Loading conditions

Two types of loads were applied to the IVD's. Firstly, a uniformly distributed pressure was applied to the superior site of the disc with a magnitude of 0.1 N/A. Secondly, a surface traction with a magnitude of 0.2 N/A was applied to mimic shear force.

Table 5.4: Intervertebral disc model mesh details

	Nucleus	*Annulus*	*Disc (NP + AF)*
Volume	2043.27	3621.85	5665.12
Nodes	116	289	405
Elements (Tet type C3D4)	280	730	1010

Material analysis

Figure 5.2 shows the three types of properties with the accessory stress scale. The maximum values are shown in Table 5.5. The visco elastic properties are said to be the closest to nature [45].

The chosen material properties for the entire model were chosen to be linear elastic to reduce the computation time and prevent convergence problems. For future reference, hyper- and visco- elastic properties can be used after the model is remeshed. As can be seen in Figure 5.2, the stresses for the VE NP are higher than in the LE model and the stress for the VE AF are lower than in the LE model. Although this will not be discussed further in this thesis, this difference will have an effect on the outcome of the FE model.

Figure 5.2: Von Mises stresses on linear-, hyper- and visco elastic model

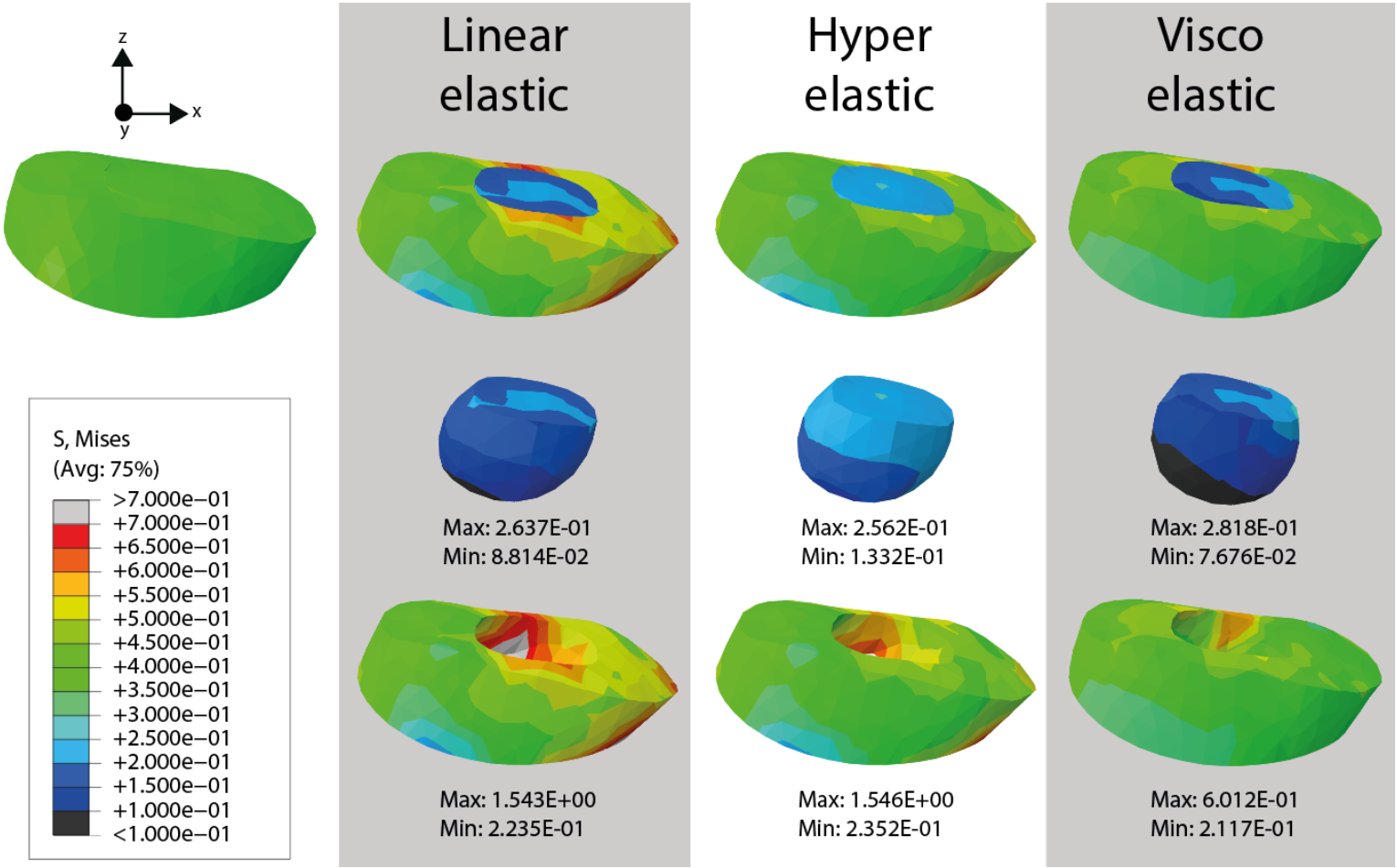


Table 5.5: Stress and deformation outcome of material study

	Linear elastic	Hyper elastic	Visco elastic
Max. Von Mises stress [MPa] (average)	1.544E+00 (AF: node 164)	1.510E+00 (AF: node 103) AF max: 1.546E+00	6.012E-01 (AF: node 232)
Min. Von Mises stress [MPa] (average)	7.684E-02 (NP: node 2)	1.298E-01 (NP: node 2	7.676E-02 (NP: node 116)
Deformation magnitude (U) [mm]	2.850E+00	2.294E+00	7.00E-01

Appendix D: Properties of the IVD's mesh

Table 5.6: Properties of the IVD lofts

Disc name	$V_{Annulus}$	$V_{Nucleus}$	#Nodes Annulus	#Nodes Nucleus	elements Annulus C3D4	elements Nucleus C3D4	elements Disc C3D4	R_{Sphere}	Triangle edge length
C7T1	212.77	80.03	185	60	419	132		3.5	1.5
T1T2	311.12	170.38	237	99	545	242		4.5	1.5
T2T3	349.64	332.73	159	95	323	225		6.0	2.0
T3T4	362.05	393.46	160	110	323	272		6.5	2.0
T4T5	336.77	506.32	166	127	332	308		7.0	2.0
T5T6	458.01	507.11	193	139	395	341		7.0	2.0
T6T7	597.98	601.87	229	148	505	369		7.0	2.0
T7T8	670.24	770.93	257	171	582	438		7.5	2.0
T8T9	770.84	941.14	283	181	668	462		8.0	2.0
T9T10	902.83	903.63	287	177	675	445		8.0	2.0
T10T11	984.30	863.27	295	173	699	434		8.0	2.0
T11T12	836.92	635.65	272	71	624	163		8.0	3.0
T12L1	1791.63	1192.48	189	91	428	209		8.5	3.0
L1L2	2336.15	1502.14	223	103	525	241		9.0	3.0
L2L3	3172.11	1719.92	260	108	636	253		9.0	3.0
L3L4	3799.17	1940.09	289	114	727	284		9.0	3.0
L4L5	3621.85	2043.27	289	116	730	280		9.0	3.0
L5S1	1201.20	1426.22	158	107	309	254		9.5	3.0
Total	22715.58	16530.64	4131	2190	9445	5352			

The IVD is named after the adjacent discs, respectively the ones above and below the IVD. Sphere radius and triangle edge length are measured in 3-Matic. Volumes are obtained from the meshed model in Abaqus.

Appendix E: Properties of the vertebrae mesh

Table 5.7: Properties of the vertebrae models, element type = tet

Vertebra name	$V_{Vertebra}$ [mm^3] (mesh)	Number of nodes	Number of elements
C7	5799.87	248	86
T1	7384.82	274	97
T2	7150.88	269	93
T3	7541.35	266	95
T4	9482.44	62	106
T5	10386.43	66	117
T6	11011.51	69	119
T7	12845.46	352	128
T8	14617.38	78	142
T9	16291.65	82	157
T10	17486.39	82	163
T11	18660.31	83	166
T12	20651.66	93	186
L1	24430.96	104	203
L2	27721.61	112	230
L3	28790.72	116	223
L4	29093.26	113	232
L5	33775.06	128	266
S1	91951.77	264	693
Total	396116	2861	3502

Part volume, number of nodes and number of elements are measured of the meshed model in Abaqus.

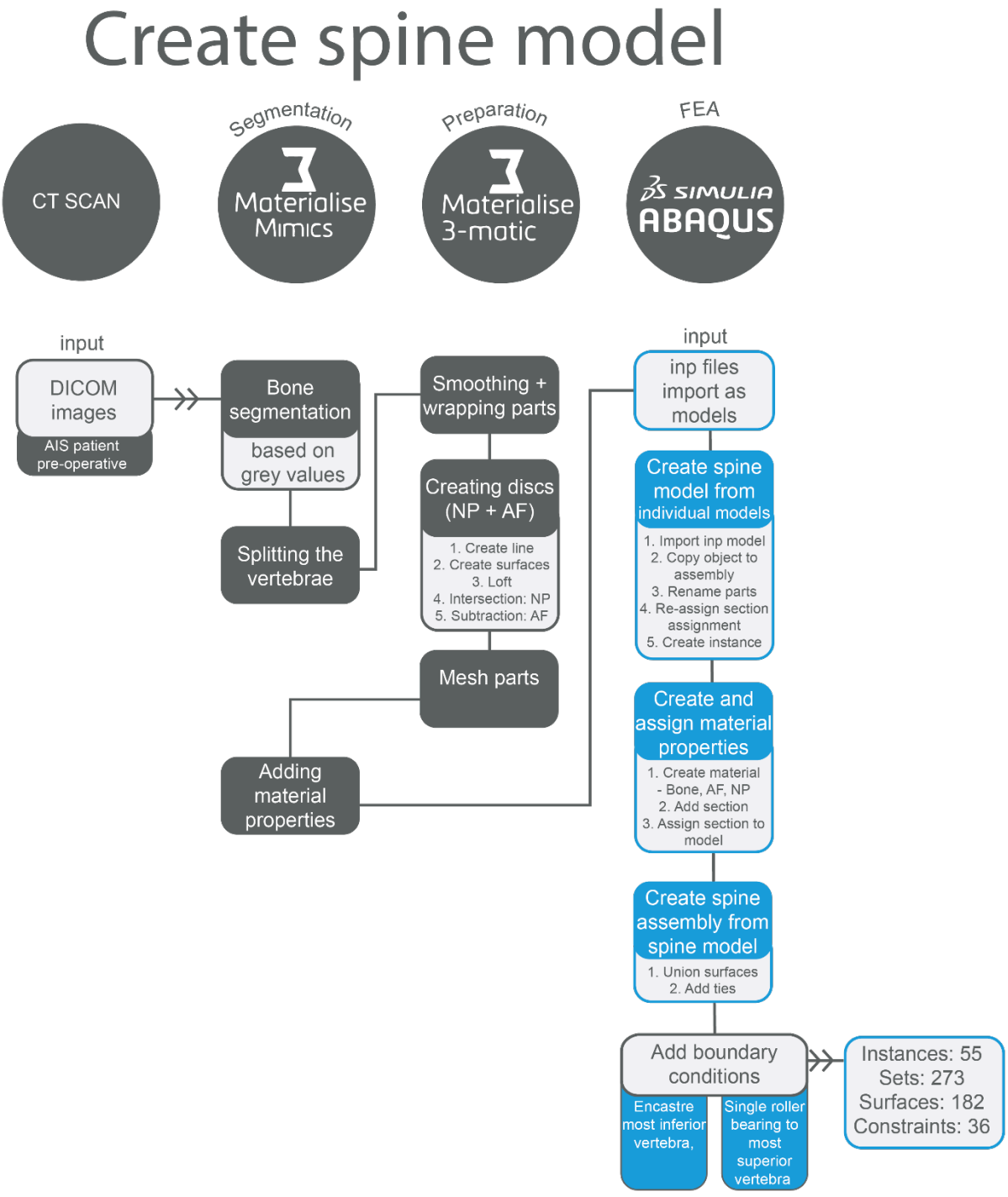
Appendix F: Step specifications of the analysis

Table 5.8: Step specifications during analysis

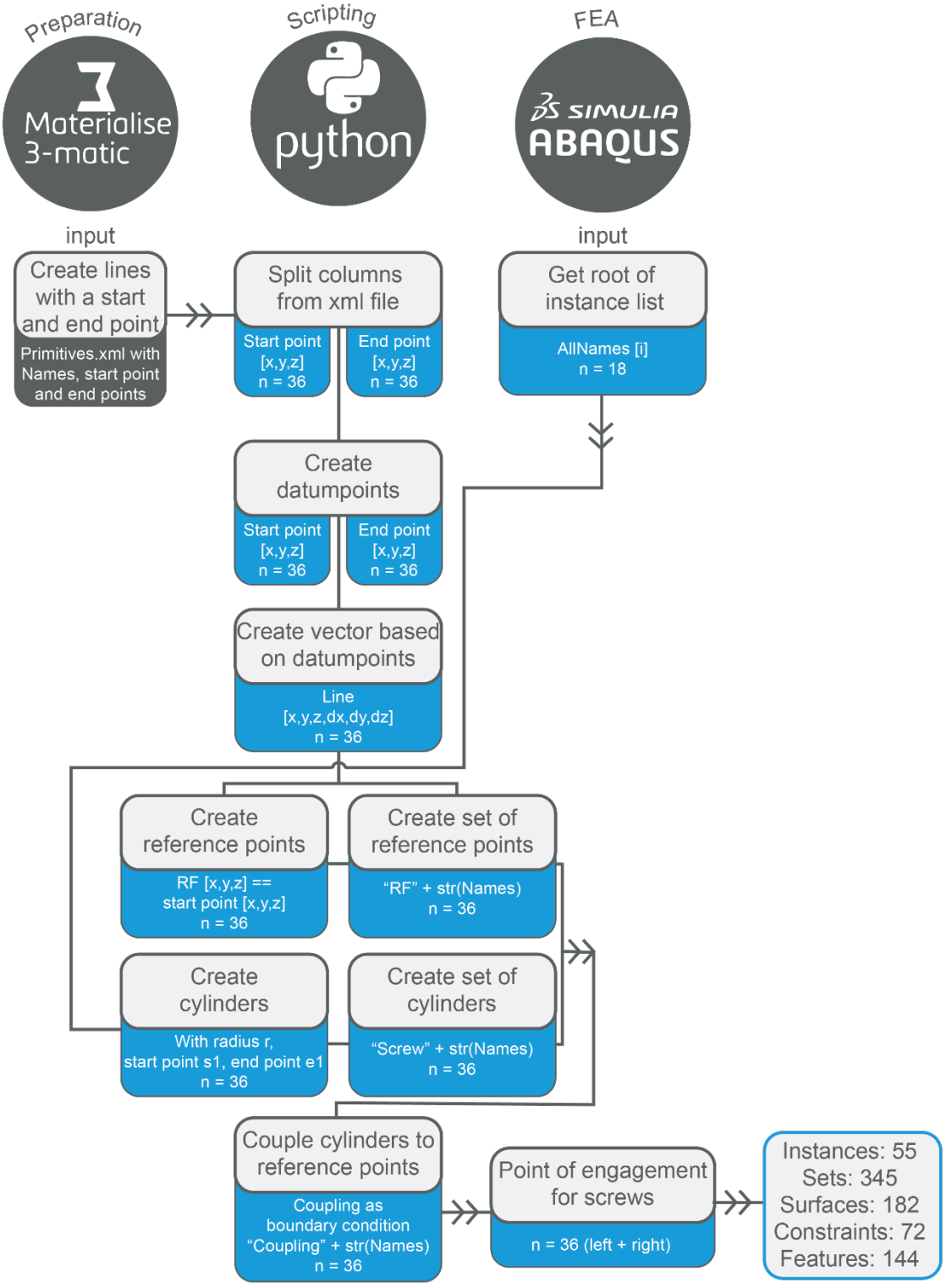
Step information			*Translation [mm]*			*Rotation [rad]*			*Time*
Steps	*Vertebra set*	*Fixed*	*u1*	*u2*	*u3*	*ur1*	*ur2*	*ur3*	*t [s]*
Step 1	C7-R	On	SET	SET	UNSET	SET	SET	SET	
	T1-R	On	SET	SET	UNSET	SET	SET	SET	
	T2-R	Off	UNSET	UNSET	UNSET	UNSET	-5	UNSET	
Step 2	T3-R	Off	UNSET	UNSET	UNSET	UNSET	-30	UNSET	
Step 3	T3-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	-15	
Step 4	T3-R	Off	UNSET	UNSET	25	UNSET	UNSET	UNSET	
Step 5	T4-R	Off	UNSET	UNSET	UNSET	UNSET	-60	UNSET	
	T2-R	On	SET	SET	UNSET	SET	SET	SET	
Step 6	T4-R	Off	UNSET	UNSET	20	UNSET	UNSET	UNSET	
Step 7	T4-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	-12	
Step 8	T5-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	5	
Step 9	T6-R	Off	UNSET	UNSET	UNSET	UNSET	-15	UNSET	
Step 10	T10-R	Off	UNSET	UNSET	UNSET	UNSET	40	UNSET	
Step 11	T6-R	Off	UNSET	UNSET	UNSET	UNSET	-35	UNSET	
Step 12	T5-R	Off	UNSET	UNSET	UNSET	UNSET	-60	UNSET	
Step 13	T7-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	40	
Step 14	T9-R	Off	UNSET	UNSET	UNSET	UNSET	40	UNSET	
Step 15	T9-R	Off	UNSET	UNSET	UNSET	UNSET	40	UNSET	
Step 16	L4-R	Off	UNSET	UNSET	UNSET	UNSET	-40	UNSET	
Step 17	T5-R	Off	UNSET	UNSET	UNSET	UNSET	-35	UNSET	
Step 18	T10-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	10	
Step 19	T11-R	Off	UNSET	UNSET	UNSET	UNSET	55	UNSET	
Step 20	T11-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	15	
Step 21	T12-R	Off	UNSET	UNSET	UNSET	UNSET	75	UNSET	
Step 22	T12-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	5	
Step 23	L1-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	5	
Step 24	L1-R	Off	UNSET	UNSET	UNSET	UNSET	35	UNSET	
Step 25	L2-R	Off	UNSET	UNSET	UNSET	UNSET	55	UNSET	
Step 26	L2-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	5	
Step 29	L3-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	5	
Step 30	L3-R	Off	UNSET	UNSET	UNSET	UNSET	-15	UNSET	
Step 31	L4-R	Off	UNSET	UNSET	UNSET	UNSET	-35	UNSET	
Step 32	L4-R	Off	UNSET	UNSET	UNSET	UNSET	UNSET	5	3643

Overview of all used steps during simulation. The dark colored, light colored and medium-colored cells represent respectively fixed vertebrae, translation where u1, u2, u3 are translations along x, y and z axis and rotation where ur1, ur2 and ur3 are rotation about the x, y and z axis.

