The Abaqus/CAE Plug-in for a Premium Threaded Connection 3D Parametric Finite Element Model

Kaidi Yan

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Mechanical Engineering

Robert L. West, Chair
Jan Helge Bøhn
John M. Kennedy

April 26th, 2017
Blacksburg, VA

Keywords: Modeling, Finite Element Analyze, Coding

The Abaqus/CAE Plug-in for a Premium Threaded Connection 3D Parametric Finite Element Model

Kaidi Yan

Abstract

Finite Element Analysis (FEA) is proposed to simulate the connection response of practical in-service conditions and test the performance of Oil Country Tubular Goods (OCTG) premium threaded connections. A plug-in is developed in Abaqus/CAE for creating the 360-degree full 3D parametric finite element model with helical threads as an effective design and analysis tool. All size, position and material data of the model are parameterized. The premium connection plug-in accepts input from the Graphical User Interface (GUI) for further modification. Each premium connection component is programed as a collection of single-purpose independent functions organized as an independent module in order to allow users to modify specific function behavior conveniently. A main program is designed as an Abaqus kernel plug-in to achieve all the functions by calling these independent functions, making the plug-in flexible. Each single script file is not too long to jeopardize readability. The GUI of the plug-in is designed with proper layout arrangement and illustrations to make the plug-in user-friendly and easy to use. The premium connection FE model is used in a virtual test to validate the model against the ISO 13679 test protocol and is used to develop the seal metrics for points on the ISO 13679 sealability envelope. The plug-in can be used to develop and evaluate the design envelope of the premium connection.

The Abaqus/CAE Plug-in for a Premium Threaded Connection 3D Parametric Finite Element Model

Kaidi Yan

General Audience Abstract

Oil Country Tubular Goods (OCTG) refers to a specific kind of steel tube used in the oil and gas industry--following the specifications set by the American Petroleum Institute (API). As the drilling of the modern oil well goes deeper, the extremely high temperatures and pressures require better quality oil tubes and connections. The drill pipe, connected by Premium Connections, are designed and tested carefully in order to avoid any possible environmental pollution or financial loss resulting from technical failures. Physical testing of each design takes time and costs a lot. Finite Element Analysis (FEA) is proposed to simulate the connection response of practical in-service conditions and test the performance of OCTG premium threaded connections. Full 360-degree 3D finite element models are the most complete representation of premium threaded connections. A plug-in is developed in Abaqus/CAE for creating the finite element model with helical threads as an effective design and analysis tool. The plug-in can be used to develop and evaluate the design envelope of the premium connection.

Acknowledgements

I would like to thank my advisor, Dr. Robert L. West for his time and guidance throughout this research effort. The accomplishments of this work would not be possible without his direction and assistance.

I would like to thank Dr. Jan Helge Bøhn and Dr. John Kennedy for their knowledge and support throughout this work.

I am very grateful for my parents Guangping Yan and Jianhua Dang for their unconditional love, support and encouragement throughout all of my life.

Thanks to all of my friends, Yichuan Wang, Shan Gao, Yi Yan, and Yi Li for their love and support.

KAIDI YAN

Virginia Polytechnic Institute and State University

May 2016

Table of Contents

List of figures

List of tables

Nomenclature

English Symbols

API	Application Program Interface
Box	Female component of the premium threaded connection
CAD	Computer aided drawing
DOF	Degree of freedom
E	Young's modulus
FE	Finite elements
FEA	Finite element analysis
GUI	Graphical User Interface
ID	Identification Data
ISO	International Organization for Standardization
Pin	Male component of the premium threaded connection
RSG	Really Simple GUI

Greek Symbols

ν	Poisson's ratio
ε_y	transverse strain
ε_x	axial strain

Other Symbols

2D	Two dimensions or two dimensional
3D	Three dimensions or three dimensional

Chapter 1. Introduction

1.1 Overview

Oil Country Tubular Goods (OCTG) [1] refers to a specific kind of steel tube used in the oil and gas industry--following the specifications set by the American Petroleum Institute (API) [2]. As the drilling of the modern oil well goes deeper, the extremely high temperatures and pressures require better quality oil tubes and connections. The drill pipe, connected by Premium Connections, are designed and tested carefully in order to avoid any possible environmental pollution or financial loss resulting from technical failures. Due to the diversity of locations and work environments, the well-design application using premium connections is complicated to satisfy different design criteria for the specific operational requirements of the wells in the oil field. OCTG manufacturers usually produce several series of connections with different sizes, weights and material grades, each of which has, similar connections [3]. If physical testing is conducted on every design application of premium connections, the cost and time to validate the design can be considerable. To deal with this issue, Finite Element Analysis (FEA) is proposed to simulate practical working conditions and test the performance of the premium connection.