Appendix G: Workflows of code



Create RP's


Preparation
Materialise 3-matic
Scripting
python
FEA
SIMULIA ABAQUS
input
Create lines with a start and end point
Primitives.xml with Names, start point and end points
Split columns from xml file
Start point [x,y,z] n = 36
End point [x,y,z] n = 36
input
Get root of instance list
AllNames [i] n = 18
Create datumpoints
Start point [x,y,z] n = 36
End point [x,y,z] n = 36
Create vector based on datumpoints
Line [x,y,z,dx,dy,dz] n = 36
Create reference points
RF [x,y,z] == start point [x,y,z] n = 36
Create set of reference points
"RF" + str(Names) n = 36
Create cylinders
With radius r, start point s1, end point e1 n = 36
Create set of cylinders
"Screw" + str(Names) n = 36
Couple cylinders to reference points
Coupling as boundary condition "Coupling" + str(Names) n = 36
Point of engagement for screws
n = 36 (left + right)
Instances: 55
Sets: 345
Surfaces: 182
Constraints: 72
Features: 144

Appendix H: Calculate ROM in Matlab

```
%% THESIS CALCULATING ROM AT DIFFERENT MOTIONS
% By Jette Smedema, August 4, 2021

%% CLEAR WORKSPACE
clear; close all; clc

%% Flexion (y,z)
% Line 1 (intitial) (base coordinates)
% A coordinates # node 740
a_1 = -1.29575e+02;
a_2 = -4.65696e+02;

% A coordinates # node 362
b_1 = -1.32482e+02;
b_2 = -4.69579e+02;

% Line 2 (deformed) (scaled)
% B coordinates # node 740
c_1 = -1.64725e+02;
c_2 = -4.72290e+02;

% B coordinates # node 362
d_1 = -1.66319e+02;
d_2 = -4.77098e+02;

%% Extension (y,z)
% Line 1 (intitial) (base coordinates)
% A coordinates # node 740
a_1 = -1.29575e+02;
a_2 = -4.65696e+02;

% A coordinates # node 362
b_1 = -1.32482e+02;
b_2 = -4.69579e+02;

% Line 2 (deformed) (scaled)
% B coordinates # node 740
c_1 = -9.30401e+01;
c_2 = -4.66961e+02;

% B coordinates # node 362
d_1 = -9.74289e+01;
d_2 = -4.69654e+02;

%% Left lateral bending (x,z)
% Line 1 (intitial) (base coordinates)
% A coordinates # node 740
a_1 = 2.50835e+01;
a_2 = -4.65696e+02;

% A coordinates # node 362
b_1 = 2.11481e+01;
b_2 = -4.69579e+02;

% Line 2 (deformed) (scaled)
% B coordinates # node 740
```

```
c_1 = 6.14152e+01;
c_2 = -4.79081e+02;

% B coordinates # node 362
d_1 =  5.59708e+01;
d_2 =  -4.82282e+02;


%% Right lateral bending (x,z)
% Line 1 (intitial) (base coordinates)
% A coordinates # node 740
a_1 = 2.50835e+01;
a_2 = -4.65696e+02;

% A coordinates # node 362
b_1 = 2.11481e+01;
b_2 = -4.69579e+02;

% Line 2 (deformed) (scaled)
% B coordinates # node 740
c_1 = -5.54358e+00;
c_2 = -4.68417e+02;

% B coordinates # node 362
d_1 =  -7.45563e+00;
d_2 =  -4.74464e+02;

%% Left axial rotation (x,y)
% Line 1 (intitial) (base coordinates)
% A coordinates # node 740
a_1 = 2.50835e+01;
a_2 = -1.29575e+02;

% A coordinates # node 362
b_1 = 2.11481e+01;
b_2 = -1.32482e+02;

% Line 2 (deformed) (scaled)
% B coordinates # node 740
c_1 = 3.88860e+01;
c_2 = -1.32130e+02;

% B coordinates # node 362
d_1 = 3.74056e+01;
d_2 = -1.37144e+02;


%% Right axial rotation (x,y)
% Line 1 (intitial) (base coordinates)
% A coordinates # node 740
a_1 = 2.50835e+01;
a_2 = -1.29575e+02;

% A coordinates # node 362
b_1 = 2.11481e+01;
b_2 = -1.32482e+02;

% Line 2 (deformed) (scaled)
% B coordinates # node 740
```

```
c_1 = 2.81457e+01;
c_2 = -1.58543e+02;

% B coordinates # node 362
d_1 = 2.20549e+01;
d_2 = -1.58693e+02;


%% Move lines to one starting point
% Lists with coordinates
A = [a_1; a_2];
B = [b_1; b_2];

C = [c_1; c_2];
D = [d_1; d_2];

% Lines
lineAB = B - A;
lineCD = D - C;

% Scalar product of the two vectors
ABCD = (lineAB(1) * lineCD(1)) + (lineAB(2) * lineCD(2));

% Magnitude of the vectors
AB = sqrt((lineAB(1)^2)+(lineAB(2)^2));
CD = sqrt((lineCD(1)^2)+(lineCD(2)^2));

Cosalpha = ABCD/(AB*CD);
Alpha_rad = acos(Cosalpha);
Alpha_deg = rad2deg(Alpha_rad)
```

Appendix I: Python codes

Name of the script files and their function

ValidateLinearModel.py
Creates the assembly of the Spine model with different motions
Import…

```
# Change modelname
modelname = 'SpineModel'
mdb.models.changeKey(fromName='Model-1', toName=modelname)

a = mdb.models[modelname].rootAssembly

# ------------------------------------------------------------------------
-----------------------------------------
## NAME PARTS
# List with names of all 55 models
PartModelName = ['C7', 'C7T1_Annulus', 'C7T1_Nucleus',
                 'T1', 'T1T2_Annulus', 'T1T2_Nucleus',
                 'T2', 'T2T3_Annulus', 'T2T3_Nucleus',
                 'T3', 'T3T4_Annulus', 'T3T4_Nucleus',
                 'T4', 'T4T5_Annulus', 'T4T5_Nucleus',
                 'T5', 'T5T6_Annulus', 'T5T6_Nucleus',
                 'T6', 'T6T7_Annulus', 'T6T7_Nucleus',
                 'T7', 'T7T8_Annulus', 'T7T8_Nucleus',
                 'T8', 'T8T9_Annulus', 'T8T9_Nucleus',
                 'T9', 'T9T10_Annulus', 'T9T10_Nucleus',
                 'T10', 'T10T11_Annulus', 'T10T11_Nucleus',
                 'T11', 'T11T12_Annulus', 'T11T12_Nucleus',
                 'T12', 'T12L1_Annulus', 'T12L1_Nucleus',
                 'L1', 'L1L2_Annulus', 'L1L2_Nucleus',
                 'L2', 'L2L3_Annulus', 'L2L3_Nucleus',
                 'L3', 'L3L4_Annulus', 'L3L4_Nucleus',
                 'L4', 'L4L5_Annulus', 'L4L5_Nucleus',
                 'L5', 'L5S1_Annulus', 'L5S1_Nucleus',
                 'S1']

DiscNames = ['C7T1', 'T1T2','T2T3', 'T3T4', 'T4T5', 'T5T6', 'T6T7', 'T7T8',
'T8T9', 'T9T10', 'T10T11', 'T11T12', 'T12L1', 'L1L2', 'L2L3', 'L3L4',
'L4L5', 'L5S1']

# Individual names per modeltype
PartModelNameVertebra = (PartModelName[0::3]) # 19 parts
PartModelNameAnnulus = (PartModelName[1::3]) # 18 parts
PartModelNameNucleus = (PartModelName[2::3]) # 18 parts

# ------------------------------------------------------------------------
-----------------------------------------
## IMPORT ALL MODELS INTO ABAQUS
for i in range(len(PartModelName)):
    # Import inp partmodel files
        ShowSession = session.viewports['Viewport: 1'].setValues(dis-
playedObject=a)
        mdb.ModelFromInputFile(name=PartModelName[i],
        inputFileName = ('C:/1. Abaqus tijdelijk/220322 ABAQUS MODELLEN EN
SCRIPTS/inp parts/' + PartModelName[i] + '.inp'))

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

```
## Create element sets of element surfaces
SurfaceAnnulusInferior_ES, SurfaceSetInferiorAnnulus_ES, SurfaceAnnulusSu-
perior_ES, SurfaceSetSuperiorAnnulus_ES, \
SurfaceNucleusInferior_ES, SurfaceSetInferiorNucleus_ES, SurfaceNucleusSu-
perior_ES, SurfaceSetSuperiorNucleus_ES= [], [], [], [], [], [], [], []
Surface_Vertebra_Inferior_ES, SurfaceSet_Vertebra_Inferior_ES = [], []

for k in range(len(PartModelNameAnnulus)):
    SurfaceAnnulusInferior_ES = mdb.models[PartModelNameAnnulus[k]].rootAs-
sembly.surfaces[DiscNames[k] + '_ANNULUS_INFERIOR'].elements
    SurfaceSetInferiorAnnulus_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.Set(elements=SurfaceAnnulusInferior_ES, name='ES_'+
DiscNames[k] + '_ANNULUS_INFERIOR_SET')

    SurfaceAnnulusSuperior_ES = mdb.models[PartModelNameAnnulus[k]].rootAs-
sembly.surfaces[DiscNames[k] + '_ANNULUS_SUPERIOR'].elements
    SurfaceSetSuperiorAnnulus_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.Set(elements=SurfaceAnnulusSuperior_ES, name='ES_' +
DiscNames[k] + '_ANNULUS_SUPERIOR_SET')

    SurfaceNucleusInferior_ES = mdb.models[PartModelNameNucleus[k]].rootAs-
sembly.surfaces[DiscNames[k] + '_NUCLEUS_INFERIOR'].elements
    SurfaceSetInferiorNucleus_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.Set(elements=SurfaceNucleusInferior_ES, name='ES_'+
DiscNames[k] + '_NUCLEUS_INFERIOR_SET')

    SurfaceNucleusSuperior_ES = mdb.models[PartModelNameNucleus[k]].rootAs-
sembly.surfaces[DiscNames[k] + '_NUCLEUS_SUPERIOR'].elements
    SurfaceSetSuperiorNucleus_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.Set(elements=SurfaceNucleusSuperior_ES, name='ES_' +
DiscNames[k] + '_NUCLEUS_SUPERIOR_SET')

for k in range(len(PartModelNameVertebra)-1):
    Surface_Vertebra_Inferior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.surfaces['ES_' + PartModelNameVertebra[k] + '_INFERI-
OR'].elements
    SurfaceSet_Vertebra_Inferior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.Set(elements=Surface_Vertebra_Inferior_ES, name='ES_'+
PartModelNameVertebra[k] + '_INFERIOR_SET')

for k in range(len(PartModelNameVertebra)):
    Surface_Vertebra_Superior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.surfaces['ES_' + PartModelNameVertebra[k] + '_SUPERI-
OR'].elements
    SurfaceSet_Vertebra_Superior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.Set(elements=Surface_Vertebra_Superior_ES, name='ES_'+
PartModelNameVertebra[k] + '_SUPERIOR_SET')

# ------------------------------------------------------------------------
-----------------------------------------
## MATERIAL PROPERTIES
## Create empty lists
NameMatPropNucleus_old, NameMatPropAnnulus_old, NameMatPropVertebra_old =
[], [], []
NameMatPropNucleus_new, NameMatPropAnnulus_new, NameMatPropVertebra_new =
[], [], []

for h in range(len(DiscNames)):
## NUCLEUS PULPOSUS
    # Change Material properties for nucleus
    NameMatPropNucleus_old.append(DiscNames[h] + '_NUCLEUS_COPY_MATERIAL0')
```

```
    NameMatPropNucleus_new.append(DiscNames[h] + '_NUCLEUS_MATERIAL')

    ChangeKeyMatPropNucleus = mdb.models[PartModelNameNucleus[h]].materi-
als.changeKey(
        fromName=NameMatPropNucleus_old[h], toName=NameMatPropNu-
cleus_new[h])

    # Elastic material properties + Density
    mdb.models[PartModelNameNucleus[h]].materials[NameMatPropNu-
cleus_new[h]].elastic.setValues(
        table=((1.0, 0.45),), moduli=LONG_TERM)
    mdb.models[PartModelNameNucleus[h]].materials[NameMatPropNu-
cleus_new[h]].Density(table=((

1.0e-09,),))

    # Edit section
    EditSectionNucleus = mdb.models[PartModelNameNucleus[h]].sections['Sec-
tion-1-ES_VOLUME0_MAT0'].setValues(
        material=PartModelNameNucleus[h] + '_MATERIAL', thickness=None)

## ANNULUS FIBROSUS
    # Change Material properties for annulus
    NameMatPropAnnulus_old.append(DiscNames[h] + '_ANNULUS_COPY_MATERIAL0')
    NameMatPropAnnulus_new.append(DiscNames[h] + '_ANNULUS_MATERIAL')

    ChangeKeyMatPropAnnulus = mdb.models[PartModelNameAnnulus[h]].materi-
als.changeKey(
    fromName=NameMatPropAnnulus_old[h], toName=NameMatPropAnnulus_new[h])

    # Elastic material properties + Density
    mdb.models[PartModelNameAnnulus[h]].materials[NameMatPropAnnu-
lus_new[h]].elastic.setValues(
        table=((4.2, 0.45),), moduli=LONG_TERM)
    mdb.models[PartModelNameAnnulus[h]].materials[NameMatPropAnnu-
lus_new[h]].Density(table=((

1.2e-09,),))

    # Edit section
    EditSectionAnnulus = mdb.models[PartModelNameAnnulus[h]].sections['Sec-
tion-1-ES_VOLUME0_MAT0'].setValues(
        material=PartModelNameAnnulus[h] + '_MATERIAL', thickness=None)

## VERTEBRA
for g in range(len(PartModelNameVertebra)):
    # Change Material properties for vertebrae
    NameMatPropVertebra_old.append(PartModelNameVertebra[g] + '_COPY_MATE-
RIAL0')
    NameMatPropVertebra_new.append(PartModelNameVertebra[g] + '_MATERIAL')

    ChangeKeyMatVertebra = mdb.models[PartModelNameVertebra[g]].materi-
als.changeKey(
    fromName=NameMatPropVertebra_old[g], toName=NameMatPropVertebra_new[g])

    # Elastic material properties + Density
    mdb.models[PartModelNameVertebra[g]].materials[NameMatPropVerte-
bra_new[g]].elastic.setValues(
        table=((12000, 0.3),), moduli=LONG_TERM)
    mdb.models[PartModelNameVertebra[g]].materials[NameMatPropVerte-
bra_new[g]].Density(table=((
```

```
1.6e-09,),))

    # Edit section
    EditSectionVertebrae = mdb.models[PartModelNameVertebra[g]].sec-
tions['Section-1-ES_VOLUME0_MAT0'].setValues(
        material=PartModelNameVertebra[g] + '_MATERIAL', thickness=None)

for q in range(len(PartModelName)):
    p1 = mdb.models[PartModelName[q]].parts['PART-1']
    # mdb.models[PartModelName[q]].parts.changeKey(fromName='PART-1', to-
Name=PartModelName[q])
    mdb.models[PartModelName[q]].rootAssembly.regenerate()
    mdb.models[PartModelName[q]].rootAssembly.features.changeKey(from-
Name='PART-1-1',
        toName=PartModelName[q])

    # ----------------------------------------------------------------------
-------------------------------------------------
    ## COPY MODELS INTO ONE MODEL: SPINEMODEL
    mdb.models[modelname].rootAssembly.Instance(name=PartModelName[q] + '-
1', model=mdb.models[PartModelName[q]])

# # ----------------------------------------------------------------------
-----------------------------------------------
## BOUNDARY CONDITIONS
# Create boundary condition encastre
InitialStepName = 'Initial'
regionEncastre = a.allInstances['S1-1'].sets['NS_S1_LATERAL']
EncastreBC = mdb.models[modelname].EncastreBC(name='S1_ENCASTRE', cre-
ateStepName=InitialStepName,
        region=regionEncastre, localCsys=None)

# # ----------------------------------------------------------------------
-----------------------------------------------
# ## CREATE LISTS
# Create empty lists for surface unions
MergeSetInferior_NS, MergeSetSuperior_NS, MergeSetNameInferior_NS, Merge-
SetNameSuperior_NS = [], [], [], []
MergeSurfaceInferior_ES, MergeSurfaceSuperior_ES, MergeSurfaceNameInfe-
rior_ES, MergeSurfaceNameSuperior_ES = [], [], [], []
MergeSetInferior_ES, MergeSetSuperior_ES, MergeSetNameInferior_ES, Merge-
SetNameSuperior_ES = [], [], [], []

# Create empty lists of tie regions
region1Inferior_NS, region2Inferior_NS, region1Superior_NS, region2Supe-
rior_NS = [], [], [], []
region1Inferior_ES, region2Inferior_ES, region1Superior_ES, region2Supe-
rior_ES = [], [], [], []

VertebraeNames = ['C7', 'T1', 'T2', 'T3', 'T4', 'T5', 'T6',
             'T7', 'T8', 'T9', 'T10', 'T11', 'T12',
             'L1', 'L2', 'L3', 'L4', 'L5', 'S1']

# ----------------------------------------------------------------------
-----------------------------------------
## MERGE SURFACES AND SETS
for m in range(len(PartModelNameAnnulus)): ## == len(DiscNames) == 18
    ## MERGE SURFACES OF NODE SETS
    MergeSetNameInferior_NS = 'Disc_'+ DiscNames[m] + '_inferior_NodeSet'
```

```
MergeSetInferior_NS.append(mdb.models[modelname].rootAssembly.SetByBool-
ean(name=MergeSetNameInferior_NS, sets=(
            mdb.models[modelname].rootAssembly.allInstances[PartModelName-
Annulus[m] + '-1'].sets[('NS_' + DiscNames[m] + '_ANNULUS_INFERIOR')],
            mdb.models[modelname].rootAssembly.allInstances[PartModelNa-
meNucleus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_NUCLEUS_INFERIOR')],)))

    MergeSetNameSuperior_NS = 'Disc_' + DiscNames[m] + '_superior_NodeSet'
    MergeSetSuperior_NS.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameSuperior_NS, sets=(
        mdb.models[modelname].rootAssembly.allInstances[PartModelNameAnnu-
lus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_ANNULUS_SUPERIOR')],
        mdb.models[modelname].rootAssembly.allInstances[PartModelNameNu-
cleus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_NUCLEUS_SUPERIOR')],)))

    ## MERGE SURFACES OF ELEMENT SURFACES
    MergeSurfaceNameInferior_ES = 'Disc_' + DiscNames[m] + '_inferior_Ele-
mentSurface'
    MergeSurfaceInferior_ES.append(mdb.models[modelname].rootAssembly.Sur-
faceByBoolean(name=MergeSurfaceNameInferior_ES, surfaces=(
            mdb.models[modelname].rootAssembly.allInstances[PartModelName-
Annulus[m] + '-1'].surfaces[(DiscNames[m] + '_ANNULUS_INFERIOR')],
            mdb.models[modelname].rootAssembly.allInstances[PartModelNa-
meNucleus[m] + '-1'].surfaces[(DiscNames[m] + '_NUCLEUS_INFERIOR')],)))

    MergeSurfaceNameSuperior_ES = 'Disc_' + DiscNames[m] + '_superior_Ele-
mentSurface'
    MergeSurfaceSuperior_ES.append(mdb.models[modelname].rootAssembly.Sur-
faceByBoolean(name=MergeSurfaceNameSuperior_ES, surfaces=(
            mdb.models[modelname].rootAssembly.allInstances[PartModelName-
Annulus[m] + '-1'].surfaces[(DiscNames[m] + '_ANNULUS_SUPERIOR')],
            mdb.models[modelname].rootAssembly.allInstances[PartModelNa-
meNucleus[m] + '-1'].surfaces[(DiscNames[m] + '_NUCLEUS_SUPERIOR')],)))

    ## MERGE SURFACES OF ELEMENT SETS
    MergeSetNameInferior_ES = 'Disc_' + DiscNames[m] + '_inferior_Ele-
mentSet'
    MergeSetInferior_ES.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameInferior_ES, sets=(
            mdb.models[modelname].rootAssembly.allInstances[PartModelName-
Annulus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_ANNULUS_INFERIOR_SET')],
            mdb.models[modelname].rootAssembly.allInstances[PartModelNa-
meNucleus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_NUCLEUS_INFE-
RIOR_SET')])))

    MergeSetNameSuperior_ES = 'Disc_' + DiscNames[m] + '_superior_Ele-
mentSet'
    MergeSetSuperior_ES.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameSuperior_ES, sets=(
            mdb.models[modelname].rootAssembly.allInstances[PartModelName-
Annulus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_ANNULUS_SUPERIOR_SET')],
            mdb.models[modelname].rootAssembly.allInstances[PartModelNa-
meNucleus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_NUCLEUS_SUPE-
RIOR_SET')])))

# ------------------------------------------------------------------------
----------------------------------------
## TIE MERGED ELEMENT SURFACES OF DISCS WITH VERTEBRAE
    region1Superior_ES.append(mdb.models[modelname].rootAssembly.sur-
faces[MergeSurfaceNameSuperior_ES])
```

```
region2Superior_ES.append(mdb.models[modelname].rootAssembly.allIn-
stances[PartModelNameVertebra[m]+ '-1'].surfaces[('ES_' + VertebraeNames[m]
+ '_INFERIOR')])
    TieConstraintSuperior_ES = mdb.models[modelname].Tie(name='Tie_' +
DiscNames[m] + '_Superior_ES', master=region2Superior_ES[m],
                                                    slave=region1Superior_ES[m],
positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
                                                      thickness=ON, constraintEn-
forcement=NODE_TO_SURFACE)

    region1Inferior_ES.append(mdb.models[modelname].rootAssembly.sur-
faces[MergeSurfaceNameInferior_ES])
    region2Inferior_ES.append(mdb.models[modelname].rootAssembly.allIn-
stances[PartModelNameVertebra[m]+ '-1'].surfaces[('ES_' + VertebraeNames[m]
+ '_SUPERIOR')])
    TieConstraintInferior_ES = mdb.models[modelname].Tie(name='Tie_' +
DiscNames[m-1] + '_Inferior_ES', master=region2Inferior_ES[m],
                                                    slave=region1Inferior_ES[m-
1], positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
                                                      thickness=ON, constraintEn-
forcement=NODE_TO_SURFACE)

    # ------------------------------------------------------------------
----------------------------------------------
    ## TIE NODE ELEMENT SURFACES OF VERTEBRAE TO A WHOLE PART

for i in range(len(PartModelNameVertebra)-1):
    WholeVertebraeNodeSet = mdb.models[PartModelNameVertebra[i]].rootAssem-
bly.SetByBoolean(name='NS_' + PartModelNameVertebra[i] + '_WHOLE_VERTEBRA',
sets=(
        mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+ Part-
ModelNameVertebra[i] + '_SUPERIOR'],
        mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+ Part-
ModelNameVertebra[i] +'_LATERAL'],
        mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+ Part-
ModelNameVertebra[i] +'_INFERIOR'],))

# Delete the L5S1_Inferior constraint that incorrectly constraints with the
C7T1_Superior disc
del mdb.models[modelname].constraints['Tie_' + DiscNames[17] + '_Infe-
rior_ES']

# # Manually recreate L5S1_Inferior tie constraint nodeset
# TieConstraintL5S1_Inferior_NS = mdb.models[modelname].Tie(name='Tie_' +
DiscNames[17] + '_Inferior_NS', master = (mdb.models[modelname].rootAssem-
bly.allInstances[PartModelNameVertebra[18] + '-1'].sets[('NS_' + Vertebrae-
Names[18] + '_SUPERIOR')]),
#                                                       slave=region1Infe-
rior_NS[17], positionToleranceMethod=COMPUTED, adjust=OFF, tieRotations=ON,
#                                                         thickness=ON, con-
straintEnforcement=SOLVER_DEFAULT)

# Manually recreate L5S1_Inferior tie constraint elementset
TieConstraintL5S1_Inferior_ES = mdb.models[modelname].Tie(name='Tie_' +
DiscNames[17] + '_Inferior_ES', master=mdb.models[modelname].rootAssem-
bly.allInstances['S1-1'].surfaces['ES_S1_SUPERIOR'],
                                    slave=region1Inferior_ES[17], position-
ToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
                                    thickness=ON, constraintEnforce-
ment=NODE_TO_SURFACE)
```

```
# ------------------------------------------------------------------------
-----------------------------------------
## CREATE COUPLING CONSTRAINT
NodeNameCenter = 'CenterOfC7'
SelectedNode_C7 = mdb.models[modelname].rootAssembly.instances['C7-
1.C7'].nodes.getSequenceFromMask(mask=('[#0:34 #40 ]',), )
region1 = mdb.models[modelname].rootAssembly.Set(nodes=SelectedNode_C7,
name=NodeNameCenter)
region2 = mdb.models[modelname].rootAssembly.allInstances['C7-1'].sur-
faces['ES_C7_SUPERIOR']

Coupling = mdb.models[modelname].Coupling(name='Coupling_C7_Moments', con-
trolPoint=region1,
                                    surface=region2, influenceRa-
dius=WHOLE_SURFACE, couplingType=KINEMATIC,
                                    localCsys=None, u1=ON, u2=ON, u3=ON,
ur1=ON, ur2=ON, ur3=ON)

# ------------------------------------------------------------------------
-----------------------------------------
## CREATE STEP
StepName1 = 'Step_1'
Step1 = mdb.models[modelname].StaticStep(name=StepName1, previous=Ini-
tialStepName,
                                    stabilizationMagnitude=0.0002,
                                    stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
                                    continueDampingFactors=False, adaptive-
DampingRatio=None,
                                    initialInc=0.1, minInc=1e-35, max-
Inc=0.1)

StepName2 = 'Step_2'
Step2 = mdb.models[modelname].StaticStep(name=StepName2, previous=Step-
Name1,
                                    stabilizationMagnitude=0.0002,
                                    stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
                                    continueDampingFactors=False, adaptive-
DampingRatio=None,
                                    initialInc=0.1, minInc=1e-35, max-
Inc=0.1)

# ------------------------------------------------------------------------
-----------------------------------------
## COPY MODELS
Motion = ['Flexion', 'Extension', 'LeftLateralBending', 'RightLateralBend-
ing', 'LeftAxialRotation', 'RightAxialRotation', 'Traction']

# ChangeKey
mdb.models.changeKey(fromName=modelname, toName=modelname + Motion[0])
session.viewports['Viewport: 1'].setValues(displayedObject=None)

for i in range(len(Motion)-1):
    # Copy Models
    mdb.Model(name=modelname + Motion[1+i], objectToCopy=mdb.models[mod-
elname + Motion[0]])
    session.viewports['Viewport: 1'].setValues(displayedObject=None)

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

```
## APPLY MOMENT
regionMomentCM1, MotionCM1, regionMomentCM2, MotionCM2, regionMomentCM3,
MotionCM3 = [], [], [], [], [], []
MotionNameCM1 = Motion[0:1]
MotionNameCM2 = Motion[2:3]
MotionNameCM3 = Motion[4:5]
CM = (10000.0, -10000.0)

FlexionMoment = mdb.models[modelname + Motion[0]].Moment(name=Motion[0],
createStepName=StepName1,
                                        region=mdb.models[modelname + Mo-
tion[0]].rootAssembly.sets[NodeNameCenter], cm1=10000, distribution-
Type=UNIFORM, field='',
                                        localCsys=None)

ExtensionMoment = mdb.models[modelname + Motion[1]].Moment(name=Motion[1],
createStepName=StepName1,
                                        region=mdb.models[modelname + Mo-
tion[1]].rootAssembly.sets[NodeNameCenter], cm1=-10000, distribution-
Type=UNIFORM, field='',
                                        localCsys=None)

LeftLat = mdb.models[modelname + Motion[2]].Moment(name=Motion[2], cre-
ateStepName=StepName1,
                                        region=mdb.models[modelname + Mo-
tion[2]].rootAssembly.sets[NodeNameCenter], cm2=10000, distribution-
Type=UNIFORM, field='',
                                        localCsys=None)

RightLat = mdb.models[modelname + Motion[3]].Moment(name=Motion[3], cre-
ateStepName=StepName1,
                                        region=mdb.models[modelname + Mo-
tion[3]].rootAssembly.sets[NodeNameCenter], cm2=-10000, distribution-
Type=UNIFORM, field='',
                                        localCsys=None)

LeftAR = mdb.models[modelname + Motion[4]].Moment(name=Motion[4], cre-
ateStepName=StepName1,
                                        region=mdb.models[modelname + Mo-
tion[4]].rootAssembly.sets[NodeNameCenter], cm3=10000, distribution-
Type=UNIFORM, field='',
                                        localCsys=None)

RightAR = mdb.models[modelname + Motion[5]].Moment(name=Motion[5], cre-
ateStepName=StepName1,
                                        region=mdb.models[modelname + Mo-
tion[5]].rootAssembly.sets[NodeNameCenter], cm3=-10000, distribution-
Type=UNIFORM, field='',
                                        localCsys=None)

regionTraction = mdb.models[modelname + Motion[6]].rootAssembly.allIn-
stances['C7-1'].sets['NS_C7_WHOLE_VERTEBRA']
mdb.models[modelname + Motion[6]].ConcentratedForce(name=Motion[6],
                                                    createStepName=StepName1,
region=regionTraction, cf3=147.15,
                                                    distributionType=UNIFORM,
field='', localCsys=None)


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

```
## APPLY AXIAL LOAD

for i in range(len(Motion)-1):
    regionAL = mdb.models[modelname + Motion[i]].rootAssembly.allIn-
stances['C7-1'].sets['NS_C7_SUPERIOR']
    mdb.models[modelname + Motion[i]].ConcentratedForce(name='Axial load_'
+ Motion[i],
                                                        createStepName=Step-
Name2, region=regionAL, cf3=-300.0,
                                                        distributionType=UNI-
FORM, field='', localCsys=None)
```

CreateAssembly.py
Creates the assembly of the Spine model

```
1    import…
36
37   time_start = time.clock()
38
39   # Change modelname
40   modelname = 'Spine model'
41   mdb.models.changeKey(fromName='Model-1', toName=modelname)
42
43   a = mdb.models[modelname].rootAssembly
44
45   # ------------------------------------------------------------------------
-----------------------------------------------
46   ## NAME PARTS
47   # List with names of all 55 models
48   PartModelName = ['C7', 'C7T1_Annulus', 'C7T1_Nucleus',
49                    'T1', 'T1T2_Annulus', 'T1T2_Nucleus',
50                    'T2', 'T2T3_Annulus', 'T2T3_Nucleus',
51                    'T3', 'T3T4_Annulus', 'T3T4_Nucleus',
52                    'T4', 'T4T5_Annulus', 'T4T5_Nucleus',
53                    'T5', 'T5T6_Annulus', 'T5T6_Nucleus',
54                    'T6', 'T6T7_Annulus', 'T6T7_Nucleus',
55                    'T7', 'T7T8_Annulus', 'T7T8_Nucleus',
56                    'T8', 'T8T9_Annulus', 'T8T9_Nucleus',
57                    'T9', 'T9T10_Annulus', 'T9T10_Nucleus',
58                    'T10', 'T10T11_Annulus', 'T10T11_Nucleus',
59                    'T11', 'T11T12_Annulus', 'T11T12_Nucleus',
60                    'T12', 'T12L1_Annulus', 'T12L1_Nucleus',
61                    'L1', 'L1L2_Annulus', 'L1L2_Nucleus',
62                    'L2', 'L2L3_Annulus', 'L2L3_Nucleus',
63                    'L3', 'L3L4_Annulus', 'L3L4_Nucleus',
64                    'L4', 'L4L5_Annulus', 'L4L5_Nucleus',
65                    'L5', 'L5S1_Annulus', 'L5S1_Nucleus',
66                    'S1']
67
68   DiscNames = ['C7T1', 'T1T2','T2T3', 'T3T4', 'T4T5', 'T5T6', 'T6T7',
'T7T8', 'T8T9', 'T9T10', 'T10T11', 'T11T12', 'T12L1', 'L1L2', 'L2L3',
'L3L4', 'L4L5', 'L5S1']
69
70   # Individual names per modeltype
71   PartModelNameVertebra = (PartModelName[0::3]) # 19 parts
72   PartModelNameAnnulus = (PartModelName[1::3]) # 18 parts
73   PartModelNameNucleus = (PartModelName[2::3]) # 18 parts
74
75   # ------------------------------------------------------------------------
-----------------------------------------------
76   ## IMPORT ALL MODELS INTO ABAQUS
77   for i in range(len(PartModelName)):
78       # Import inp partmodel files
79           ShowSession = session.viewports['Viewport: 1'].setValues(dis-
playedObject=a)
80           mdb.ModelFromInputFile(name=PartModelName[i],
81           inputFileName = ('C:/1. Abaqus tijdelijk/220322 ABAQUS MODEL-
LEN EN SCRIPTS/inp parts/' + PartModelName[i] + '.inp'))
82
83   # ------------------------------------------------------------------------
-----------------------------------------------
84   ## Create element sets of element surfaces
```

```
85    SurfaceAnnulusInferior_ES, SurfaceSetInferiorAnnulus_ES, SurfaceAnnu-
lusSuperior_ES, SurfaceSetSuperiorAnnulus_ES, \
86    SurfaceNucleusInferior_ES, SurfaceSetInferiorNucleus_ES, Surface-
NucleusSuperior_ES, SurfaceSetSuperiorNucleus_ES= [], [], [], [], [], [],
[], []
87    Surface_Vertebra_Inferior_ES, SurfaceSet_Vertebra_Inferior_ES = [], []
88
89    for k in range(len(PartModelNameAnnulus)):
90        SurfaceAnnulusInferior_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.surfaces[DiscNames[k] + '_ANNULUS_INFERIOR'].elements
91        SurfaceSetInferiorAnnulus_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.Set(elements=SurfaceAnnulusInferior_ES, name='ES_'+
DiscNames[k] + '_ANNULUS_INFERIOR_SET')
92
93        SurfaceAnnulusSuperior_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.surfaces[DiscNames[k] + '_ANNULUS_SUPERIOR'].elements
94        SurfaceSetSuperiorAnnulus_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.Set(elements=SurfaceAnnulusSuperior_ES, name='ES_' +
DiscNames[k] + '_ANNULUS_SUPERIOR_SET')
95
96        SurfaceNucleusInferior_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.surfaces[DiscNames[k] + '_NUCLEUS_INFERIOR'].ele-
ments
97        SurfaceSetInferiorNucleus_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.Set(elements=SurfaceNucleusInferior_ES, name='ES_'+
DiscNames[k] + '_NUCLEUS_INFERIOR_SET')
98
99        SurfaceNucleusSuperior_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.surfaces[DiscNames[k] + '_NUCLEUS_SUPERIOR'].ele-
ments
100       SurfaceSetSuperiorNucleus_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.Set(elements=SurfaceNucleusSuperior_ES, name='ES_' +
DiscNames[k] + '_NUCLEUS_SUPERIOR_SET')
101
102   for k in range(len(PartModelNameVertebra)-1):
103       Surface_Vertebra_Inferior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.surfaces['ES_' + PartModelNameVertebra[k] + '_INFERI-
OR'].elements
104       SurfaceSet_Vertebra_Inferior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.Set(elements=Surface_Vertebra_Inferior_ES, name='ES_'+
PartModelNameVertebra[k] + '_INFERIOR_SET')
105
106   for k in range(len(PartModelNameVertebra)):
107       Surface_Vertebra_Superior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.surfaces['ES_' + PartModelNameVertebra[k] + '_SUPERI-
OR'].elements
108       SurfaceSet_Vertebra_Superior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.Set(elements=Surface_Vertebra_Superior_ES, name='ES_'+
PartModelNameVertebra[k] + '_SUPERIOR_SET')
109
110   # ------------------------------------------------------------------------
-------------------------------------------------
111   ## Change material properties and edit section
112   ChangeMatPropNucleus = []
113   ChangeMatPropAnnulus = []
114   ChangeMatPropVertebrae = []
115
116   for h in range(len(DiscNames)):
117       # Change Material properties for nucleus
118       ChangeMatPropNucleus = mdb.models[PartModelNameNucleus[h]].materi-
als[DiscNames[h] + '_NUCLEUS_COPY_MATERIAL0'].elastic.setValues(
```

```
119           table=((1.0, 0.45),), moduli=LONG_TERM)
120
121       ChangeKeyMatPropNucleus = mdb.models[PartModelNameNucleus[h]].ma-
terials.changeKey(
122           fromName=DiscNames[h] + '_NUCLEUS_COPY_MATERIAL0', to-
Name=PartModelNameNucleus[h] + '_MATERIAL')
123
124       # Edit section
125       EditSectionNucleus = mdb.models[PartModelNameNucleus[h]].sec-
tions['Section-1-ES_VOLUME0_MAT0'].setValues(
126           material=PartModelNameNucleus[h] + '_MATERIAL', thick-
ness=None)
127
128       # Change Material properties for annulus
129       ChangeMatPropAnnulus = mdb.models[PartModelNameAnnulus[h]].materi-
als[DiscNames[h] + '_ANNULUS_COPY_MATERIAL0'].elastic.setValues(
130           table=((4.2, 0.45),), moduli=LONG_TERM)
131
132       ChangeKeyMatPropAnnulus = mdb.models[PartModelNameAnnulus[h]].ma-
terials.changeKey(
133           fromName=DiscNames[h] + '_ANNULUS_COPY_MATERIAL0', to-
Name=PartModelNameAnnulus[h] + '_MATERIAL')
134
135       # Edit section
136       EditSectionAnnulus = mdb.models[PartModelNameAnnulus[h]].sec-
tions['Section-1-ES_VOLUME0_MAT0'].setValues(
137           material=PartModelNameAnnulus[h] + '_MATERIAL', thick-
ness=None)
138
139   for g in range(len(PartModelNameVertebra)):
140       # Change Material properties for vertebrae
141       ChangeMatPropVertebrae = mdb.models[PartModelNameVertebra[g]].ma-
terials[PartModelNameVertebra[g] + '_COPY_MATERIAL0'].elastic.setValues(
142           table=((12000, 0.3),), moduli=LONG_TERM)
143
144       ChangeKeyMatPropVertebrae =
mdb.models[PartModelNameVertebra[g]].materials.changeKey(
145           fromName=PartModelNameVertebra[g] + '_COPY_MATERIAL0', to-
Name=PartModelNameVertebra[g] + '_MATERIAL')
146
147       # Edit section
148       EditSectionVertebrae = mdb.models[PartModelNameVertebra[g]].sec-
tions['Section-1-ES_VOLUME0_MAT0'].setValues(
149           material=PartModelNameVertebra[g] + '_MATERIAL', thick-
ness=None)
150
151   for q in range(len(PartModelName)):
152       p1 = mdb.models[PartModelName[q]].parts['PART-1']
153       # mdb.models[PartModelName[q]].parts.changeKey(fromName='PART-1',
toName=PartModelName[q])
154       mdb.models[PartModelName[q]].rootAssembly.regenerate()
155       mdb.models[PartModelName[q]].rootAssembly.features.changeKey(from-
Name='PART-1-1',
156           toName=PartModelName[q])
157
158       # ----------------------------------------------------------------
----------------------------------------------------
159       ## COPY MODELS INTO ONE MODEL: SPINE MODEL
160       mdb.models[modelname].rootAssembly.Instance(name=PartModelName[q]
+ '-1', model=mdb.models[PartModelName[q]])
161
```

```
162  # # ------------------------------------------------------------------------
------------------------------------------------
163  ## BOUNDARY CONDITIONS
164  # Create boundary condition encastre
165  InitialStepName = 'Initial'
166  regionEncastre = a.allInstances['S1-1'].sets['NS_S1_LATERAL']
167  EncastreBC = mdb.models[modelname].EncastreBC(name='S1_ENCASTRE', cre-
ateStepName=InitialStepName,
168          region=regionEncastre, localCsys=None)
169
170  # # ------------------------------------------------------------------------
------------------------------------------------
171  # ## CREATE LISTS
172  # Create empty lists for surface unions
173  MergeSetInferior_NS, MergeSetSuperior_NS, MergeSetNameInferior_NS,
MergeSetNameSuperior_NS = [], [], [], []
174  MergeSurfaceInferior_ES, MergeSurfaceSuperior_ES, MergeSurfaceNameIn-
ferior_ES, MergeSurfaceNameSuperior_ES = [], [], [], []
175  MergeSetInferior_ES, MergeSetSuperior_ES, MergeSetNameInferior_ES,
MergeSetNameSuperior_ES = [], [], [], []
176
177  # Create empty lists of tie regions
178  region1Inferior_NS, region2Inferior_NS, region1Superior_NS, region2Su-
perior_NS = [], [], [], []
179  region1Inferior_ES, region2Inferior_ES, region1Superior_ES, region2Su-
perior_ES = [], [], [], []
180
181  VertebraeNames = ['C7', 'T1', 'T2', 'T3', 'T4', 'T5', 'T6',
182               'T7', 'T8', 'T9', 'T10', 'T11', 'T12',
183               'L1', 'L2', 'L3', 'L4', 'L5', 'S1']
184
185  # ------------------------------------------------------------------------
------------------------------------------------
186  ## MERGE SURFACES AND SETS
187  for m in range(len(PartModelNameAnnulus)): ## == len(DiscNames) == 18
188      ## MERGE SURFACES OF NODE SETS
189      MergeSetNameInferior_NS = 'Disc_'+ DiscNames[m] + '_infe-
rior_NodeSet'
190      MergeSetInferior_NS.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameInferior_NS, sets=(
191              mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].sets[('NS_' + DiscNames[m] + '_ANNULUS_INFERI-
OR')],
192              mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_NUCLEUS_INFERI-
OR')],)))
193
194      MergeSetNameSuperior_NS = 'Disc_' + DiscNames[m] + '_supe-
rior_NodeSet'
195      MergeSetSuperior_NS.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameSuperior_NS, sets=(
196          mdb.models[modelname].rootAssembly.allInstances[PartModelName-
Annulus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_ANNULUS_SUPERIOR')],
197          mdb.models[modelname].rootAssembly.allInstances[PartModelNa-
meNucleus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_NUCLEUS_SUPERIOR')],)))
198
199      ## MERGE SURFACES OF ELEMENT SURFACES
200      MergeSurfaceNameInferior_ES = 'Disc_' + DiscNames[m] + '_infe-
rior_ElementSurface'
201      MergeSurfaceInferior_ES.append(mdb.models[modelname].rootAssem-
bly.SurfaceByBoolean(name=MergeSurfaceNameInferior_ES, surfaces=(
```

```
202             mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].surfaces[(DiscNames[m] + '_ANNULUS_INFERIOR')],
203             mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m] + '-1'].surfaces[(DiscNames[m] + '_NUCLEUS_INFERIOR')],)))
204
205     MergeSurfaceNameSuperior_ES = 'Disc_' + DiscNames[m] + '_supe-
rior_ElementSurface'
206     MergeSurfaceSuperior_ES.append(mdb.models[modelname].rootAssem-
bly.SurfaceByBoolean(name=MergeSurfaceNameSuperior_ES, surfaces=(
207             mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].surfaces[(DiscNames[m] + '_ANNULUS_SUPERIOR')],
208             mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m] + '-1'].surfaces[(DiscNames[m] + '_NUCLEUS_SUPERIOR')],)))
209
210     ## MERGE SURFACES OF ELEMENT SETS
211     MergeSetNameInferior_ES = 'Disc_' + DiscNames[m] + '_inferior_Ele-
mentSet'
212     MergeSetInferior_ES.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameInferior_ES, sets=(
213             mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_ANNULUS_INFE-
RIOR_SET')],
214             mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_NUCLEUS_INFE-
RIOR_SET')])))
215
216     MergeSetNameSuperior_ES = 'Disc_' + DiscNames[m] + '_superior_Ele-
mentSet'
217     MergeSetSuperior_ES.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameSuperior_ES, sets=(
218             mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_ANNULUS_SUPE-
RIOR_SET')],
219             mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_NUCLEUS_SUPE-
RIOR_SET')])))
220
221  # ------------------------------------------------------------------------
-------------------------------------------------
222  ## TIE MERGED ELEMENT SURFACES OF DISCS WITH VERTEBRAE
223     region1Superior_ES.append(mdb.models[modelname].rootAssembly.sur-
faces[MergeSurfaceNameSuperior_ES])
224     region2Superior_ES.append(mdb.models[modelname].rootAssembly.al-
lInstances[PartModelNameVertebra[m]+ '-1'].surfaces[('ES_' + Vertebrae-
Names[m] + '_INFERIOR')])
225     TieConstraintSuperior_ES = mdb.models[modelname].Tie(name='Tie_' +
DiscNames[m] + '_Superior_ES', master=region2Superior_ES[m],
226                                                 slave=region1Supe-
rior_ES[m], positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
227                                                   thickness=ON, con-
straintEnforcement=NODE_TO_SURFACE)
228
229     region1Inferior_ES.append(mdb.models[modelname].rootAssembly.sur-
faces[MergeSurfaceNameInferior_ES])
230     region2Inferior_ES.append(mdb.models[modelname].rootAssembly.al-
lInstances[PartModelNameVertebra[m]+ '-1'].surfaces[('ES_' + Vertebrae-
Names[m] + '_SUPERIOR')])
231     TieConstraintInferior_ES = mdb.models[modelname].Tie(name='Tie_' +
DiscNames[m-1] + '_Inferior_ES', master=region2Inferior_ES[m],
232                                                 slave=region1Infe-
rior_ES[m-1], positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
```

```
233                                                     thickness=ON, con-
straintEnforcement=NODE_TO_SURFACE)
234
235       # ------------------------------------------------------------------
----------------------------------------------------
236       ## TIE NODE ELEMENT SURFACES OF VERTABRAE TO A WHOLE PART
237
238   for i in range(len(PartModelNameVertebra)-1):
239       WholeVertebraeNodeSet = mdb.models[PartModelNameVerte-
bra[i]].rootAssembly.SetByBoolean(name='NS_' + PartModelNameVertebra[i] +
'_WHOLE_VERTEBRA', sets=(
240           mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+
PartModelNameVertebra[i] + '_SUPERIOR'],
241           mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+
PartModelNameVertebra[i] +'_LATERAL'],
242           mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+
PartModelNameVertebra[i] +'_INFERIOR'],))
243
244   # Delete the L5S1_Inferior constraint that incorrectly constraints
with the C7T1_Superior disc
245   del mdb.models[modelname].constraints['Tie_' + DiscNames[17] + '_Infe-
rior_ES']
246
247   # # Manually recreate L5S1_Inferior tie constraint nodeset
248   # TieConstraintL5S1_Inferior_NS = mdb.models[mod-
elname].Tie(name='Tie_' + DiscNames[17] + '_Inferior_NS', master =
(mdb.models[modelname].rootAssembly.allInstances[PartModelNameVertebra[18]
+ '-1'].sets[('NS_' + VertebraeNames[18] + '_SUPERIOR')]),
249   #                                                       slave=re-
gion1Inferior_NS[17], positionToleranceMethod=COMPUTED, adjust=OFF, tieRo-
tations=ON,
250   #                                                   thickness=ON, con-
straintEnforcement=SOLVER_DEFAULT)
251
252   # Manually recreate L5S1_Inferior tie constraint elementset
253   TieConstraintL5S1_Inferior_ES = mdb.models[modelname].Tie(name='Tie_'
+ DiscNames[17] + '_Inferior_ES', master=mdb.models[modelname].rootAssem-
bly.allInstances['S1-1'].surfaces['ES_S1_SUPERIOR'],
254                                       slave=region1Inferior_ES[17], po-
sitionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
255                                       thickness=ON, constraintEnforce-
ment=NODE_TO_SURFACE)
256
257   # ------------------------------------------------------------------
----------------------------------------------
258   ## CREATE POLYLINE ORIGINAL SHAPE
259
260   CoG_inv =
([12.64,-103.16,-850.11],
261
[13.88,-109.17,-847.84],
262
[19.93,-125.42,-830.94],
263
[25.57,-128.00,-821.66],
264
[29.51,-132.61,-799.24],
265
[30.28,-132.59,-788.90],
266
[31.69,-131.64,-765.58],
```