The development of a finite element (FE) model for premium connections is a complex process, which often includes repetitive tasks. When creating a FE model, the geometry is developed using CAD software or based on design drawings. The premium connection assembly includes contact between different parts in the assembly model as well as loads, boundary conditions, material properties of real products, as well as environmental conditions that may be used in the FEA computation. This process requires a solid background and practical experience of using CAD software, comprehensive knowledge of how FEA works as well as problem specific knowledge and expertise to simulate the model's representative response.

Developing a finite element model is not unlike programming at a very high level. Each operation can be seen as a command to the finite element software describing the data used to depict the model and the operations to be performed on the model. For Abaqus, a large-scale nonlinear commercial FEA code, most all modeling and post processing operations are completed by sending the corresponding commands from inside Abaqus/CAE. This thesis seeks to exploit the Abaqus scripting interface to directly communicate with the Abaqus CAE kernel without using the Abaqus/CAE Graphical User Interface (GUI). A script is created by storing all the script interface commands into a file, which can be run directly to execute these instructions. The premium connection script developed for this research is written in Python and must have the following functionalities:

1. Automatically executes repetitive tasks
2. Conducts parameter analysis. Control the change of a single metric, for example, the material property, through the script, conduct the analysis and provide output.
3. Creates and modifies models
4. Creates Abaqus/CAE plug-in programs.

The objective of this research is to implement an Abaqus/CAE plug-in program, which can automatically create models based on input parameters and prepare the premium connection model for FEA computation with only a few operations. This plug-in program does not require users to have expert skills in FEA modeling techniques or extensive expertise in the premium connection application scope. Figure 1-1 provides an illustration of the collaboration of this research effort in OCTG premium threaded connection design, testing, and qualification process.

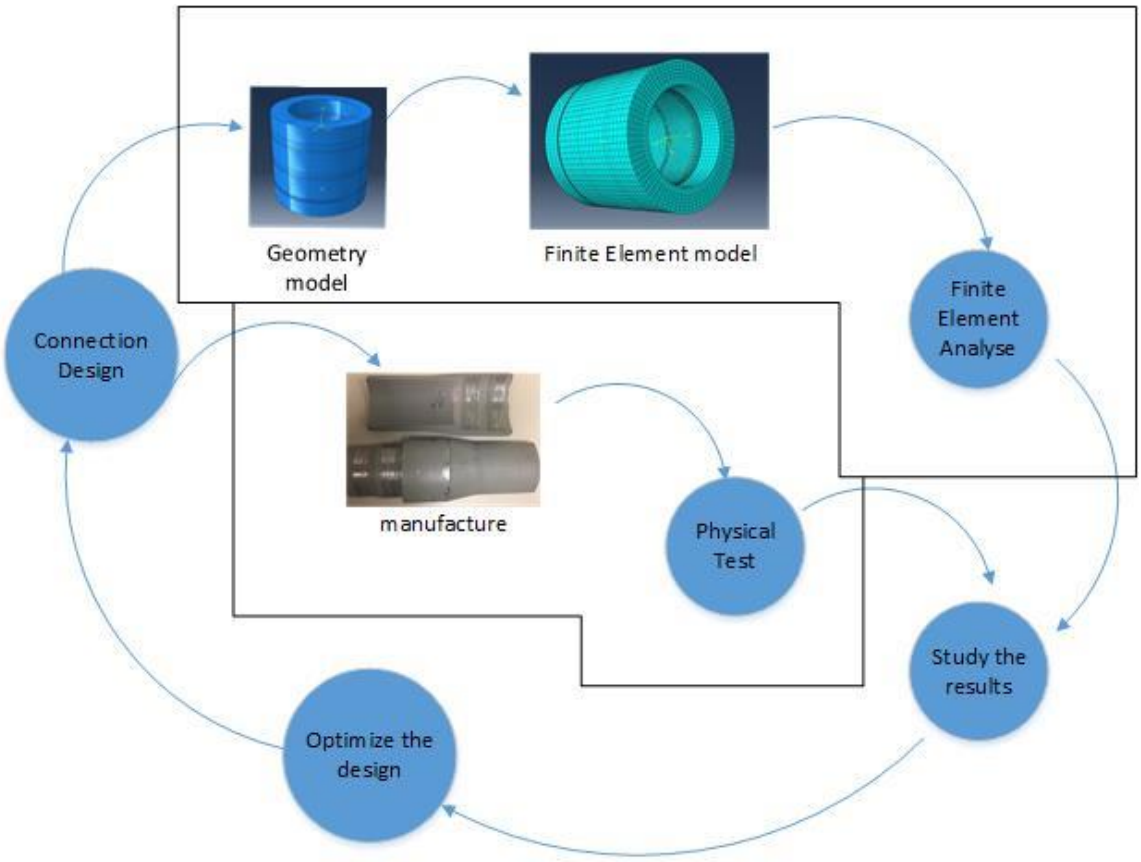

Figure 1-1: OCTG premium threaded connection design and physical test flow collaboration with the research effort.