```
267
[30.81,-130.25,-756.60],
268
[26.74,-128.55,-733.28],
269
[23.76,-127.07,-726.18],
270
[14.49,-125.61,-705.78],
271
[10.02,-125.42,-699.69],
272
[-3.58,-121.55,-682.12],
273
[-7.26,-120.79,-678.57],
274
[-20.26,-117.08,-662.46],
275
[-23.83,-115.83,-657.57],
276
[-32.76,-112.70,-641.56],
277
[-35.04,-111.19,-635.92],
278
[-39.77,-108.27,-619.32],
279
[-39.56,-107.92,-613.33],
280
[-38.32,-104.46,-597.46],
281
[-36.96,-103.89,-592.08],
282
[-31.74,-103.20,-577.76],
283
[-28.78,-103.09,-573.48],
284
[-20.86,-103.19,-560.15],
285
[-16.93,-104.07,-557.18],
286
[-6.06,-105.00,-545.56],
287
[-2.09,-104.69,-542.44],
288
[8.45,-107.28,-530.83],
289
[10.61,-107.76,-526.99],
290
[16.06,-110.37,-514.10],
291
[15.83,-112.65,-509.99],
292
[17.57,-117.35,-495.74],
293
[16.77,-118.81,-492.11],
294
[13.63,-125.33,-477.85],
295
[13.45,-127.39,-474.95],
296
[12.92,-130.37,-462.26])
297
```

```
298
299  # Reverse the CoG list (
300  CoG = list(reversed(CoG_inv))
301
302
303  # Select only the coordinates that make the centroid of the vertebral
bodies
304  CoGVertebralCentroid = (CoG[0::2])
305
306  PolyLineInitial = mdb.models[modelname].rootAssembly.WirePol-
yLine(points=([CoG[0][0], CoG[0][1], CoG[0][2]],
307
[CoG[1][0], CoG[1][1], CoG[1][2]],
308
[CoG[2][0], CoG[2][1], CoG[2][2]],
309
[CoG[3][0], CoG[3][1], CoG[3][2]],
310
[CoG[4][0], CoG[4][1], CoG[4][2]],
311
[CoG[5][0], CoG[5][1], CoG[5][2]],
312
[CoG[6][0], CoG[6][1], CoG[6][2]],
313
[CoG[7][0], CoG[7][1], CoG[7][2]],
314
[CoG[8][0], CoG[8][1], CoG[8][2]],
315
[CoG[9][0], CoG[9][1], CoG[9][2]],
316
[CoG[10][0], CoG[10][1], CoG[10][2]],
317
[CoG[11][0], CoG[11][1], CoG[11][2]],
318
[CoG[12][0], CoG[12][1], CoG[12][2]],
319
[CoG[13][0], CoG[13][1], CoG[13][2]],
320
[CoG[14][0], CoG[14][1], CoG[14][2]],
321
[CoG[15][0], CoG[15][1], CoG[15][2]],
322
[CoG[16][0], CoG[16][1], CoG[16][2]],
323
[CoG[17][0], CoG[17][1], CoG[17][2]],
324
[CoG[18][0], CoG[18][1], CoG[18][2]],
325
[CoG[19][0], CoG[19][1], CoG[19][2]],
326
[CoG[20][0], CoG[20][1], CoG[20][2]],
327
[CoG[21][0], CoG[21][1], CoG[21][2]],
328
[CoG[22][0], CoG[22][1], CoG[22][2]],
329
[CoG[23][0], CoG[23][1], CoG[23][2]],
330
[CoG[24][0], CoG[24][1], CoG[24][2]],
331
[CoG[25][0], CoG[25][1], CoG[25][2]],
```

```
332
[CoG[26][0], CoG[26][1], CoG[26][2]],
333
[CoG[27][0], CoG[27][1], CoG[27][2]],
334
[CoG[28][0], CoG[28][1], CoG[28][2]],
335
[CoG[29][0], CoG[29][1], CoG[29][2]],
336
[CoG[30][0], CoG[30][1], CoG[30][2]],
337
[CoG[31][0], CoG[31][1], CoG[31][2]],
338
[CoG[32][0], CoG[32][1], CoG[32][2]],
339
[CoG[33][0], CoG[33][1], CoG[33][2]],
340
[CoG[34][0], CoG[34][1], CoG[34][2]],
341
[CoG[35][0], CoG[35][1], CoG[35][2]],
342
[CoG[36][0], CoG[36][1], CoG[36][2]]), mergeType= IMPRINT, meshable=OFF)
343
344  ChangeNamePolyline1 = mdb.models[modelname].rootAssembly.fea-
tures.changeKey(fromName='Wire-1',
345                                                              toName='Ini-
tial Shape')
346
347  # ------------------------------------------------------------------
----------------------------------------------
348  # CREATE COG OF VERTEBRAL BODIES
349  print(len(CoG))
350  Points, ChangeKeyDP, NearestNode, CoGVertebralBodies, DatumCoordi-
natesCoG, DatumCoordinatesCoG_x, DatumCoordinatesCoG_y, DatumCoordinates-
CoG_z, FindNearestNodes, n1,SphereNodes, CreateSetsofVertebralCoG
=[],[],[], [], [], [], [], [], [], [], [], []
351  for i in range(len(PartModelNameVertebra)-1):
352      Points.append(mdb.models[PartModelNameVertebra[i]].rootAssem-
bly.DatumPointByMidPoint(point1=[CoGVertebralCentroid[i][0], CoGVerte-
bralCentroid[i][1], CoGVertebralCentroid[i][2]], point2=[CoGVertebralCen-
troid[(i+1)][0], CoGVertebralCentroid[(i+1)][1], CoGVertebralCen-
troid[(i+1)][2]]))
353      ChangeKeyDP.append(mdb.models[PartModelNameVertebra[i]].rootAssem-
bly.features.changeKey(fromName='Datum pt-1',
354                                                                  to-
Name="CoG_"+ PartModelNameVertebra[i]))
355
356      CoGVertebralBodies.append(mdb.models[PartModelNameVerte-
bra[i]].rootAssembly.features['CoG_' + PartModelNameVertebra[i]].id)
357      print('CoGVertebralBodies', CoGVertebralBodies)
358
359      DatumCoordinatesCoG.append(mdb.models[PartModelNameVerte-
bra[i]].rootAssembly.datums[CoGVertebralBodies[i]].pointOn)
360      print('DatumCoordinatesCoG', DatumCoordinatesCoG)
361
362      # Split coordinates into separate x, y and z arrays
363      DatumCoordinatesCoG_x.append(DatumCoordinatesCoG[i][0])
364      DatumCoordinatesCoG_y.append(DatumCoordinatesCoG[i][1])
365      DatumCoordinatesCoG_z.append(DatumCoordinatesCoG[i][2])
366
```

```
367  # ------------------------------------------------------------------------
---------------------------------------------
368  # FIND NEAREST NODE TO COG OF VERTEBRAL BODIES
369      GetClosest = []
370      # Go to mesh module
371      session.viewports['Viewport: 1'].setValues(displayedOb-
ject=mdb.models[PartModelNameVertebra[i]].rootAssembly)
372      session.viewports['Viewport: 1'].assemblyDisplay.set-
Values(mesh=ON)
373      session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.set-
Values(
374          meshTechnique=ON)
375
376      import sys
377      sys.path.insert(8,
378          r'c:/SIMULIA/EstProducts/2020/win_b64/code/py-
thon2.7/lib/abaqus_plugins/findNearestNode')
379      import nearestNodeModule
380
381      n1.append(mdb.models[PartModelNameVertebra[i]].rootAssembly.in-
stances[PartModelNameVertebra[i]].nodes)
382      SphereNodes.append(mdb.models[PartModelNameVertebra[i]].rootAssem-
bly.instances[PartModelNameVertebra[i]].nodes.getByBoundingSphere(center =
DatumCoordinatesCoG[i], radius = 5))
383      print('SphereNodes', SphereNodes)
384
385      # Find nearest nodes to find vertebral body centroids, gives near-
est node label, instance name, distance to coordinates and coordinates of
the node.
386      FindNearestNodes.append(nearestNodeModule.findNearest-
Node(xcoord=DatumCoordinatesCoG_x[i], ycoord=DatumCoordinatesCoG_y[i],
387          zcoord=DatumCoordinatesCoG_z[i], name=''))
388
389      session.viewports['Viewport: 1'].setValues(displayedOb-
ject=mdb.models[PartModelNameVertebra[i]].rootAssembly)
390
391      # Show location of node
392      nearestNodeModule.showTextAndArrow()
393
394      session.viewports['Viewport: 1'].setValues(displayedOb-
ject=mdb.models['Spine model'].rootAssembly)
395      session.viewports['Viewport: 1'].view.setValues(ses-
sion.views['Bottom'])
396
397
398      CreateSetsofVertebralCoG.append(mdb.models[PartModelNameVerte-
bra[i]].rootAssembly.Set(nodes=mdb.models[PartModelNameVertebra[i]].rootAs-
sembly.instances[PartModelNameVertebra[i]].nodes.getByBoundingSphere(center
= DatumCoordinatesCoG[i], radius = 6), name="Vertebral_CoG_Set_" + PartMod-
elNameVertebra[i]))
399
400
401  # ------------------------------------------------------------------------
---------------------------------------------
402  # CREATE SCREW SETS WITH REFERENCE POINTS
403
404  # Input map of the xml file
405  mypath = r'C:\1. Abaqus tijdelijk\220131 Automated model with steps
displacement\CurvesAndPrimitives.xml'
406
407  # Location of the xml file
```

```
408  tree = ET.parse(mypath)
409
410  # Get the root of the xml file
411  root = tree.getroot()
412
413  # Loop to create points from xml file (37 rows, 3 columns)
414  # for i in range(36):
415  for i in range(len(root)):
416      Point1 = root[i][1].text.split()  # Second column
417      Point2 = root[i][2].text.split()  # Third column
418
419      # Create datumpoints
420      P1 = mdb.models[modelname].rootAssembly.DatumPointByCoordinate((
421          float(Point1[0]), float(Point1[1]), float(Point1[2])))
422      P2 = mdb.models[modelname].rootAssembly.DatumPointByCoordinate((
423          float(Point2[0]), float(Point2[1]), float(Point2[2])))
424
425      # Give name to points
426      Names = root[i][0].text.split()  # First column
427
428      Ppoint1 = mdb.models[modelname].rootAssembly.fea-
tures.changeKey(fromName=P1.name, toName=str(Names) + str('start'))
429      Ppoint2 = mdb.models[modelname].rootAssembly.fea-
tures.changeKey(fromName=P2.name, toName=str(Names) + str('end'))
430
431      # Create vector line through start- and endpoint
432      FirstLine = mdb.models[modelname].rootAssembly.DatumAxisByTwo-
Point(point1=
433
mdb.models[modelname].rootAssembly.datums[P1.id],
434
point2=
435
mdb.models[modelname].rootAssembly.datums[P2.id])
436
437      Line = mdb.models[modelname].rootAssembly.features.changeKey(from-
Name=FirstLine.name,
438                                                                   to-
Name=str(Names) + str('vector'))
439
440      # Create reference point
441      RFid = mdb.models[modelname].rootAssembly.ReferencePoint(
442          point=(float(Point1[0]), float(Point1[1]),
float(Point1[2]))).id
443      r1 = mdb.models[modelname].rootAssembly.referencePoints
444      refPoints1 = (r1[RFid],)
445      region = regionToolset.Region(referencePoints=refPoints1)
446
447      AllNames = ['T3-1.T3', 'T3-1.T3',
448                  "T4-1.T4", "T4-1.T4",
449                  "T5-1.T5", "T5-1.T5",
450                  "T6-1.T6", "T6-1.T6",
451                  "T7-1.T7", "T7-1.T7",
452                  "T8-1.T8",  "T8-1.T8",
453                  "T9-1.T9",  "T9-1.T9",
454                  "T10-1.T10", "T10-1.T10",
455                  "T11-1.T11", "T11-1.T11",
456                  "T12-1.T12","T12-1.T12",
457                  "L1-1.L1", "L1-1.L1",
458                  "L2-1.L2",  "L2-1.L2",
459                  "L3-1.L3", "L3-1.L3",
```

```
460                   "L4-1.L4", "L4-1.L4",
461                   "L5-1.L5",  "L5-1.L5",
462                   "T2-1.T2",  "T2-1.T2",
463                   "T1-1.T1",   "T1-1.T1",
464                   "C7-1.C7", "C7-1.C7" ]
465
466       # Set parameters to create a cylinder
467       r2 = 1.5  # Radius of screw [mm] # Creating a cylinder around the
screw points and reference point
468       r3 = 4 # 2.6
469       s1 = float(Point1[0]), float(Point1[1]), float(Point1[2])  #
Starting point of screw [x,y,z]
470       s2 = float(Point2[0]), float(Point2[1]), float(Point2[2])  # End
point of screw [x,y,z]
471
472       # Create empty lists
473       ScrewsCylinder, ScrewsCylinderES, SetRefPoints, SetRefPointsES =
[], [], [], []
474
475       # Create nodeset of cylinders
476       ScrewsCylinder.append(mdb.models[modelname].rootAssem-
bly.Set(name='Screw' + str(Names), nodes=[
477           mdb.models[modelname].rootAssembly.in-
stances[AllNames[i]].nodes.getByBoundingCylinder(s1, s2, r2), ], ))
478
479       # Create set of reference points
480       SetRefPoints.append(mdb.models[modelname].rootAssem-
bly.Set(name='RF' + str(Names), referencePoints=(
481           mdb.models[modelname].rootAssembly.referencePoints[RFid],)))
482
483       # Coupling nodeset with reference points
484       mdb.models[modelname].Coupling(controlPoint=
485                                      mdb.models[modelname].rootAssem-
bly.sets['RF' + str(Names)],
486                                      couplingType=KINEMATIC
487                                      , influenceRadius=WHOLE_SURFACE,
localCsys=None, name='Coupling' + str(Names),
488                                      surface=mdb.models[mod-
elname].rootAssembly.sets['Screw' + str(Names)], u1=ON,
489                                      u2=ON,
490                                      u3=ON, ur1=ON, ur2=ON, ur3=ON)
491
492       # Create elementset of cylinders
493       ScrewsCylinderES.append(mdb.models[modelname].rootAssem-
bly.Set(name='Screw_ES_' + str(Names), nodes=[
494           mdb.models[modelname].rootAssembly.instances[AllNames[i]].ele-
ments.getByBoundingCylinder(s1, s2, r3), ], ))
495
496   # ---------------------------------------------------------------------
-----------------------------------------------
497   ## VIEWPORT AND COMPUTATION TIME
498
499   # Select right view (z,x plane)
500   session.viewports['Viewport: 1'].setValues(displayedObject=mdb.mod-
els[modelname].rootAssembly)
501   session.viewports['Viewport: 1'].view.setValues(session.views['Bot-
tom'])
502
503   # COMPUTATION TIME
504   time_elapsed = (time.clock() - time_start)
505   print('Time elapsed =', time_elapsed, 'seconds')
```

```
506
507
```

ValidateLinearModel.py
Validates the created model.

```
1     import section
2     import regionToolset
3     import displayGroupMdbToolset as dgm
4     import part
5     import material
6     import assembly
7     import step
8     import interaction
9     import load
10    import mesh
11    import optimization
12    import job
13    import sketch
14    import visualization
15    import xyPlot
16    import displayGroupOdbToolset as dgo
17    import connectorBehavior
18    import part
19    import assembly
20    import material
21    import section
22    import interaction
23    import numpy as np
24    import time
25    from symbolicConstants import *
26    from abaqusConstants import *
27    import itertools
28    import sys
29    import xml.etree.ElementTree as ET
30    import regionToolset  # to add reference points
31    import interaction  # to include constraints
32    import subprocess  # to restart program
33    import time
34    from os import listdir
35    from os.path import isfile, join
36
37    # Change modelname
38    modelname = 'Spine model'
39    mdb.models.changeKey(fromName='Model-1', toName=modelname)
40
41    a = mdb.models[modelname].rootAssembly
42
43    # ------------------------------------------------------------------
------------------------------------------------
44    ## NAME PARTS
45    # List with names of all 55 models
46    PartModelName = ['C7', 'C7T1_Annulus', 'C7T1_Nucleus',
47                     'T1', 'T1T2_Annulus', 'T1T2_Nucleus',
48                     'T2', 'T2T3_Annulus', 'T2T3_Nucleus',
49                     'T3', 'T3T4_Annulus', 'T3T4_Nucleus',
50                     'T4', 'T4T5_Annulus', 'T4T5_Nucleus',
51                     'T5', 'T5T6_Annulus', 'T5T6_Nucleus',
52                     'T6', 'T6T7_Annulus', 'T6T7_Nucleus',
53                     'T7', 'T7T8_Annulus', 'T7T8_Nucleus',
54                     'T8', 'T8T9_Annulus', 'T8T9_Nucleus',
```

```
55                      'T9', 'T9T10_Annulus', 'T9T10_Nucleus',
56                      'T10', 'T10T11_Annulus', 'T10T11_Nucleus',
57                      'T11', 'T11T12_Annulus', 'T11T12_Nucleus',
58                      'T12', 'T12L1_Annulus', 'T12L1_Nucleus',
59                      'L1', 'L1L2_Annulus', 'L1L2_Nucleus',
60                      'L2', 'L2L3_Annulus', 'L2L3_Nucleus',
61                      'L3', 'L3L4_Annulus', 'L3L4_Nucleus',
62                      'L4', 'L4L5_Annulus', 'L4L5_Nucleus',
63                      'L5', 'L5S1_Annulus', 'L5S1_Nucleus',
64                      'S1']
65
66   DiscNames = ['C7T1', 'T1T2','T2T3', 'T3T4', 'T4T5', 'T5T6', 'T6T7',
'T7T8', 'T8T9', 'T9T10', 'T10T11', 'T11T12', 'T12L1', 'L1L2', 'L2L3',
'L3L4', 'L4L5', 'L5S1']
67
68   # Individual names per modeltype
69   PartModelNameVertebra = (PartModelName[0::3]) # 19 parts
70   PartModelNameAnnulus = (PartModelName[1::3]) # 18 parts
71   PartModelNameNucleus = (PartModelName[2::3]) # 18 parts
72
73   # ----------------------------------------------------------------------
----------------------------------------------
74   ## IMPORT ALL MODELS INTO ABAQUS
75   for i in range(len(PartModelName)):
76       # Import inp partmodel files
77           ShowSession = session.viewports['Viewport: 1'].setValues(dis-
playedObject=a)
78           mdb.ModelFromInputFile(name=PartModelName[i],
79           inputFileName = ('C:/1. Abaqus tijdelijk/220322 ABAQUS MODEL-
LEN EN SCRIPTS/inp parts/' + PartModelName[i] + '.inp'))
80
81   # ----------------------------------------------------------------------
----------------------------------------------
82   ## Create element sets of element surfaces
83   SurfaceAnnulusInferior_ES, SurfaceSetInferiorAnnulus_ES, SurfaceAnnu-
lusSuperior_ES, SurfaceSetSuperiorAnnulus_ES, \
84   SurfaceNucleusInferior_ES, SurfaceSetInferiorNucleus_ES, Surface-
NucleusSuperior_ES, SurfaceSetSuperiorNucleus_ES= [], [], [], [], [], [],
[], []
85   Surface_Vertebra_Inferior_ES, SurfaceSet_Vertebra_Inferior_ES = [], []
86
87   for k in range(len(PartModelNameAnnulus)):
88       SurfaceAnnulusInferior_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.surfaces[DiscNames[k] + '_ANNULUS_INFERIOR'].elements
89       SurfaceSetInferiorAnnulus_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.Set(elements=SurfaceAnnulusInferior_ES, name='ES_'+
DiscNames[k] + '_ANNULUS_INFERIOR_SET')
90
91       SurfaceAnnulusSuperior_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.surfaces[DiscNames[k] + '_ANNULUS_SUPERIOR'].elements
92       SurfaceSetSuperiorAnnulus_ES = mdb.models[PartModelNameAnnu-
lus[k]].rootAssembly.Set(elements=SurfaceAnnulusSuperior_ES, name='ES_' +
DiscNames[k] + '_ANNULUS_SUPERIOR_SET')
93
94       SurfaceNucleusInferior_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.surfaces[DiscNames[k] + '_NUCLEUS_INFERIOR'].ele-
ments
95       SurfaceSetInferiorNucleus_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.Set(elements=SurfaceNucleusInferior_ES, name='ES_'+
DiscNames[k] + '_NUCLEUS_INFERIOR_SET')
96
```

```
97        SurfaceNucleusSuperior_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.surfaces[DiscNames[k] + '_NUCLEUS_SUPERIOR'].ele-
ments
98        SurfaceSetSuperiorNucleus_ES = mdb.models[PartModelNameNu-
cleus[k]].rootAssembly.Set(elements=SurfaceNucleusSuperior_ES, name='ES_' +
DiscNames[k] + '_NUCLEUS_SUPERIOR_SET')
99
100   for k in range(len(PartModelNameVertebra)-1):
101       Surface_Vertebra_Inferior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.surfaces['ES_' + PartModelNameVertebra[k] + '_INFERI-
OR'].elements
102       SurfaceSet_Vertebra_Inferior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.Set(elements=Surface_Vertebra_Inferior_ES, name='ES_'+
PartModelNameVertebra[k] + '_INFERIOR_SET')
103
104   for k in range(len(PartModelNameVertebra)):
105       Surface_Vertebra_Superior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.surfaces['ES_' + PartModelNameVertebra[k] + '_SUPERI-
OR'].elements
106       SurfaceSet_Vertebra_Superior_ES = mdb.models[PartModelNameVerte-
bra[k]].rootAssembly.Set(elements=Surface_Vertebra_Superior_ES, name='ES_'+
PartModelNameVertebra[k] + '_SUPERIOR_SET')
107
108   # ------------------------------------------------------------------
---------------------------------------------
109   ## MATERIAL PROPERTIES
110   ## Create empty lists
111   NameMatPropNucleus_old, NameMatPropAnnulus_old, NameMatPropVerte-
bra_old = [], [], []
112   NameMatPropNucleus_new, NameMatPropAnnulus_new, NameMatPropVerte-
bra_new = [], [], []
113
114   for h in range(len(DiscNames)):
115   ## NUCLEUS PULPOSUS
116       # Change Material properties for nucleus
117       NameMatPropNucleus_old.append(DiscNames[h] + '_NUCLEUS_COPY_MATE-
RIAL0')
118       NameMatPropNucleus_new.append(DiscNames[h] + '_NUCLEUS_MATERIAL')
119
120       ChangeKeyMatPropNucleus = mdb.models[PartModelNameNucleus[h]].ma-
terials.changeKey(
121           fromName=NameMatPropNucleus_old[h], toName=NameMatPropNu-
cleus_new[h])
122
123       # Elastic material properties + Density
124       mdb.models[PartModelNameNucleus[h]].materials[NameMatPropNu-
cleus_new[h]].elastic.setValues(
125           table=((1.0, 0.45),), moduli=LONG_TERM)
126       mdb.models[PartModelNameNucleus[h]].materials[NameMatPropNu-
cleus_new[h]].Density(table=((
127
1.0e-09,),))
128
129       # Edit section
130       EditSectionNucleus = mdb.models[PartModelNameNucleus[h]].sec-
tions['Section-1-ES_VOLUME0_MAT0'].setValues(
131           material=PartModelNameNucleus[h] + '_MATERIAL', thick-
ness=None)
132
133   ## ANNULUS FIBROSUS
134       # Change Material properties for annulus
```

```
135      NameMatPropAnnulus_old.append(DiscNames[h] + '_ANNULUS_COPY_MATE-
RIAL0')
136      NameMatPropAnnulus_new.append(DiscNames[h] + '_ANNULUS_MATERIAL')
137
138      ChangeKeyMatPropAnnulus = mdb.models[PartModelNameAnnulus[h]].ma-
terials.changeKey(
139      fromName=NameMatPropAnnulus_old[h], toName=NameMatPropAnnu-
lus_new[h])
140
141      # Elastic material properties + Density
142      mdb.models[PartModelNameAnnulus[h]].materials[NameMatPropAnnu-
lus_new[h]].elastic.setValues(
143           table=((4.2, 0.45),), moduli=LONG_TERM)
144      mdb.models[PartModelNameAnnulus[h]].materials[NameMatPropAnnu-
lus_new[h]].Density(table=((
145
1.2e-09,),))
146
147      # Edit section
148      EditSectionAnnulus = mdb.models[PartModelNameAnnulus[h]].sec-
tions['Section-1-ES_VOLUME0_MAT0'].setValues(
149           material=PartModelNameAnnulus[h] + '_MATERIAL', thick-
ness=None)
150
151  ## VERTEBRA
152  for g in range(len(PartModelNameVertebra)):
153      # Change Material properties for vertebrae
154      NameMatPropVertebra_old.append(PartModelNameVertebra[g] +
'_COPY_MATERIAL0')
155      NameMatPropVertebra_new.append(PartModelNameVertebra[g] + '_MATE-
RIAL')
156
157      ChangeKeyMatVertebra = mdb.models[PartModelNameVertebra[g]].mate-
rials.changeKey(
158      fromName=NameMatPropVertebra_old[g], toName=NameMatPropVerte-
bra_new[g])
159
160      # Elastic material properties + Density
161      mdb.models[PartModelNameVertebra[g]].materials[NameMatPropVerte-
bra_new[g]].elastic.setValues(
162           table=((12000, 0.3),), moduli=LONG_TERM)
163      mdb.models[PartModelNameVertebra[g]].materials[NameMatPropVerte-
bra_new[g]].Density(table=((
164
1.6e-09,),))
165
166      # Edit section
167      EditSectionVertebrae = mdb.models[PartModelNameVertebra[g]].sec-
tions['Section-1-ES_VOLUME0_MAT0'].setValues(
168           material=PartModelNameVertebra[g] + '_MATERIAL', thick-
ness=None)
169
170  for q in range(len(PartModelName)):
171      p1 = mdb.models[PartModelName[q]].parts['PART-1']
172      # mdb.models[PartModelName[q]].parts.changeKey(fromName='PART-1',
toName=PartModelName[q])
173      mdb.models[PartModelName[q]].rootAssembly.regenerate()
174      mdb.models[PartModelName[q]].rootAssembly.features.changeKey(from-
Name='PART-1-1',
175           toName=PartModelName[q])
176
```

```
177      # ------------------------------------------------------------------
---------------------------------------------------
178      ## COPY MODELS INTO ONE MODEL: SPINE MODEL
179      mdb.models[modelname].rootAssembly.Instance(name=PartModelName[q]
+ '-1', model=mdb.models[PartModelName[q]])
180
181  # # ------------------------------------------------------------------
---------------------------------------------------
182  ## BOUNDARY CONDITIONS
183  # Create boundary condition encastre
184  InitialStepName = 'Initial'
185  regionEncastre = a.allInstances['S1-1'].sets['NS_S1_LATERAL']
186  EncastreBC = mdb.models[modelname].EncastreBC(name='S1_ENCASTRE', cre-
ateStepName=InitialStepName,
187          region=regionEncastre, localCsys=None)
188
189  # # ------------------------------------------------------------------
---------------------------------------------------
190  # ## CREATE LISTS
191  # Create empty lists for surface unions
192  MergeSetInferior_NS, MergeSetSuperior_NS, MergeSetNameInferior_NS,
MergeSetNameSuperior_NS = [], [], [], []
193  MergeSurfaceInferior_ES, MergeSurfaceSuperior_ES, MergeSurfaceNameIn-
ferior_ES, MergeSurfaceNameSuperior_ES = [], [], [], []
194  MergeSetInferior_ES, MergeSetSuperior_ES, MergeSetNameInferior_ES,
MergeSetNameSuperior_ES = [], [], [], []
195
196  # Create empty lists of tie regions
197  region1Inferior_NS, region2Inferior_NS, region1Superior_NS, region2Su-
perior_NS = [], [], [], []
198  region1Inferior_ES, region2Inferior_ES, region1Superior_ES, region2Su-
perior_ES = [], [], [], []
199
200  VertebraeNames = ['C7', 'T1', 'T2', 'T3', 'T4', 'T5', 'T6',
201               'T7', 'T8', 'T9', 'T10', 'T11', 'T12',
202               'L1', 'L2', 'L3', 'L4', 'L5', 'S1']
203
204  # ------------------------------------------------------------------
---------------------------------------------------
205  ## MERGE SURFACES AND SETS
206  for m in range(len(PartModelNameAnnulus)): ## == len(DiscNames) == 18
207      ## MERGE SURFACES OF NODE SETS
208      MergeSetNameInferior_NS = 'Disc_'+ DiscNames[m] + '_infe-
rior_NodeSet'
209      MergeSetInferior_NS.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameInferior_NS, sets=(
210              mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].sets[('NS_' + DiscNames[m] + '_ANNULUS_INFERI-
OR')],
211              mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_NUCLEUS_INFERI-
OR')],)))
212
213      MergeSetNameSuperior_NS = 'Disc_' + DiscNames[m] + '_supe-
rior_NodeSet'
214      MergeSetSuperior_NS.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameSuperior_NS, sets=(
215          mdb.models[modelname].rootAssembly.allInstances[PartModelName-
Annulus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_ANNULUS_SUPERIOR')],
216          mdb.models[modelname].rootAssembly.allInstances[PartModelNa-
meNucleus[m]+ '-1'].sets[('NS_' + DiscNames[m] + '_NUCLEUS_SUPERIOR')],)))
```

```
217
218       ## MERGE SURFACES OF ELEMENT SURFACES
219       MergeSurfaceNameInferior_ES = 'Disc_' + DiscNames[m] + '_infe-
rior_ElementSurface'
220       MergeSurfaceInferior_ES.append(mdb.models[modelname].rootAssem-
bly.SurfaceByBoolean(name=MergeSurfaceNameInferior_ES, surfaces=(
221                 mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].surfaces[(DiscNames[m] + '_ANNULUS_INFERIOR')],
222                 mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m] + '-1'].surfaces[(DiscNames[m] + '_NUCLEUS_INFERIOR')],)))
223
224       MergeSurfaceNameSuperior_ES = 'Disc_' + DiscNames[m] + '_supe-
rior_ElementSurface'
225       MergeSurfaceSuperior_ES.append(mdb.models[modelname].rootAssem-
bly.SurfaceByBoolean(name=MergeSurfaceNameSuperior_ES, surfaces=(
226                 mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].surfaces[(DiscNames[m] + '_ANNULUS_SUPERIOR')],
227                 mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m] + '-1'].surfaces[(DiscNames[m] + '_NUCLEUS_SUPERIOR')],)))
228
229       ## MERGE SURFACES OF ELEMENT SETS
230       MergeSetNameInferior_ES = 'Disc_' + DiscNames[m] + '_inferior_Ele-
mentSet'
231       MergeSetInferior_ES.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameInferior_ES, sets=(
232                 mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_ANNULUS_INFE-
RIOR_SET')],
233                 mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_NUCLEUS_INFE-
RIOR_SET')])))
234
235       MergeSetNameSuperior_ES = 'Disc_' + DiscNames[m] + '_superior_Ele-
mentSet'
236       MergeSetSuperior_ES.append(mdb.models[modelname].rootAssembly.Set-
ByBoolean(name=MergeSetNameSuperior_ES, sets=(
237                 mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameAnnulus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_ANNULUS_SUPE-
RIOR_SET')],
238                 mdb.models[modelname].rootAssembly.allInstances[PartMod-
elNameNucleus[m] + '-1'].sets[('ES_' + DiscNames[m] + '_NUCLEUS_SUPE-
RIOR_SET')])))
239
240   # ------------------------------------------------------------------------
-----------------------------------------------
241   ## TIE MERGED ELEMENT SURFACES OF DISCS WITH VERTEBRAE
242       region1Superior_ES.append(mdb.models[modelname].rootAssembly.sur-
faces[MergeSurfaceNameSuperior_ES])
243       region2Superior_ES.append(mdb.models[modelname].rootAssembly.al-
lInstances[PartModelNameVertebra[m]+ '-1'].surfaces[('ES_' + Vertebrae-
Names[m] + '_INFERIOR')])
244       TieConstraintSuperior_ES = mdb.models[modelname].Tie(name='Tie_' +
DiscNames[m] + '_Superior_ES', master=region2Superior_ES[m],
245                                                 slave=region1Supe-
rior_ES[m], positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
246                                                   thickness=ON, con-
straintEnforcement=NODE_TO_SURFACE)
247
248       region1Inferior_ES.append(mdb.models[modelname].rootAssembly.sur-
faces[MergeSurfaceNameInferior_ES])
```

```
249       region2Inferior_ES.append(mdb.models[modelname].rootAssembly.al-
lInstances[PartModelNameVertebra[m]+ '-1'].surfaces[('ES_' + Vertebrae-
Names[m] + '_SUPERIOR')])
250       TieConstraintInferior_ES = mdb.models[modelname].Tie(name='Tie_' +
DiscNames[m-1] + '_Inferior_ES', master=region2Inferior_ES[m],
251                                                          slave=region1Infe-
rior_ES[m-1], positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
252                                                           thickness=ON, con-
straintEnforcement=NODE_TO_SURFACE)
253
254       # ------------------------------------------------------------------
-----------------------------------------------------
255       ## TIE NODE ELEMENT SURFACES OF VERTABRAE TO A WHOLE PART
256
257   for i in range(len(PartModelNameVertebra)-1):
258       WholeVertebraeNodeSet = mdb.models[PartModelNameVerte-
bra[i]].rootAssembly.SetByBoolean(name='NS_' + PartModelNameVertebra[i] +
'_WHOLE_VERTEBRA', sets=(
259           mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+
PartModelNameVertebra[i] + '_SUPERIOR'],
260           mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+
PartModelNameVertebra[i] +'_LATERAL'],
261           mdb.models[PartModelNameVertebra[i]].rootAssembly.sets['NS_'+
PartModelNameVertebra[i] +'_INFERIOR'],))
262
263   # Delete the L5S1_Inferior constraint that incorrectly constraints
with the C7T1_Superior disc
264   del mdb.models[modelname].constraints['Tie_' + DiscNames[17] + '_Infe-
rior_ES']
265
266   # # Manually recreate L5S1_Inferior tie constraint nodeset
267   # TieConstraintL5S1_Inferior_NS = mdb.models[mod-
elname].Tie(name='Tie_' + DiscNames[17] + '_Inferior_NS', master =
(mdb.models[modelname].rootAssembly.allInstances[PartModelNameVertebra[18]
+ '-1'].sets[('NS_' + VertebraeNames[18] + '_SUPERIOR')]),
268   #                                                              slave=re-
gion1Inferior_NS[17], positionToleranceMethod=COMPUTED, adjust=OFF, tieRo-
tations=ON,
269   #                                                           thickness=ON, con-
straintEnforcement=SOLVER_DEFAULT)
270
271   # Manually recreate L5S1_Inferior tie constraint elementset
272   TieConstraintL5S1_Inferior_ES = mdb.models[modelname].Tie(name='Tie_'
+ DiscNames[17] + '_Inferior_ES', master=mdb.models[modelname].rootAssem-
bly.allInstances['S1-1'].surfaces['ES_S1_SUPERIOR'],
273                                           slave=region1Inferior_ES[17], po-
sitionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
274                                           thickness=ON, constraintEnforce-
ment=NODE_TO_SURFACE)
275
276   # ------------------------------------------------------------------------
-----------------------------------------------
277   ## CREATE COUPLING CONSTRAINT
278   NodeNameCenter = 'CenterOfC7'
279   SelectedNode_C7 = mdb.models[modelname].rootAssembly.instances['C7-
1.C7'].nodes.getSequenceFromMask(mask=('[#0:34 #40 ]',), )
280   region1 = mdb.models[modelname].rootAssembly.Set(nodes=Selected-
Node_C7, name=NodeNameCenter)
281   region2 = mdb.models[modelname].rootAssembly.allInstances['C7-1'].sur-
faces['ES_C7_SUPERIOR']
282
```

```
283  Coupling = mdb.models[modelname].Coupling(name='Coupling_C7_Moments',
controlPoint=region1,
284                                       surface=region2, influenceRa-
dius=WHOLE_SURFACE, couplingType=KINEMATIC,
285                                       localCsys=None, u1=ON, u2=ON, u3=ON,
ur1=ON, ur2=ON, ur3=ON)
286
287  # ------------------------------------------------------------------------
-----------------------------------------------
288  ## CREATE STEP
289  InitialStepName = 'Initial'
290  StepName1 = 'Load'
291  Step1 = mdb.models[modelname].StaticStep(name=StepName1, previous=Ini-
tialStepName,
292                                           stabilizationMagnitude=0.0002,
293                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
294                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
295                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
296
297  # ------------------------------------------------------------------------
-----------------------------------------------
298  ## COPY MODELS
299  Motion = ['Flexion', 'Extension', 'LeftLateralBending', 'RightLateral-
Bending', 'LeftAxialRotation', 'RightAxialRotation', 'Traction']
300
301  # ChangeKey
302  mdb.models.changeKey(fromName=modelname, toName=modelname + Motion[0])
303  session.viewports['Viewport: 1'].setValues(displayedObject=None)
304
305  for i in range(len(Motion)-1):
306      # Copy Models
307      mdb.Model(name=modelname + Motion[1+i], objectToCopy=mdb.mod-
els[modelname + Motion[0]])
308      session.viewports['Viewport: 1'].setValues(displayedObject=None)
309
310  # ------------------------------------------------------------------------
-----------------------------------------------
311  ## APPLY MOMENT
312  regionMomentCM1, MotionCM1, regionMomentCM2, MotionCM2, re-
gionMomentCM3, MotionCM3 = [], [], [], [], [], []
313  MotionNameCM1 = Motion[0:1]
314  MotionNameCM2 = Motion[2:3]
315  MotionNameCM3 = Motion[4:5]
316  CM = (10000.0, -10000.0)
317
318  FlexionMoment = mdb.models[modelname + Motion[0]].Moment(name=Mo-
tion[0], createStepName=StepName1,
319                                           region=mdb.models[modelname +
Motion[0]].rootAssembly.sets[NodeNameCenter], cm1=10000, distribution-
Type=UNIFORM, field='',
320                                           localCsys=None)
321
322  ExtensionMoment = mdb.models[modelname + Motion[1]].Moment(name=Mo-
tion[1], createStepName=StepName1,
323                                           region=mdb.models[modelname +
Motion[1]].rootAssembly.sets[NodeNameCenter], cm1=-10000, distribution-
Type=UNIFORM, field='',
324                                           localCsys=None)
```

```
325
326  LeftLat = mdb.models[modelname + Motion[2]].Moment(name=Motion[2],
createStepName=StepName1,
327                                                region=mdb.models[modelname +
Motion[2]].rootAssembly.sets[NodeNameCenter], cm2=10000, distribution-
Type=UNIFORM, field='',
328                                                localCsys=None)
329
330  RightLat = mdb.models[modelname + Motion[3]].Moment(name=Motion[3],
createStepName=StepName1,
331                                                region=mdb.models[modelname +
Motion[3]].rootAssembly.sets[NodeNameCenter], cm2=-10000, distribution-
Type=UNIFORM, field='',
332                                                localCsys=None)
333
334
335  LeftAR = mdb.models[modelname + Motion[4]].Moment(name=Motion[4], cre-
ateStepName=StepName1,
336                                                region=mdb.models[modelname +
Motion[4]].rootAssembly.sets[NodeNameCenter], cm3=10000, distribution-
Type=UNIFORM, field='',
337                                                localCsys=None)
338
339  RightAR = mdb.models[modelname + Motion[5]].Moment(name=Motion[5],
createStepName=StepName1,
340                                                region=mdb.models[modelname +
Motion[5]].rootAssembly.sets[NodeNameCenter], cm3=-10000, distribution-
Type=UNIFORM, field='',
341                                                localCsys=None)
342
343  regionTraction = mdb.models[modelname + Motion[6]].rootAssembly.allIn-
stances['C7-1'].sets['NS_C7_WHOLE_VERTEBRA']
344  mdb.models[modelname + Motion[6]].ConcentratedForce(name=Motion[6],
345                                                      createStepName=Step-
Name1, region=regionTraction, cf3=147.15,
346                                                      distribution-
Type=UNIFORM, field='', localCsys=None)
347
348  # ------------------------------------------------------------------------
-----------------------------------------------
349  ## CREATE AND RUN JOB
350  jobnames, newmodelnames, Jobs = [], [], []
351
352  for i in range(len(Motion)):
353      time_start = time.clock()
354      jobnames.append("Job_" + Motion[i])
355      newmodelnames.append(modelname + Motion[i])
356
357      Jobs = mdb.Job(name=jobnames[i], model=mdb.models[newmod-
elnames[i]], atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
358                     explicitPrecision=SINGLE, getMemoryFromAnaly-
sis=True, historyPrint=OFF,
359                     memory=90, memoryUnits=PERCENTAGE, mod-
elPrint=OFF,
360                     multiprocessingMode=DEFAULT, nodalOutputPreci-
sion=SINGLE,
361                     numCpus=1, queue=None, scratch='', type=ANALYSIS,
userSubroutine='',
362                     waitHours=0, waitMinutes=0)
363
364      # Submit the first job - this returns immediately
```

```
365       Jobs.submit(consistencyChecking=OFF)
366
367       # Now wait for the first job - this will block until the job com-
pletes
368       Jobs.waitForCompletion()
369
370       # Add delay in execution of the program
371       t = 1  # In seconds
372       time.sleep(t)
373
374   # # ------------------------------------------------------------------
----------------------------------------------
375   # COMPUTATION TIME
376       time_elapsed = (time.clock() - time_start)
377       print('Time elapsed =', time_elapsed, 'seconds')
```

Steps&RunJob.py

Creates all steps and submits and runs the job. CreateAssembly.py should be executed first.