1.2 Research Hypothesis

The research hypothesis of this thesis is that a 3-D parametric FE model for premium connections can be created by developing a plug-in to represent the connection behavior, which can be used to achieve the functions below:

1. The physical dimensions of the model can be adjusted by changing the input data through the interface of the plug-in,
2. The resulting finite element connection model can be reconciled with data to properly represent the make-up conditions, including the loads and the boundary conditions,

3. The resulting finite element connection model can be used in a virtual test to validate the model against the ISO 13679 test protocol [4],
4. The resulting finite element connection model enables better modeling of extreme events such as overpull, high temperature performance, casing wear impact on the connection, and sour environment performance[5].
5. The resulting finite element connection model can be used to develop the seal metrics for points on the ISO 13679 sealability envelope [4],
6. The plug-in can be used to develop and evaluate the design envelope of the connection design.

1.3 Research goals

The breakdown of the premium connection model is shown in the Figure 1-2. Only the joint is modeled in this research and will be divided into six parts, the pin, the box and four threads. The pin part and the box part are designed by the technical drawings with the threads removed. Helical threads, which preload the connection seal and define the torque-turn behavior, are created as separate components presented later in section 4.3. Overall geometric dimensions required to breakdown the connection features are extracted from manufacturer supplied technical drawings.

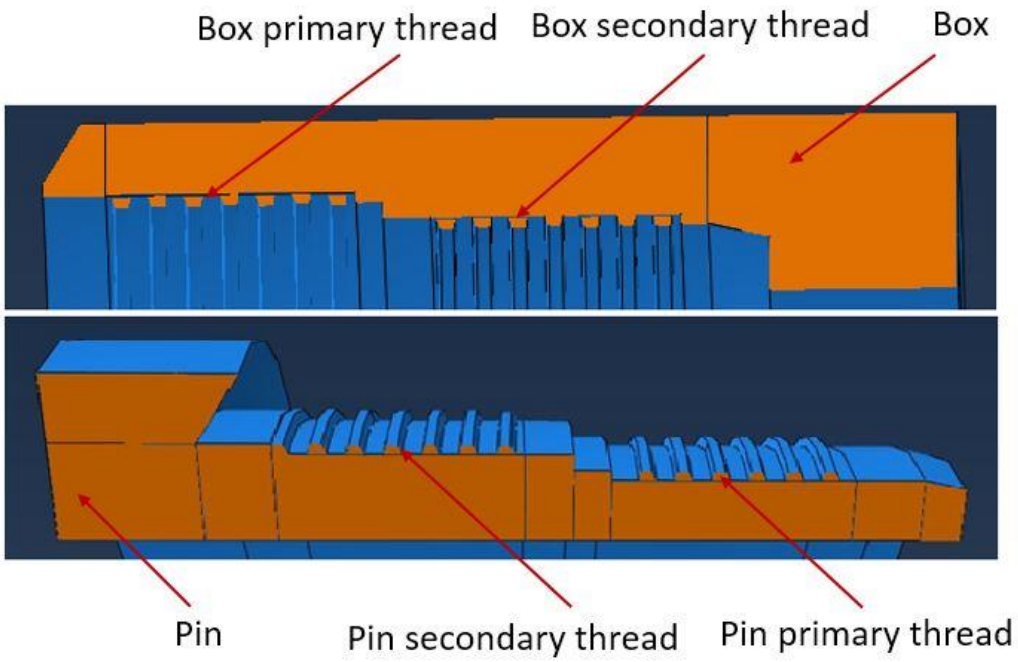

Figure 1-2: Component breakdown of the premium connection.

The selected premium connections model has a complex shape, which makes it difficult to build a plug-in program with parametric modeling. The plug-in should have the following functionality:

1. Automatically model a particular type of connection based on the connection size data provided by the engineering drawing and support updates in response to changes in sizes in the pin, box and threads parts.
2. Pass data to a geometric model based on changing the mechanical properties of materials.
3. Partition models for effective meshing, application of loads and boundary conditions. Divide a whole part into several components in order to mesh effectively and refine the mesh for model convergence studies.
4. Define "sets" for referring to features, components, loads, boundary conditions, data extraction or other functional requirements for the model.
5. Define the "surfaces" of part objects for interaction, constraints and loading.
6. Assemble the initial positions and orientations of the component parts, including the definition of reference points defined to support the assembly of the connection.