```
1     from part import *
2     from material import *
3     from section import *
4     from assembly import *
5     from step import *
6     from interaction import *
7     from load import *
8     from mesh import *
9     from optimization import *
10    from job import *
11    from sketch import *
12    from visualization import *
13    from connectorBehavior import *
14    from os import listdir
15    from os.path import isfile, join
16    import xml.etree.ElementTree as ET
17    import regionToolset  # to add reference points
18    import interaction  # to include constraints
19    import subprocess  # to restart program
20    import odbAccess
21    import numpy as np
22    from itertools import repeat
23    from scipy.linalg import norm
24    from scipy.io import loadmat  # import matlab data into python
25    from os.path import dirname, join as pjoin
26    import scipy.io as sio
27    import sys
28    from sys import argv
29    # Read odb field output
30    from odbAccess import *
31    from abaqusConstants import *
32    from odbMaterial import *
33    from odbSection import *
34    from abaqus import *
35    from abaqusConstants import *
36    from viewerModules import *
37    from driverUtils import executeOnCaeStartup
38    import time
39    from abaqus import *
40    from abaqusConstants import *
41    from caeModules import *
42    import step
```

```
43   import time
44   import matplotlib
45   import csv
46   import os
47   import math
48
49   # ------------------------------------------------------------------------
-------------------------------------------------
50
51   # Define the names
52   modelname = "Spine model"
53   time_start = time.clock()
54
55   SetsScrewsLeft, SetScrewsRight = [], []
56
57   # Define list with with partnames
58   PartNames = ["C7", "T1", "T2", "T3", "T4", "T5", "T6", "T7", "T8",
"T9", "T10", "T11", "T12",
59                "L1", "L2", "L3", "L4", "L5", "S1"]
60
61   for q in range(len(PartNames)-1):
62       SetsScrewsLeft = ["Screw['" + PartNames[q] + "_L']"]
63       print(SetsScrewsLeft)
64
65   # ------------------------------------------------------------------------
-------------------------------------------------
66   # Create straight line through S1-C7 as aiming position
67
68   CreateDP_S1 = mdb.models['S1'].parts['PART-1'].DatumPointByCoordi-
nate(coords=(10.32, -102.65, -852.11))
69   CreateDP_C7 = mdb.models['C7'].parts['PART-1'].DatumPointByCoordi-
nate(coords=(12.27, -127.87, -468.18))
70
71   RegenerateModel = mdb.models[modelname].rootAssembly.regenerate()
72
73   d1 = mdb.models[modelname].rootAssembly.instances['S1-1.S1'].datums
74   d2 = mdb.models[modelname].rootAssembly.instances['C7-1.C7'].datums
75
76   # ------------------------------------------------------------------------
-------------------------------------------------
77   # SELECT THREE NODES PER NODESET
78
79   # Create lists to write data to
80   PartLevelLeft, PartLevelRight, pt1L, pt2L, pt3L, Threepoints = [], [],
[], [], [], []
81
82   # Create the list to save all coordinates
83   aLabels, axcoords, aycoords, azcoords = [],[],[],[]
84   pL = []
85   LabelsOfAllNodes = []
86   ThreePointSet = []
87
88   for i in range(len(PartNames)-1):  # S1 does not contain screws
89       # Select nodesets
90       # PartLevelLeft = mdb.models[modelname].rootAssembly.in-
stances[PartNames[i] + "-1." + PartNames[i]].nodesets["Screw['" + Part-
Names[i] + "_L']"]
91       PartLevelRight.append(mdb.models[modelname].rootAssem-
bly.sets["Screw['" + PartNames[i] + "_R']"])
92
93   # Create step 1
```

```
94   InitialStepName = 'Initial'
95   StepName = ['Step-1', 'Step-2', 'Step-3', 'Step-4', 'Step-5', 'Step-
6', 'Step-7', 'Step-8', 'Step-9', 'Step-10',
96               'Step-11', 'Step-12', 'Step-13', 'Step-14', 'Step-15',
'Step-16', 'Step-17', 'Step-18',  'Step-19', 'Step-20',
97               'Step-21', 'Step-22','Step-23', 'Step-24', 'Step-25',
'Step-26','Step-27', 'Step-28', 'Step-29','Step-30',
98               'Step-31', 'Step-32']
99
100
101
102  # ------------------------------------------------------------------------
-----------------------------------------------
103  ## CREATE ROD
104  # ------------------------------------------------------------------------
-----------------------------------------------
105  partname = 'Rod'
106
107  # ------------------------------------------------------------------------
-----------------------------------------------
108  ## DRAW SKETCH
109  RodSketch = mdb.models[modelname].ConstrainedSketch(name='__pro-
file__',
110                                                      sheetSize=200.0)
111  g, v, d, c = RodSketch.geometry, RodSketch.vertices, RodSketch.dimen-
sions, RodSketch.constraints
112  RodSketch.setPrimaryObject(option=STANDALONE)
113
114  RodSketch.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(3.0,
0.0)) # Radius = 3.0 mm
115  CreateRodPart = mdb.models[modelname].Part(name=partname, dimensional-
ity=THREE_D,
116                                       type=DEFORMABLE_BODY)
117
118  # ------------------------------------------------------------------------
-----------------------------------------------
119  ## EXTRUDE SKETCH
120  depth_rod1 = 380.0 # Rod length in [mm]
121  ExtrudeSketch = mdb.models[modelname].parts[partname].BaseSolidEx-
trude(sketch=RodSketch, depth=depth_rod1)
122  RodSketch.unsetPrimaryObject()
123
124  # ------------------------------------------------------------------------
-----------------------------------------------
125  ## CREATE MATERIAL
126  MaterialNameCoCr = 'CoCr'
127  mdb.models[modelname].Material(name=MaterialNameCoCr)
128  mdb.models[modelname].materials[MaterialNameCoCr].Elastic(ta-
ble=((53150.0, 0.0),))
129
130  # ------------------------------------------------------------------------
-----------------------------------------------
131  ## CREATE SECTION
132  SectionNameCoCr = 'Section_CoCr_Rod'
133  mdb.models[modelname].HomogeneousSolidSection(name=SectionNameCoCr,
134                                                material=Material-
NameCoCr, thickness=None)
135
136  # ------------------------------------------------------------------------
-----------------------------------------------
137  ## ASSIGN SECTION
```

```
138  regionNameRod = 'Rod_Set'
139  cells = mdb.models[modelname].parts[partname].cells.getSequenceFrom-
Mask(mask=('[#1 ]',), )
140  region = mdb.models[modelname].parts[partname].Set(cells=cells,
name=regionNameRod)
141  mdb.models[modelname].parts[partname].SectionAssignment(region=region,
sectionName=SectionNameCoCr, offset=0.0,
142                      offsetType=MIDDLE_SURFACE, offsetField='',
143                      thicknessAssignment=FROM_SECTION)
144
145  # ------------------------------------------------------------------------
-----------------------------------------------
146  ## CREATE SURFACES
147
148  # Inferior surface
149  RodFaces_Inferior = mdb.models[modelname].parts[partname].faces.get-
SequenceFromMask(mask=('[#4 ]',), )
150  CreateRod_Inferior_Surface = mdb.models[modelname].parts[part-
name].Surface(side1Faces=RodFaces_Inferior, name='Inferior surface')
151
152  # Superior surface
153  RodFaces_Superior = mdb.models[modelname].parts[partname].faces.get-
SequenceFromMask(mask=('[#2 ]',), )
154  CreateRod_Superior_Surface = mdb.models[modelname].parts[part-
name].Surface(side1Faces=RodFaces_Superior, name='Superior surface')
155
156  # Side- or lateral surface
157  RodFaces_Lateral = mdb.models[modelname].parts[partname].faces.get-
SequenceFromMask(mask=('[#1 ]',), )
158  CreateRod_Lateral_Surface = mdb.models[modelname].parts[partname].Sur-
face(side1Faces=RodFaces_Lateral, name='Lateral surface')
159
160  # ------------------------------------------------------------------------
-----------------------------------------------
161  ## CREATE SETS
162  # Inferior set
163  CreateRod_Inferior_Set = mdb.models[modelname].parts[part-
name].Set(faces=RodFaces_Inferior, name='Inferior set')
164
165  # Superior set
166  CreateRod_Superior_Set = mdb.models[modelname].parts[part-
name].Set(faces=RodFaces_Superior, name='Superior set')
167
168  # Side- or lateral set
169  CreateRod_Lateral_Set = mdb.models[modelname].parts[part-
name].Set(faces=RodFaces_Lateral, name='Lateral set')
170
171  # ------------------------------------------------------------------------
-----------------------------------------------
172  ## CREATE ROD INSTANCE IN SPINE MODEL
173  instancerodname = 'Rod-1'
174  mdb.models[modelname].rootAssembly.DatumCsysByDefault(CARTESIAN)
175  CreateRodInstance = mdb.models[modelname].rootAssembly.In-
stance(name=instancerodname, part=mdb.models[modelname].parts[partname],
dependent=ON)
176
177  # ------------------------------------------------------------------------
-----------------------------------------------
178  ## BOUNDARY CONDITIONS
179  regionrod1 = mdb.models[modelname].rootAssembly.instances[instancerod-
name].sets['Inferior set']
```

```
180  Create_BC_Encastre = mdb.models[modelname].EncastreBC(name='Encastre',
createStepName=InitialStepName,
181                                      region=regionrod1, localCsys=None)
182
183  regionrod2 = mdb.models[modelname].rootAssembly.instances[instancerod-
name].sets['Superior set']
184  Create_BC_Roller = mdb.models[modelname].ZasymmBC(name='Single
roller', createStepName=InitialStepName,
185                                    region=regionrod2, localCsys=None)
186
187  # ------------------------------------------------------------------------
-------------------------------------------------
188  ## MESH PART
189  SeedPart = mdb.models[modelname].parts[partname].seedPart(size=1.0,
deviationFactor=0.1, minSizeFactor=0.1)
190  GenerateMesh = mdb.models[modelname].parts[partname].generateMesh()
191
192  # ------------------------------------------------------------------------
-------------------------------------------------
193  ## CREATE AXIS THROUGH CYLINDER
194  CreateAxis = mdb.models[modelname].parts[partname].DatumAxisByCyl-
Face(face=mdb.models[modelname].parts[partname].faces[0])
195
196  # ------------------------------------------------------------------------
-------------------------------------------------
197  ## CREATE DATUM POINTS
198  # Create datum points using parameters
199  edgesRod = mdb.models[modelname].rootAssembly.instances[instancerod-
name].edges
200  CreateDPsLeft_Superior = mdb.models[modelname].rootAssembly.Da-
tumPointByEdgeParam(edge=edgesRod[0], parameter=0.75)
201  CreateDPsRight_Superior = mdb.models[modelname].rootAssembly.Da-
tumPointByEdgeParam(edge=edgesRod[0], parameter=0.25)
202  CreateDPsLeft_Inferior = mdb.models[modelname].rootAssembly.Da-
tumPointByEdgeParam(edge=edgesRod[1], parameter=0.25)
203  CreateDPsRight_Inferior = mdb.models[modelname].rootAssembly.Da-
tumPointByEdgeParam(edge=edgesRod[1], parameter=0.75)
204
205  print(CreateDPsRight_Superior.name)
206
207  DPsLeft_Sup_ID = CreateDPsLeft_Superior.id
208  DPsRight_Sup_ID = CreateDPsRight_Superior.id
209  DPsLeft_Inf_ID = CreateDPsLeft_Inferior.id
210  DPsRight_Inf_ID = CreateDPsRight_Inferior.id
211
212  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Creat-
eDPsLeft_Superior.name, toName='DP_Left_Superior')
213  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Creat-
eDPsRight_Superior.name, toName='DP_Right_Superior')
214  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Creat-
eDPsLeft_Inferior.name, toName='DP_Left_Inferior')
215  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Creat-
eDPsRight_Inferior.name, toName='DP_Right_Inferior')
216
217
218  # Create midpoint
219  datumpts = mdb.models[modelname].rootAssembly.datums
220  print('datumpts', datumpts.keys())
221  CreateMidPointsSuperiorSurface = mdb.models[modelname].rootAssem-
bly.DatumPointByMidPoint(point1=datumpts[DPsLeft_Sup_ID], point2=da-
tumpts[DPsRight_Sup_ID])
```

```
CreateMidPointsInferiorSurface = mdb.models[modelname].rootAssembly.DatumPointByMidPoint(point1=datumpts[DPsLeft_Inf_ID], point2=datumpts[DPsRight_Inf_ID])

mdb.models[modelname].rootAssembly.features.changeKey(fromName=CreateMidPointsSuperiorSurface.name, toName='DP_Midpoint_Superior')
mdb.models[modelname].rootAssembly.features.changeKey(fromName=CreateMidPointsInferiorSurface.name, toName='DP_Midpoint_Inferior')

# Create lines
AxisLeft = mdb.models[modelname].rootAssembly.DatumAxisByTwoPoint(point1=datumpts[DPsLeft_Sup_ID], point2=datumpts[DPsLeft_Inf_ID])
AxisRight = mdb.models[modelname].rootAssembly.DatumAxisByTwoPoint(point1=datumpts[DPsRight_Sup_ID], point2=datumpts[DPsRight_Inf_ID])

AxisLeft_rename = mdb.models[modelname].rootAssembly.features.changeKey(fromName=AxisLeft.name, toName='Axis_Left')
AxisRight_rename = mdb.models[modelname].rootAssembly.features.changeKey(fromName=AxisRight.name, toName='Axis_Right')

# ------------------------------------------------------------------------------------------------
## ADD STEPS AND DISPLACEMENT
# ------------------------------------------------------------------------------------------------
## STEP 1
# Create step
Step1 = mdb.models[modelname].StaticStep(name=StepName[0], previous=InitialStepName,
                                         stabilizationMagnitude=0.0002,
                                         stabilizationMethod=DISSIPATED_ENERGY_FRACTION,
                                         continueDampingFactors=False, adaptiveDampingRatio=None,
                                         initialInc=0.1, minInc=1e-35, maxInc=0.1)

# Fix vertebra
regionStep1 = mdb.models[modelname].rootAssembly.allInstances['C7-1'].sets['NS_C7_WHOLE_VERTEBRA']
FixStep1 = mdb.models[modelname].DisplacementBC(name='Fix step 1 - C7', createStepName=StepName[0],
                                                region=regionStep1, u1=SET, u2=SET, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET,
                                                amplitude=UNSET, fixed=ON, distributionType=UNIFORM, fieldName='',
                                                localCsys=None)

UR2_T2_step1 = math.radians(-10)

# Add displacement
regionDP_Step1 = mdb.models[modelname].rootAssembly.sets["RF['T2_R']"]
DPStep1 = mdb.models[modelname].DisplacementBC(name='Rotation step 1 - T2', createStepName=StepName[0],
                                               region=regionDP_Step1, u1=UNSET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T2_step1,
                                               ur3=UNSET, amplitude=UNSET, fixed=OFF, distributionType=UNIFORM,
                                               fieldName='', localCsys=None)


```

```
262  # ------------------------------------------------------------------------
-------------------------------------------------
263  ## STEP 2
264  # Create step
265  Step2 = mdb.models[modelname].StaticStep(name=StepName[1], previ-
ous=StepName[0],
266                                           stabilizationMagnitude=0.0002,
267                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
268                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
269                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
270
271  UR2_T3_step2 = math.radians(-55)
272
273  # Add displacement
274  regionDP_Step2 = mdb.models[modelname].rootAssembly.sets["RF['T3_R']"]
275  DPStep2 = mdb.models[modelname].DisplacementBC(name='Rotation step 1 -
T3', createStepName=StepName[1],
276                                                 region=regionDP_Step2, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T3_step2,
277                                                 ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
278                                                 fieldName='', localCsys=None)
279
280  # ------------------------------------------------------------------------
-------------------------------------------------
281  ## STEP 3
282  # Create step
283  Step3 = mdb.models[modelname].StaticStep(name=StepName[2], previ-
ous=StepName[1],
284                                           stabilizationMagnitude=0.0002,
285                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
286                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
287                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
288
289
290  UR3_T3_step3 = math.radians(-5)
291
292  # Add displacement
293  regionDP_Step3 = mdb.models[modelname].rootAssembly.sets["RF['T3_R']"]
294  DPStep3 = mdb.models[modelname].DisplacementBC(name='Rotation step 3 -
T3', createStepName=StepName[2],
295                                                 region=regionDP_Step3, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
296                                                 ur3=UR3_T3_step3, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
297                                                 fieldName='', localCsys=None)
298
299  # ------------------------------------------------------------------------
-------------------------------------------------
300  ## STEP 4
301  # Create step
302  Step4 = mdb.models[modelname].StaticStep(name=StepName[3], previ-
ous=StepName[2],
303                                           stabilizationMagnitude=0.0002,
```

```
304                                     stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
305                                     continueDampingFactors=False,
adaptiveDampingRatio=None,
306                                     initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
307
308
309  UR2_T4_step4 = math.radians(-15)
310
311  # Add displacement
312  regionDP_Step4 = mdb.models[modelname].rootAssembly.sets["RF['T4_L']"]
313  DPStep4 = mdb.models[modelname].DisplacementBC(name='Rotation step 4 -
T4', createStepName=StepName[3],
314                                         region=regionDP_Step4, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
315                                         ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
316                                         fieldName='', localCsys=None)
317
318  # ----------------------------------------------------------------------
-----------------------------------------------
319  ## STEP 5
320  # Create step
321  Step5 = mdb.models[modelname].StaticStep(name=StepName[4], previ-
ous=StepName[3],
322                                     stabilizationMagnitude=0.0002,
323                                     stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
324                                     continueDampingFactors=False,
adaptiveDampingRatio=None,
325                                     initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
326
327
328  UR2_T4_step5 = math.radians(-35)
329
330  # Add displacement
331  regionDP_Step5 = mdb.models[modelname].rootAssembly.sets["RF['T4_R']"]
332  DPStep5 = mdb.models[modelname].DisplacementBC(name='Rotation step 5 -
T3', createStepName=StepName[4],
333                                         region=regionDP_Step5, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
334                                         ur3=UR2_T4_step5, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
335                                         fieldName='', localCsys=None)
336
337  # ----------------------------------------------------------------------
-----------------------------------------------
338  ## STEP 6
339  # Create step
340  Step6 = mdb.models[modelname].StaticStep(name=StepName[5], previ-
ous=StepName[4],
341                                     stabilizationMagnitude=0.0002,
342                                     stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
343                                     continueDampingFactors=False,
adaptiveDampingRatio=None,
344                                     initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
345
```

```
346  UR2_T6_step6 = math.radians(15)
347
348  # Add displacement
349  regionDP_Step6 = mdb.models[modelname].rootAssembly.sets["RF['T5_R']"]
350  DPStep6 = mdb.models[modelname].DisplacementBC(name='Rotation step 6 -
T3', createStepName=StepName[5],
351                                                 region=regionDP_Step6, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
352                                                 ur3=UR2_T6_step6, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
353                                                 fieldName='', localCsys=None)
354
355  # ------------------------------------------------------------------------
-----------------------------------------------
356  ## STEP 7
357  # Create step
358  Step7 = mdb.models[modelname].StaticStep(name=StepName[6], previ-
ous=StepName[5],
359                                           stabilizationMagnitude=0.0002,
360                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
361                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
362                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
363
364  UR2_T6_step7 = math.radians(-55) # 95
365
366  # Add displacement
367  regionDP_Step7 = mdb.models[modelname].rootAssembly.sets["RF['T5_R']"]
368  DPStep7 = mdb.models[modelname].DisplacementBC(name='Rotation step 7 -
T3', createStepName=StepName[6],
369                                                 region=regionDP_Step7, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T6_step7,
370                                                 ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
371                                                 fieldName='', localCsys=None)
372
373  # ------------------------------------------------------------------------
-----------------------------------------------
374  ## STEP 8
375  # Create step
376  Step8 = mdb.models[modelname].StaticStep(name=StepName[7], previ-
ous=StepName[6],
377                                           stabilizationMagnitude=0.0002,
378                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
379                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
380                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
381
382  UR2_T6_step8 = math.radians(20)
383
384  # Add displacement
385  regionDP_Step8 = mdb.models[modelname].rootAssembly.sets["RF['T6_R']"]
386  DPStep8 = mdb.models[modelname].DisplacementBC(name='Rotation step 8 -
T3', createStepName=StepName[7],
387                                                 region=regionDP_Step8, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
```

```
388                                             ur3=UR2_T6_step8, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
389                                             fieldName='', localCsys=None)
390
391  # ------------------------------------------------------------------------
-----------------------------------------------
392  ## STEP 9
393  # Create step
394  Step9 = mdb.models[modelname].StaticStep(name=StepName[8], previ-
ous=StepName[7],
395                                          stabilizationMagnitude=0.0002,
396                                          stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
397                                          continueDampingFactors=False,
adaptiveDampingRatio=None,
398                                          initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
399
400  UR2_T6_step9 = math.radians(-30)
401
402  # Add displacement
403  regionDP_Step9 = mdb.models[modelname].rootAssembly.sets["RF['T6_R']"]
404  DPStep9 = mdb.models[modelname].DisplacementBC(name='Rotation step 9 -
T3', createStepName=StepName[8],
405                                             region=regionDP_Step9, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
406                                             ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
407                                             fieldName='', localCsys=None)
408
409  # ------------------------------------------------------------------------
-----------------------------------------------
410  ## STEP 10
411  # Create step
412  Step10 = mdb.models[modelname].StaticStep(name=StepName[9], previ-
ous=StepName[8],
413                                          stabilizationMagnitude=0.0002,
414                                          stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
415                                          continueDampingFactors=False,
adaptiveDampingRatio=None,
416                                          initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
417
418  # ------------------------------------------------------------------------
-----------------------------------------------
419  ## STEP 11
420  # Create step
421  Step11 = mdb.models[modelname].StaticStep(name=StepName[10], previ-
ous=StepName[9],
422                                          stabilizationMagnitude=0.0002,
423                                          stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
424                                          continueDampingFactors=False,
adaptiveDampingRatio=None,
425                                          initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
426
427  UR2_T6_step11 = math.radians(-55)
428
429  # Add displacement
```

```
430  regionDP_Step11 = mdb.models[modelname].rootAssem-
bly.sets["RF['T7_R']"]
431  DPStep11 = mdb.models[modelname].DisplacementBC(name='Rotation step 11
- T3', createStepName=StepName[10],
432                                                  region=regionDP_Step11, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T6_step11,
433                                                  ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
434                                                  fieldName='', localCsys=None)
435
436  # ------------------------------------------------------------------------
-----------------------------------------------
437  ## STEP 12
438  # Create step
439  Step12 = mdb.models[modelname].StaticStep(name=StepName[11], previ-
ous=StepName[10],
440                                            stabilizationMagnitude=0.0002,
441                                            stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
442                                            continueDampingFactors=False,
adaptiveDampingRatio=None,
443                                            initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
444
445
446  UR2_T6_step12 = math.radians(15)
447
448  # Add displacement
449  regionDP_Step12 = mdb.models[modelname].rootAssem-
bly.sets["RF['T7_R']"]
450  DPStep12 = mdb.models[modelname].DisplacementBC(name='Rotation step 12
- T3', createStepName=StepName[11],
451                                                  region=regionDP_Step12, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
452                                                  ur3=UR2_T6_step12, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
453                                                  fieldName='', localCsys=None)
454
455  # ------------------------------------------------------------------------
-----------------------------------------------
456  ## STEP 13
457  # Create step
458  Step13 = mdb.models[modelname].StaticStep(name=StepName[12], previ-
ous=StepName[11],
459                                            stabilizationMagnitude=0.0002,
460                                            stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
461                                            continueDampingFactors=False,
adaptiveDampingRatio=None,
462                                            initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
463
464
465  UR2_T6_step13 = math.radians(30)
466
467  # Add displacement
468  regionDP_Step13 = mdb.models[modelname].rootAssem-
bly.sets["RF['T8_R']"]
469  DPStep13 = mdb.models[modelname].DisplacementBC(name='Rotation step 13
- T3', createStepName=StepName[12],
```

```
470                                             region=regionDP_Step13, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
471                                             ur3=UR2_T6_step13, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
472                                             fieldName='', localCsys=None)
473
474  # ------------------------------------------------------------------------
-------------------------------------------------
475  ## STEP 14
476  # Create step
477  Step14 = mdb.models[modelname].StaticStep(name=StepName[13], previ-
ous=StepName[12],
478                                           stabilizationMagnitude=0.0002,
479                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
480                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
481                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
482
483  UR2_T6_step14 = math.radians(-20)
484
485  # Add displacement
486  regionDP_Step14 = mdb.models[modelname].rootAssem-
bly.sets["RF['T8_R']"]
487  DPStep14 = mdb.models[modelname].DisplacementBC(name='Rotation step 14
- T3', createStepName=StepName[13],
488                                             region=regionDP_Step14, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T6_step14,
489                                             ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
490                                             fieldName='', localCsys=None)
491
492  # ------------------------------------------------------------------------
-------------------------------------------------
493  ## STEP 15
494  # Create step
495  Step15 = mdb.models[modelname].StaticStep(name=StepName[14], previ-
ous=StepName[13],
496                                           stabilizationMagnitude=0.0002,
497                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
498                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
499                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
500
501  UR2_T6_step15 = math.radians(40)
502
503  # Add displacement
504  regionDP_Step15 = mdb.models[modelname].rootAssem-
bly.sets["RF['T9_R']"]
505  DPStep15 = mdb.models[modelname].DisplacementBC(name='Rotation step 15
- T3', createStepName=StepName[14],
506                                             region=regionDP_Step15, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T6_step15,
507                                             ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
508                                             fieldName='', localCsys=None)
509
```

```
510  # ------------------------------------------------------------------------
-------------------------------------------------
511  ## STEP 16
512  # Create step
513  Step16 = mdb.models[modelname].StaticStep(name=StepName[15], previ-
ous=StepName[14],
514                                        stabilizationMagnitude=0.0002,
515                                        stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
516                                        continueDampingFactors=False,
adaptiveDampingRatio=None,
517                                        initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
518
519  UR2_T6_step16 = math.radians(10)
520
521  # Add displacement
522  regionDP_Step16 = mdb.models[modelname].rootAssem-
bly.sets["RF['T9_R']"]
523  DPStep16 = mdb.models[modelname].DisplacementBC(name='Rotation step 16
- T3', createStepName=StepName[15],
524                                            region=regionDP_Step16, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
525                                            ur3=UR2_T6_step16, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
526                                            fieldName='', localCsys=None)
527
528  # ------------------------------------------------------------------------
-------------------------------------------------
529  ## STEP 17
530  # Create step
531  Step17 = mdb.models[modelname].StaticStep(name=StepName[16], previ-
ous=StepName[15],
532                                        stabilizationMagnitude=0.0002,
533                                        stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
534                                        continueDampingFactors=False,
adaptiveDampingRatio=None,
535                                        initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
536
537
538  UR2_T6_step17 = math.radians(55)
539
540  # Add displacement
541  regionDP_Step17 = mdb.models[modelname].rootAssem-
bly.sets["RF['T10_L']"]
542  DPStep17 = mdb.models[modelname].DisplacementBC(name='Rotation step 17
- T3', createStepName=StepName[16],
543                                            region=regionDP_Step17, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T6_step17,
544                                            ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
545                                            fieldName='', localCsys=None)
546
547
548
549  # ------------------------------------------------------------------------
-------------------------------------------------
550  ## STEP 18
551  # Create step
```

```
552  Step18 = mdb.models[modelname].StaticStep(name=StepName[17], previ-
ous=StepName[16],
553                                           stabilizationMagnitude=0.0002,
554                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
555                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
556                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
557
558  UR2_T6_step18 = math.radians(10)
559
560  # Add displacement
561  regionDP_Step18 = mdb.models[modelname].rootAssem-
bly.sets["RF['T10_R']"]
562  DPStep18 = mdb.models[modelname].DisplacementBC(name='Rotation step 18
- T3', createStepName=StepName[17],
563                                                 region=regionDP_Step18, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
564                                                 ur3=UR2_T6_step18, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
565                                                 fieldName='', localCsys=None)
566
567  # ------------------------------------------------------------------------
-------------------------------------------------
568  ## STEP 19
569  # Create step
570  Step19 = mdb.models[modelname].StaticStep(name=StepName[18], previ-
ous=StepName[17],
571                                           stabilizationMagnitude=0.0002,
572                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
573                                           continueDampingFactors=False,
adaptiveDampingRatio=None,
574                                           initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
575
576  UR2_T6_step19 = math.radians(55)
577
578  # Add displacement
579  regionDP_Step19 = mdb.models[modelname].rootAssem-
bly.sets["RF['T11_R']"]
580  DPStep19 = mdb.models[modelname].DisplacementBC(name='Rotation step 19
- T11', createStepName=StepName[18],
581                                                 region=regionDP_Step19, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T6_step19,
582                                                 ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
583                                                 fieldName='', localCsys=None)
584
585
586  # ------------------------------------------------------------------------
-------------------------------------------------
587  ## STEP 20
588  # Create step
589  Step20 = mdb.models[modelname].StaticStep(name=StepName[19], previ-
ous=StepName[18],
590                                           stabilizationMagnitude=0.0002,
591                                           stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
```

```
592                                     continueDampingFactors=False,
adaptiveDampingRatio=None,
593                                     initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
594
595  UR2_T6_step20 = math.radians(15)
596
597  # Add displacement
598  regionDP_Step20 = mdb.models[modelname].rootAssem-
bly.sets["RF['T11_R']"]
599  DPStep20 = mdb.models[modelname].DisplacementBC(name='Rotation step 20
- T11', createStepName=StepName[19],
600                                         region=regionDP_Step20, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
601                                         ur3=UR2_T6_step20, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
602                                         fieldName='', localCsys=None)
603
604  # ----------------------------------------------------------------------
-------------------------------------------------
605  ## STEP 21
606  # Create step
607  Step21 = mdb.models[modelname].StaticStep(name=StepName[20], previ-
ous=StepName[19],
608                                     stabilizationMagnitude=0.0002,
609                                     stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
610                                     continueDampingFactors=False,
adaptiveDampingRatio=None,
611                                     initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
612
613  UR2_T12_step21 = math.radians(75)
614
615  # Add displacement
616  regionDP_Step21 = mdb.models[modelname].rootAssem-
bly.sets["RF['T12_R']"]
617  DPStep21 = mdb.models[modelname].DisplacementBC(name='Rotation step 21
- T11', createStepName=StepName[20],
618                                         region=regionDP_Step21, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T12_step21,
619                                         ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
620                                         fieldName='', localCsys=None)
621
622  # ----------------------------------------------------------------------
-------------------------------------------------
623  ## STEP 22
624  # Create step
625  Step22 = mdb.models[modelname].StaticStep(name=StepName[21], previ-
ous=StepName[20],
626                                     stabilizationMagnitude=0.0002,
627                                     stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
628                                     continueDampingFactors=False,
adaptiveDampingRatio=None,
629                                     initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
630
631  UR2_T12_step22 = math.radians(5)
632
```

```
633  # Add displacement
634  regionDP_Step22 = mdb.models[modelname].rootAssem-
bly.sets["RF['T12_R']"]
635  DPStep22 = mdb.models[modelname].DisplacementBC(name='Rotation step 22
- T11', createStepName=StepName[21],
636                                                  region=regionDP_Step22, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
637                                                  ur3=UR2_T12_step22, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
638                                                  fieldName='', localCsys=None)
639
640  # ----------------------------------------------------------------------
----------------------------------------------
641  ## STEP 23
642  # Create step
643  Step23 = mdb.models[modelname].StaticStep(name=StepName[22], previ-
ous=StepName[21],
644                                            stabilizationMagnitude=0.0002,
645                                            stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
646                                            continueDampingFactors=False,
adaptiveDampingRatio=None,
647                                            initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
648
649  UR2_T12_step23 = math.radians(5)
650
651  # Add displacement
652  regionDP_Step23 = mdb.models[modelname].rootAssem-
bly.sets["RF['L1_R']"]
653  DPStep23 = mdb.models[modelname].DisplacementBC(name='Rotation step 23
- T11', createStepName=StepName[22],
654                                                  region=regionDP_Step23, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
655                                                  ur3=UR2_T12_step23, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
656                                                  fieldName='', localCsys=None)
657
658  # ----------------------------------------------------------------------
----------------------------------------------
659  ## STEP 24
660  # Create step
661  Step24 = mdb.models[modelname].StaticStep(name=StepName[23], previ-
ous=StepName[22],
662                                            stabilizationMagnitude=0.0002,
663                                            stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
664                                            continueDampingFactors=False,
adaptiveDampingRatio=None,
665                                            initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
666
667  UR2_T12_step24 = math.radians(35)
668
669  # Add displacement
670  regionDP_Step24 = mdb.models[modelname].rootAssem-
bly.sets["RF['L1_R']"]
671  DPStep24 = mdb.models[modelname].DisplacementBC(name='Rotation step 24
- T11', createStepName=StepName[23],
672                                                  region=regionDP_Step24, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T12_step24,
```

```
673                                             ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
674                                             fieldName='', localCsys=None)
675
676  # ------------------------------------------------------------------------
-----------------------------------------------
677  ## STEP 25
678  # Create step
679  Step25 = mdb.models[modelname].StaticStep(name=StepName[24], previ-
ous=StepName[23],
680                                         stabilizationMagnitude=0.0002,
681                                         stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
682                                         continueDampingFactors=False,
adaptiveDampingRatio=None,
683                                         initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
684
685  UR2_T12_step25 = math.radians(55)
686
687  # Add displacement
688  regionDP_Step25 = mdb.models[modelname].rootAssem-
bly.sets["RF['L2_R']"]
689  DPStep25 = mdb.models[modelname].DisplacementBC(name='Rotation step 25
- T11', createStepName=StepName[24],
690                                             region=regionDP_Step25, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T12_step25,
691                                             ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
692                                             fieldName='', localCsys=None)
693
694  # ------------------------------------------------------------------------
-----------------------------------------------
695  ## STEP 26
696  # Create step
697  Step26 = mdb.models[modelname].StaticStep(name=StepName[25], previ-
ous=StepName[24],
698                                         stabilizationMagnitude=0.0002,
699                                         stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
700                                         continueDampingFactors=False,
adaptiveDampingRatio=None,
701                                         initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
702
703  UR2_T12_step26 = math.radians(5)
704
705  # Add displacement
706  regionDP_Step26 = mdb.models[modelname].rootAssem-
bly.sets["RF['L2_R']"]
707  DPStep26 = mdb.models[modelname].DisplacementBC(name='Rotation step 26
- T11', createStepName=StepName[25],
708                                             region=regionDP_Step26, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
709                                             ur3=UR2_T12_step26, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
710                                             fieldName='', localCsys=None)
711
712
713
```

```
714  # ------------------------------------------------------------------------
-------------------------------------------------
715  ## STEP 27
716  # Create step
717  Step27 = mdb.models[modelname].StaticStep(name=StepName[26], previ-
ous=StepName[25],
718                                        stabilizationMagnitude=0.0002,
719                                        stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
720                                        continueDampingFactors=False,
adaptiveDampingRatio=None,
721                                        initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
722
723  UR2_T12_step27 = math.radians(5)
724
725  # Add displacement
726  regionDP_Step27 = mdb.models[modelname].rootAssem-
bly.sets["RF['L3_R']"]
727  DPStep27 = mdb.models[modelname].DisplacementBC(name='Rotation step 27
- T11', createStepName=StepName[26],
728                                             region=regionDP_Step27, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
729                                             ur3=UR2_T12_step27, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
730                                             fieldName='', localCsys=None)
731
732
733
734  # ------------------------------------------------------------------------
-------------------------------------------------
735  ## STEP 28
736  # Create step
737  Step28 = mdb.models[modelname].StaticStep(name=StepName[27], previ-
ous=StepName[26],
738                                        stabilizationMagnitude=0.0002,
739                                        stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
740                                        continueDampingFactors=False,
adaptiveDampingRatio=None,
741                                        initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
742
743  UR2_T12_step28 = math.radians(-15)
744
745
746  # Add displacement
747  regionDP_Step28 = mdb.models[modelname].rootAssem-
bly.sets["RF['L3_R']"]
748  DPStep28 = mdb.models[modelname].DisplacementBC(name='Rotation step 28
- T11', createStepName=StepName[27],
749                                             region=regionDP_Step28, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T12_step28,
750                                             ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
751                                             fieldName='', localCsys=None)
752  # ------------------------------------------------------------------------
-------------------------------------------------
753  ## STEP 29
754  # Create step
```

```
755  Step29 = mdb.models[modelname].StaticStep(name=StepName[28], previ-
ous=StepName[27],
756                                         stabilizationMagnitude=0.0002,
757                                         stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
758                                         continueDampingFactors=False,
adaptiveDampingRatio=None,
759                                         initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
760
761  UR2_T12_step29 = math.radians(-35)
762
763  # Add displacement
764  regionDP_Step29 = mdb.models[modelname].rootAssem-
bly.sets["RF['L4_R']"]
765  DPStep29 = mdb.models[modelname].DisplacementBC(name='Rotation step 29
- T11', createStepName=StepName[28],
766                                             region=regionDP_Step29, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T12_step29,
767                                             ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
768                                             fieldName='', localCsys=None)
769
770  # ------------------------------------------------------------------------
-------------------------------------------------
771  ## STEP 30
772  # Create step
773  Step30 = mdb.models[modelname].StaticStep(name=StepName[29], previ-
ous=StepName[28],
774                                         stabilizationMagnitude=0.0002,
775                                         stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
776                                         continueDampingFactors=False,
adaptiveDampingRatio=None,
777                                         initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
778
779  UR2_T12_step30 = math.radians(5)
780
781  # Add displacement
782  regionDP_Step30 = mdb.models[modelname].rootAssem-
bly.sets["RF['L4_R']"]
783  DPStep30 = mdb.models[modelname].DisplacementBC(name='Rotation step 30
- T11', createStepName=StepName[29],
784                                             region=regionDP_Step30, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET,
785                                             ur3=UR2_T12_step30, ampli-
tude=UNSET, fixed=OFF, distributionType=UNIFORM,
786                                             fieldName='', localCsys=None)
787
788
789
790  # ------------------------------------------------------------------------
-------------------------------------------------
791  ## STEP 31
792  # Create step
793  Step31 = mdb.models[modelname].StaticStep(name=StepName[30], previ-
ous=StepName[29],
794                                         stabilizationMagnitude=0.0002,
795                                         stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
```

```
796                                         continueDampingFactors=False,
adaptiveDampingRatio=None,
797                                         initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
798
799  UR2_T12_step31 = math.radians(-35)
800
801  # Add displacement
802  regionDP_Step31 = mdb.models[modelname].rootAssem-
bly.sets["RF['L5_R']"]
803  DPStep31 = mdb.models[modelname].DisplacementBC(name='Rotation step 31
- T11', createStepName=StepName[30],
804                                             region=regionDP_Step31, u1=UN-
SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UR2_T12_step31,
805                                             ur3=UNSET, amplitude=UNSET,
fixed=OFF, distributionType=UNIFORM,
806                                             fieldName='', localCsys=None)
807
808
809  # ----------------------------------------------------------------------
----------------------------------------------
810  ## STEP 32
811  # Create step
812  Step32 = mdb.models[modelname].StaticStep(name=StepName[31], previ-
ous=StepName[30],
813                                         stabilizationMagnitude=0.0002,
814                                         stabilizationMethod=DISSIPATED_EN-
ERGY_FRACTION,
815                                         continueDampingFactors=False,
adaptiveDampingRatio=None,
816                                         initialInc=0.1, minInc=1e-35, max-
Inc=0.1)
817
818  for i in range(len(PartNames)-1):
819      # Fix vertebra
820      regionStep32 = mdb.models[modelname].rootAssembly.allIn-
stances[PartNames[i] + '-1'].sets['NS_'+ PartNames[i] +'_WHOLE_VERTEBRA']
821      FixStep32 = mdb.models[modelname].DisplacementBC(name='Fix step 32
- ' + PartNames[i], createStepName=StepName[31],
822                                                 region=regionStep32,
u1=SET, u2=SET, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET,
823                                                 amplitude=UNSET, fixed=ON,
distributionType=UNIFORM, fieldName='',
824                                                 localCsys=None)
825
826
827  # ----------------------------------------------------------------------
----------------------------------------------
828  ## CREATE HISTORY OUTPUT REQUESTS
829  for i in range(len(PartNames)-1):
830      RegionSetName = "RF['" +  PartNames[i] +"_R']"
831      regionDef_RF = mdb.models[modelname].rootAssembly.sets[RegionSet-
Name]
832      mdb.models[modelname].HistoryOutputRequest(name='HO_RF_' + Part-
Names[i],
833                                                 createStepName=Step-
Name[0], variables=('RF1', 'RF2', 'RF3', 'RM1', 'RM2',
834
'RM3', 'RT', 'RM'), region=regionDef_RF,
835                                                 sectionPoints=DE-
FAULT,
```

```
836                                                     rebar=EXCLUDE)
837
838  # ------------------------------------------------------------------------
-----------------------------------------------
839  ## CREATE AND RUN JOB
840
841  # Create a Job from a model definition
842  jobname= "JobSpine model5"
843
844  j1 = mdb.Job(name=jobname, model=mdb.models[modelname], atTime=None,
contactPrint=OFF, description='', echoPrint=OFF,
845                       explicitPrecision=SINGLE, getMemoryFromAnaly-
sis=True, historyPrint=OFF,
846                       memory=90, memoryUnits=PERCENTAGE, mod-
elPrint=OFF,
847                       multiprocessingMode=DEFAULT, nodalOutputPreci-
sion=SINGLE,
848                       numCpus=1, queue=None, scratch='', type=ANALYSIS,
userSubroutine='',
849                       waitHours=0, waitMinutes=0)
850
851  # Submit the first job - this returns immediately
852  j1.submit(consistencyChecking=OFF)
853
854  # Now wait for the first job - this will block until the job completes
855  j1.waitForCompletion()
856
857  # Add delay in execution of the program
858  t = 1  # In seconds
859  time.sleep(t)
860
861
862  # COMPUTATION TIME
863  time_elapsed = (time.clock() - time_start)
864  print('Time elapsed = ', time_elapsed)
```

CreateRod.py
Creates the rod and applies the reaction moments as calculated in Steps&RunJob

```
1    import section
2    import regionToolset
3    import displayGroupMdbToolset as dgm
4    import part
5    import material
6    import assembly
7    import step
8    import interaction
9    import load
10   import mesh
11   import optimization
12   import job
13   import sketch
14   import visualization
15   import xyPlot
16   import displayGroupOdbToolset as dgo
17   import connectorBehavior
18   import part
19   import assembly
20   import material
21   import section
```

```
22   import interaction
23   import numpy as np
24   import time
25   from symbolicConstants import *
26   from abaqusConstants import *
27   import itertools
28   import sys
29   import xml.etree.ElementTree as ET
30   import regionToolset  # to add reference points
31   import interaction  # to include constraints
32   import subprocess  # to restart program
33   import time
34   from os import listdir
35   from os.path import isfile, join
36
37   # ------------------------------------------------------------------------
-----------------------------------------------
38   ## CHANGE MODELNAME
39   modelname = 'Rod model'
40   mdb.models.changeKey(fromName='Model-1', toName=modelname)
41
42   # ------------------------------------------------------------------------
-----------------------------------------------
43   ## CREATE SKETCH
44   CreateSketch = mdb.models[modelname].ConstrainedSketch(name='__pro-
file__',
45                                                       sheetSize=200.0)
46
47   g, v, d, c = CreateSketch.geometry, CreateSketch.vertices, Cre-
ateSketch.dimensions, CreateSketch.constraints
48   CreateSketch.setPrimaryObject(option=STANDALONE)
49   R_Rod = 5.5/2
50   CreateSketch.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(0.0,
R_Rod))
51
52
53   # Create 3D, deformable body
54   partName = 'Rod'
55   CreatePart = mdb.models[modelname].Part(name=partName, dimensional-
ity=THREE_D,
56                                       type=DEFORMABLE_BODY)
57
58   # ------------------------------------------------------------------------
-----------------------------------------------
59   ## EXTRUDE SKETCH
60   depth_rod1 = 380.0 # Rod length in [mm]
61   ExtrudeSketch = mdb.models[modelname].parts[partName].BaseSolidEx-
trude(sketch=CreateSketch, depth=depth_rod1)
62   CreateSketch.unsetPrimaryObject()
63
64   # ------------------------------------------------------------------------
-----------------------------------------------
65   ## CREATE MATERIAL
66   materialName = 'CoCr'
67   # CoCr
68   CreateMaterial = mdb.models[modelname].Material(name=materialName, de-
scription='CoCrMoC')
69   InsertValues = mdb.models[modelname].materials[materialName].Elas-
tic(table=((230000.0, 0.0),))
70
```

```
71   # ------------------------------------------------------------------------
---------------------------------------------
72   ## CREATE SECTION
73   sectionName = 'Rod'
74   mdb.models[modelname].HomogeneousSolidSection(name=sectionName, mate-
rial='CoCr',
75                                                 thickness=None)
76
77   # ------------------------------------------------------------------------
---------------------------------------------
78   ## ASSIGN SECTION
79   regionName = 'Rod'
80   cellsRod =  mdb.models[modelname].parts[partName].cells.getSequence-
FromMask(mask=('[#1 ]',), )
81   regionRod = mdb.models[modelname].parts[partName].Set(cells=cellsRod,
name=regionName)
82
83   mdb.models[modelname].parts[partName].SectionAssignment(region=re-
gionRod, sectionName=sectionName, offset=0.0,
84                       offsetType=MIDDLE_SURFACE, offsetField='',
85                       thicknessAssignment=FROM_SECTION)
86
87
88   # ------------------------------------------------------------------------
---------------------------------------------
89   ## CREATE SURFACES
90
91   # Inferior surface
92   RodFaces_Inferior = mdb.models[modelname].parts[partName].faces.get-
SequenceFromMask(mask=('[#4 ]',), )
93   CreateRod_Inferior_Surface = mdb.models[modelname].parts[part-
Name].Surface(side1Faces=RodFaces_Inferior, name='Inferior surface')
94
95   # Superior surface
96   RodFaces_Superior = mdb.models[modelname].parts[partName].faces.get-
SequenceFromMask(mask=('[#2 ]',), )
97   CreateRod_Superior_Surface = mdb.models[modelname].parts[part-
Name].Surface(side1Faces=RodFaces_Superior, name='Superior surface')
98
99   # Side- or lateral surface
100  RodFaces_Lateral = mdb.models[modelname].parts[partName].faces.get-
SequenceFromMask(mask=('[#1 ]',), )
101  CreateRod_Lateral_Surface = mdb.models[modelname].parts[partName].Sur-
face(side1Faces=RodFaces_Lateral, name='Lateral surface')
102
103
104  # ------------------------------------------------------------------------
---------------------------------------------
105  ## CREATE SETS
106  # Inferior set
107  CreateRod_Inferior_Set = mdb.models[modelname].parts[part-
Name].Set(faces=RodFaces_Inferior, name='Inferior set')
108
109  # Superior set
110  CreateRod_Superior_Set = mdb.models[modelname].parts[part-
Name].Set(faces=RodFaces_Superior, name='Superior set')
111
112  # Side- or lateral set
113  CreateRod_Lateral_Set = mdb.models[modelname].parts[part-
Name].Set(faces=RodFaces_Lateral, name='Lateral set')
114
```

```
115  # ------------------------------------------------------------------------
-------------------------------------------------
116  ## CREATE INSTANCE OF ASSEMBLY
117  instanceName = 'Rod-1'
118  CreateRodInstance = mdb.models[modelname].rootAssembly.In-
stance(name=instanceName, part=mdb.models[modelname].parts[partName], de-
pendent=ON)
119
120  # ------------------------------------------------------------------------
-------------------------------------------------
121  ## BOUNDARY CONDITIONS
122  region1 = mdb.models[modelname].rootAssembly.instances[in-
stanceName].sets['Inferior set']
123  Create_BC_Encastre = mdb.models[modelname].EncastreBC(name='Encastre',
createStepName='Initial',
124                                       region=region1, localCsys=None)
125
126  region2 = mdb.models[modelname].rootAssembly.instances[in-
stanceName].sets['Superior set']
127  Create_BC_Roller = mdb.models[modelname].ZasymmBC(name='Single
roller', createStepName='Initial',
128                                     region=region2, localCsys=None)
129
130  # ------------------------------------------------------------------------
-------------------------------------------------
131  ## MESH PART
132  SeedPart = mdb.models[modelname].parts[partName].seedPart(size=1.0,
deviationFactor=0.1, minSizeFactor=0.1)
133  GenerateMesh = mdb.models[modelname].parts[partName].generateMesh()
134
135
136  # ------------------------------------------------------------------------
-------------------------------------------------
137  ## CREATE AXIS THROUGH CYLINDER
138  CreateAxis = mdb.models[modelname].parts[partName].DatumAxisByCyl-
Face(face=mdb.models[modelname].parts[partName].faces[0])
139
140  # ------------------------------------------------------------------------
-------------------------------------------------
141  ## CREATE DATUM POINTS
142  # Create datum points using parameters
143  edgesRod = mdb.models[modelname].rootAssembly.instances[in-
stanceName].edges
144  CreateDPsLeft_Superior = mdb.models[modelname].rootAssembly.Da-
tumPointByEdgeParam(edge=edgesRod[0], parameter=0.75)
145  CreateDPsRight_Superior = mdb.models[modelname].rootAssembly.Da-
tumPointByEdgeParam(edge=edgesRod[0], parameter=0.25)
146  CreateDPsLeft_Inferior = mdb.models[modelname].rootAssembly.Da-
tumPointByEdgeParam(edge=edgesRod[1], parameter=0.25)
147  CreateDPsRight_Inferior = mdb.models[modelname].rootAssembly.Da-
tumPointByEdgeParam(edge=edgesRod[1], parameter=0.75)
148
149  print(CreateDPsRight_Superior.name)
150
151  DPsLeft_Sup_ID = CreateDPsLeft_Superior.id
152  DPsRight_Sup_ID = CreateDPsRight_Superior.id
153  DPsLeft_Inf_ID = CreateDPsLeft_Inferior.id
154  DPsRight_Inf_ID = CreateDPsRight_Inferior.id
155
156  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Creat-
eDPsLeft_Superior.name, toName='DP_Left_Superior')
```

```
157  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Creat-
eDPsRight_Superior.name, toName='DP_Right_Superior')
158  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Creat-
eDPsLeft_Inferior.name, toName='DP_Left_Inferior')
159  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Creat-
eDPsRight_Inferior.name, toName='DP_Right_Inferior')
160
161
162  # Create midpoint
163  datumpts = mdb.models[modelname].rootAssembly.datums
164  print('datumpts', datumpts.keys())
165  CreateMidPointsSuperiorSurface = mdb.models[modelname].rootAssem-
bly.DatumPointByMidPoint(point1=datumpts[DPsLeft_Sup_ID], point2=da-
tumpts[DPsRight_Sup_ID])
166  CreateMidPointsInferiorSurface = mdb.models[modelname].rootAssem-
bly.DatumPointByMidPoint(point1=datumpts[DPsLeft_Inf_ID], point2=da-
tumpts[DPsRight_Inf_ID])
167
168  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Cre-
ateMidPointsSuperiorSurface.name, toName='DP_Midpoint_Superior')
169  mdb.models[modelname].rootAssembly.features.changeKey(fromName=Cre-
ateMidPointsInferiorSurface.name, toName='DP_Midpoint_Inferior')
170
171  # Create lines
172  AxisLeft = mdb.models[modelname].rootAssembly.DatumAxisByTwo-
Point(point1=datumpts[DPsLeft_Sup_ID], point2=datumpts[DPsLeft_Inf_ID])
173  AxisRight = mdb.models[modelname].rootAssembly.DatumAxisByTwo-
Point(point1=datumpts[DPsRight_Sup_ID], point2=datumpts[DPsRight_Inf_ID])
174
175  AxisLeft_rename = mdb.models[modelname].rootAssembly.fea-
tures.changeKey(fromName=AxisLeft.name, toName='Axis_Left')
176  AxisRight_rename = mdb.models[modelname].rootAssembly.fea-
tures.changeKey(fromName=AxisRight.name, toName='Axis_Right')
177
178  # ------------------------------------------------------------------
---------------------------------------------
179  # LOAD REFERENCE POINTS
180
181  # Input map of the xml file
182  mypath = r'C:\1. Abaqus tijdelijk\220131 Automated model with steps
displacement\CurvesAndPrimitives.xml'
183
184  # Location of the xml file
185  tree = ET.parse(mypath)
186
187  # Get the root of the xml file
188  root = tree.getroot()
189
190  # Create empty lists
191  SetRefPoints, RFid = [], []
192  ScrewsCylinder, ScrewsCylinderES, SetRefPointsES = [], [], []
193  PartNames = ['T3', 'T4', 'T5', 'T6', 'T7', 'T8', 'T9', 'T10', 'T11',
'T12', 'L1', 'L2', 'L3', 'L4', 'L5', 'T2', 'T1', 'C7']
194  RFInitialName = []
195
196  # Loop to create points from xml file (37 rows, 3 columns)
197  for i in range(len(root)):
198      Point1 = root[i][1].text.split()  # Second column
199
200      # Create reference point
201      RFid = mdb.models[modelname].rootAssembly.ReferencePoint(
```

```
202          point=(float(Point1[0]), float(Point1[1]),
float(Point1[2]))).id
203      r1 = mdb.models[modelname].rootAssembly.referencePoints
204      refPoints1 = (r1[RFid],)
205      region = regionToolset.Region(referencePoints=refPoints1)
206
207  # Delete half of reference points
208  for i in range(2,37,2):
209      del mdb.models[modelname].rootAssembly.features['RP-' + str(i)]
210
211  for i in range(1,36,2):
212      RFInitialName.append('RP-' + str(i))
213      print(RFInitialName)
214
215  for i in range(len(PartNames)):
216      # Change name of reference points
217      mdb.models[modelname].rootAssembly.features.changeKey(fromName =
RFInitialName[i], toName='RF_' + PartNames[i])
218
219      # Move reference points along axis
220      coordDtP = [-2.8442,-99.1331,-811.33]
221
222  # ------------------------------------------------------------------
-----------------------------------------------
223  # TRANSLATE INSTANCE
224  TranslateRod = mdb.models[modelname].rootAssembly.translate(in-
stanceList=(instanceName, ), vector=(-2.8442,-99.1331,-811.33))
225
226  # ------------------------------------------------------------------
-----------------------------------------------
227  # TRANSLATE RF TO AXIS
228  MoveRFtoAxis, SetRF_Rod, RFID, refPoints= [], [], [], []
229
230  for i in range(len(PartNames)):
231      MoveRFtoAxis.append(mdb.models[modelname].rootAssembly.fea-
tures['RF_' + PartNames[i]].setValues(xValue=-5.8442, yValue=-99.1331))
232      RegenerateRod model = mdb.models[modelname].rootAssembly.regener-
ate()
233      RFID.append(mdb.models[modelname].rootAssembly.features['RF_' +
PartNames[i]].id)
234
235      # Create referencepoint sets
236      NameRFSets = 'RF_SET_' + PartNames[i]
237      refPoints.append((mdb.models[modelname].rootAssembly.reference-
Points[RFID[i]],))
238      RF_Sets = mdb.models[modelname].rootAssembly.Set(reference-
Points=refPoints[i], name=NameRFSets)
239
240  xcoord_RF = [-5.8442]
241  ycoord_RF = [-95.8227]
242  zcoord_RF = [-505.0691, -520.965161, -537.413514, -557.580445, -
581.4732, -605.928098, -629.68041, -654.151132,
243                   -676.9018, -701.1928, -720.753694, -743.977, -
768.2518, -792.5808, -811.33, -490.174002, -473.647653,
244                   -457.853373]
245
246  # Create nodesets
247  # NameNodeSetRod = 'C7_NS'
248  # nodes1 = mdb.models[modelname].rootAssembly.instances[in-
stanceName].nodes.getSequenceFromMask(mask=('[#0:617 #4000000 #60 #60040000
]', ), )
```

```
249  # mdb.models[modelname].rootAssembly.Set(nodes=nodes1,
name=NameNodeSetRod)
250
251  maskNS = ['[#0:535 #60040000 #0 #600400 ]',      # T3
252  '[#0:507 #60040000 #0 #600400 ]',                # T4
253  '[#0:477 #4000000 #60 #60040000 ]',              # T5
254  '[#0:442 #4000000 #60 #60040000 ]',              # T6
255  '[#0:400 #4000000 #60 #60040000 ]',              # T7
256  '[#0:358 #4000000 #60 #60040000 ]', # T7
257  '[#0:316 #4000000 #60 #60040000 ]',# T9
258  '[#0:274 #4000000 #60 #60040000 ]', # T10
259  '[#0:234 #60040000 #0 #600400 ]', # T11
260  '[#0:192 #60040000 #0 #600400 ]', # T12
261  '[#0:157 #60040000 #0 #600400 ]', # L1
262  '[#0:117 #600400 #0 #6004 ]', # L2
263  '[#0:75 #600400 #0 #6004 ]', # L3
264  '[#0:31 #60040000 #0 #600400 ]', # L4
265  '[#6004 #4000000 #60 ]', # L5
266  '[#0:589 #4000000 #60 #60040000 ]', # T2
267  '[#0:617 #4000000 #60 #60040000 ]', # T1
268  '[#0:561 #4000000 #60 #60040000 ]'] # C7
269
270  for i in range(len(PartNames)):
271      nodes1 = mdb.models[modelname].rootAssembly.instances[in-
stanceName].nodes.getSequenceFromMask(mask=(maskNS[i], ), )
272      mdb.models[modelname].rootAssembly.Set(nodes=nodes1, name='NS_' +
PartNames[i])
273
274
275
276  # ------------------------------------------------------------------------
-----------------------------------------------
277  ## CREATE COUPLING CONSTRAINTS
278
279  for i in range(len(PartNames)):
280      regionBC1 = mdb.models[modelname].rootAssembly.sets['RF_SET_' +
PartNames[i]]
281      regionBC2 = mdb.models[modelname].rootAssembly.sets['NS_' + Part-
Names[i]]
282
283      # Couple set with nodeset
284      mdb.models[modelname].Coupling(name='Constraint_' + PartNames[i],
controlPoint=regionBC1,
285                                     surface=regionBC2, influenceRa-
dius=WHOLE_SURFACE, couplingType=KINEMATIC,
286                                     localCsys=None, u1=ON, u2=ON,
u3=ON, ur1=ON, ur2=ON, ur3=ON)
287
288
289  # ------------------------------------------------------------------------
-----------------------------------------------
290  ## CREATE STEP 1
291  stepnameinitial = 'Initial'
292  stepname1 = 'Step_1'
293  StepRod1 = mdb.models[modelname].StaticStep(name=stepname1, previ-
ous=stepnameinitial,
294                                    initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
295
296  region1 = mdb.models[modelname].rootAssembly.sets['RF_SET_T3']
297  forcename1 = 'Load_T3'
```

```
298  mdb.models[modelname].ConcentratedForce(name=forcename1, createStep-
Name=stepname1,
299                                         region=region1, cf2=-
9008.123046875, distributionType=UNIFORM, field='',
300                                         localCsys=None)
301
302  # ------------------------------------------------------------------
-----------------------------------------------
303  ## CREATE STEP 2
304  stepname2 = 'Step_2'
305  StepRod2 = mdb.models[modelname].StaticStep(name=stepname2, previ-
ous=stepname1,
306                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
307
308  region2 = mdb.models[modelname].rootAssembly.sets['RF_SET_T5']
309  forcename2 = 'Load_T5'
310  mdb.models[modelname].ConcentratedForce(name=forcename2, createStep-
Name=stepname2,
311                                         region=region2, cf2=-
44000.890625, distributionType=UNIFORM, field='',
312                                         localCsys=None)
313
314
315  # ------------------------------------------------------------------
-----------------------------------------------
316  ## CREATE STEP 3
317  stepname3 = 'Step_3'
318  StepRod3 = mdb.models[modelname].StaticStep(name=stepname3, previ-
ous=stepname2,
319                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
320
321  region3 = mdb.models[modelname].rootAssembly.sets['RF_SET_T7']
322  forcename3 = 'Load_T7'
323  mdb.models[modelname].ConcentratedForce(name=forcename3, createStep-
Name=stepname3,
324                                         region=region3, cf2=-
8749.8212890625, distributionType=UNIFORM, field='',
325                                         localCsys=None)
326
327  # ------------------------------------------------------------------
-----------------------------------------------
328  ## CREATE STEP 4
329  stepname4 = 'Step_4'
330  StepRod4 = mdb.models[modelname].StaticStep(name=stepname4, previ-
ous=stepname3,
331                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
332
333  region4 = mdb.models[modelname].rootAssembly.sets['RF_SET_T8']
334  forcename4 = 'Load_T8'
335  mdb.models[modelname].ConcentratedForce(name=forcename4, createStep-
Name=stepname4,
336                                         region=region4,
cf2=10097.3095703125, distributionType=UNIFORM, field='',
337                                         localCsys=None)
338
339
340  # ------------------------------------------------------------------
-----------------------------------------------
```

```
341  ## CREATE STEP 5
342  stepname5 = 'Step_5'
343  StepRod5 = mdb.models[modelname].StaticStep(name=stepname5, previ-
ous=stepname4,
344                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
345
346  region5 = mdb.models[modelname].rootAssembly.sets['RF_SET_T9']
347  forcename5 = 'Load_T9'
348  mdb.models[modelname].ConcentratedForce(name=forcename5, createStep-
Name=stepname5,
349                                          region=region5,
cf2=7019.3505859375, distributionType=UNIFORM, field='',
350                                          localCsys=None)
351
352  # ------------------------------------------------------------------
----------------------------------------------
353  ## CREATE STEP 6
354  stepname6 = 'Step_6'
355  StepRod6 = mdb.models[modelname].StaticStep(name=stepname6, previ-
ous=stepname5,
356                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
357
358  region6 = mdb.models[modelname].rootAssembly.sets['RF_SET_T11']
359  forcename6 = 'Load_T11'
360  mdb.models[modelname].ConcentratedForce(name=forcename6, createStep-
Name=stepname6,
361                                          region=region6, cf2=-
17206.68359375, distributionType=UNIFORM, field='',
362                                          localCsys=None)
363
364
365  # ------------------------------------------------------------------
----------------------------------------------
366  ## CREATE STEP 7
367  stepname7 = 'Step_7'
368  StepRod7 = mdb.models[modelname].StaticStep(name=stepname7, previ-
ous=stepname6,
369                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
370
371  region7 = mdb.models[modelname].rootAssembly.sets['RF_SET_T12']
372  forcename7 = 'Load_T12'
373  mdb.models[modelname].ConcentratedForce(name=forcename7, createStep-
Name=stepname7,
374                                          region=region7,
cf2=21659.62109375, distributionType=UNIFORM, field='',
375                                          localCsys=None)
376
377
378  # ------------------------------------------------------------------
----------------------------------------------
379  ## CREATE STEP 8
380  stepname8 = 'Step_8'
381  StepRod8 = mdb.models[modelname].StaticStep(name=stepname8, previ-
ous=stepname7,
382                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
383
384  region8 = mdb.models[modelname].rootAssembly.sets['RF_SET_L1']
```

```
385  forcename8 = 'Load_L1'
386  mdb.models[modelname].ConcentratedForce(name=forcename8, createStep-
Name=stepname8,
387                                          region=region8, cf2=-
36971.51953125, distributionType=UNIFORM, field='',
388                                          localCsys=None)
389
390  # ------------------------------------------------------------------------
-----------------------------------------------
391  ## CREATE STEP 9
392  stepname9 = 'Step_9'
393  StepRod9 = mdb.models[modelname].StaticStep(name=stepname9, previ-
ous=stepname8,
394                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
395
396  region9 = mdb.models[modelname].rootAssembly.sets['RF_SET_L2']
397  forcename9 = 'Load_L2'
398  mdb.models[modelname].ConcentratedForce(name=forcename9, createStep-
Name=stepname9,
399                                          region=region9,
cf2=68211.578125, distributionType=UNIFORM, field='',
400                                          localCsys=None)
401
402  # ------------------------------------------------------------------------
-----------------------------------------------
403  ## CREATE STEP 10
404  stepname10 = 'Step_10'
405  StepRod10 = mdb.models[modelname].StaticStep(name=stepname10, previ-
ous=stepname9,
406                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
407
408  region10 = mdb.models[modelname].rootAssembly.sets['RF_SET_L3']
409  forcename10 = 'Load_L3'
410  mdb.models[modelname].ConcentratedForce(name=forcename10, createStep-
Name=stepname10,
411                                          region=region10, cf2=-
53638.359375, distributionType=UNIFORM, field='',
412                                          localCsys=None)
413
414
415  # ------------------------------------------------------------------------
-----------------------------------------------
416  ## CREATE STEP 11
417  stepname11 = 'Step_11'
418  StepRod11 = mdb.models[modelname].StaticStep(name=stepname11, previ-
ous=stepname10,
419                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
420
421  region11 = mdb.models[modelname].rootAssembly.sets['RF_SET_L4']
422  forcename11 = 'Load_L4'
423  mdb.models[modelname].ConcentratedForce(name=forcename11, createStep-
Name=stepname11,
424                                          region=region11, cf2=-
33872.84765625, distributionType=UNIFORM, field='',
425                                          localCsys=None)
426
427  # ------------------------------------------------------------------------
-----------------------------------------------
```

```
428  ## CREATE STEP 12
429  stepname12 = 'Step_12'
430  StepRod12 = mdb.models[modelname].StaticStep(name=stepname12, previ-
ous=stepname11,
431                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
432
433  region12 = mdb.models[modelname].rootAssembly.sets['RF_SET_L5']
434  forcename12 = 'Load_L5'
435  mdb.models[modelname].ConcentratedForce(name=forcename12, createStep-
Name=stepname12,
436                                          region=region12, cf2=-
23715.29296875, distributionType=UNIFORM, field='',
437                                          localCsys=None)
438
439  # ------------------------------------------------------------------
----------------------------------------------
440  ## CREATE STEP 13
441  stepname13 = 'Step_13'
442  StepRod13 = mdb.models[modelname].StaticStep(name=stepname13, previ-
ous=stepname12,
443                                      initialInc=0.01, minInc=1e-35, max-
Inc=0.01)
444
445  region13 = mdb.models[modelname].rootAssembly.sets['RF_SET_T2']
446  forcename13 = 'Load_T2'
447  mdb.models[modelname].ConcentratedForce(name=forcename13, createStep-
Name=stepname13,
448                                          region=region13,
cf2=3354.5380859375, distributionType=UNIFORM, field='',
449                                          localCsys=None)
450
451  # ------------------------------------------------------------------
----------------------------------------------
452  ## CREATE AND RUN JOB
453  jobname = 'Job_Rod model'
454  JobRod = mdb.Job(name=jobname, model=modelname, description='',
type=ANALYSIS,
455          atTime=None, waitMinutes=0, waitHours=0, queue=None,
memory=90,
456          memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
457          explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echo-
Print=OFF,
458          modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSub-
routine='',
459          scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT,
numCpus=1,
460          numGPUs=0)
461
462
463  # Submit the first job - this returns immediately
464  JobRod.submit(consistencyChecking=OFF)
465
466  # Now wait for the first job - this will block until the job completes
467  JobRod.waitForCompletion()
468
469  # Add delay in execution of the program
470  t = 1  # In seconds
471  time.sleep(t)
```