The goal is to develop a plug-in program that does as many "repetitive" tasks or programmed features to support conducting model experiments or tests upon different sizes of models. The objective is to build a finite element model that can be operational. The point that a finite element model is "operational" is required for design studies, correlation of the model with experimental data supporting verification and validation and the task of optimizing the model for design or operational criteria. The goal of this research is to develop a parametric finite element model that can be operated on to support further study using the finite element model. A plug-in-based approach to finite element modeling can support these types of tasks, using problem domain expertise and effectively exploiting the finite element model, providing time to apply toward more advanced efforts using the models. From the perspective of dollars and cents, developing a plug-in-based parametric finite element model leverages the time and effort put into finite element modeling toward validated models supporting advanced analysis, design and optimization tasks.

1.4 Research scope

The main goal of this research is to develop a plug-in to achieve the following functionalities for a fully parametric, operational finite element model of a premium connection:

1. The creation of parametric part geometries,
2. The model can be resized with input data that can be easily changed,
3. The model can be designed for effective meshing and mesh refinement,
4. The model can be used with associative references to features of components, loading, boundary conditions, meshing, constraints or the extraction of results to make the model "mesh independent",
5. Development of surfaces for referencing loading, contact interaction, constraints to make the model "mesh independent",
6. Include options for design configurations that can be used for further work,

7. Property of the parts, primarily associated with section and material properties,
8. Assemble the parts to the initial location in the premium connection assembly,
9. Allow for the use of tolerances on the parts and assemblies.

The model created by the plug-in is defined as a parametric finite element functional geometric model. The plug-in operations for geometry building are followed by creating analysis steps, settings the interactions, meshing the parts, setting loads and boundary conditions that has been established through sets or surfaces when the geometry was specified. The plug-in also defines the surfaces and reference points for setting the interactions. Sometimes it is more helpful for the user to set interactions themselves, this option is always available.

Since the choice of mesh varies, mesh refinements need to be made to the mesh to achieve a converged computational result even for a given single model. The user should mesh all the parts by choosing the meshing technique and the element types themselves. Supporting this effort is the reason the plug-in parts/components are decomposed, defeatured and partitioned in the plug-in under the directional and expertise of an experienced FE analyst. The resulting premium connection parts are ready for meshing but not meshed under the direction of the plug-in. The loads and the boundary conditions also vary based on the working environment and operational conditions being explored for the real connections. This application of loads and boundary conditions is setup for load and boundary condition definitions but is expected to be completed by experienced users. While the plug-in will define typical sets and surfaces to apply loads and boundary conditions, the user must apply them for meeting the objectives of a specific analysis.

The plug-in is to simplify the process of creating geometry used for forming an operational FE model. The plug-in has nothing to do with the computational methods associated with the model solution or post-processing of solution results. Use of the model created by the plug-in is up to the analyst.

The virtual test of the premium connection model in the research is only verifying that the thread in the model is working properly and finding the hand tight position. In future work, the model will be tested in a variety of work environments. For example, in the case of real design studies, the premium connections must work very deep underground with high temperature and high pressure for oil and gas transport. The sealing of the connection joint in this process requires assurance. In virtual test, the model will experience high temperature and high pressure as a boundary condition on the outside surfaces. As for loads, the model will have axial tension or compression loads on the reference points, as well as internal and external pressures on the surfaces. The premium connection model may experience bending during test or under in-service loads. Models created through plug-in should be able to support these boundary conditions and loads, and be able to perform this type of test.

1.5 Thesis Organization and discussion

The thesis write up follows the order stated below:

1. Chapter 3: Design concept for the parameter FE model plug-in. This chapter describes the design of the plug-in. Moreover, how the plug-in can achieve this project's goals.

2. Chapter 4: Details the design of the plug-in. This chapter provides the details and the steps used to create the plug-in, including the interface of the plug-in and the modules that control each step of the modeling.
3. Chapter 5: The application of the plug-in. Chapter 5 explains the steps associated with using the model created by running this plug-in and doing the virtual test to verify if the model works as expected.
4. Chapter 6: Summary and conclusion. This chapter summarizes the work in this research and recommendations for future work.

Chapter 2. Literature Review

The focus of this research is to develop a plug-in to create a full 3D parametric operational finite element model of an OCTG helical threaded premium connection for full contact and operational loading. Today, most of the models for premium connections are axisymmetric models created by hand each time to perform a specific analysis. The plug-in is developed to reduce working time and therefore costs, provide a model that can be operated on to validate or optimize the connection design and performance. Today, there is no plug-in to create a full 3D OCTG premium connection model, especially with 3D helical threads and full contact. The following sections provide a brief background on benefits of FEA for premium connection design and testing and the methods that engineers use to create the connection model.

2.1 Motivation for Studies FE models on Premium Connections

FEA is an important tool being used for the design, prediction of structural response as well as the prediction of metal-to-metal sealing of premium connections. A full 360-degree 3D finite element model can positively affect the design and optimization of OCTG premium connections. Users can also use the plug-in for trade-off studies, model verification and validation. FEA is an ideal computational tool for evaluating threaded connection designs because its numerical formulation enables analysis of complex geometries and materials[6]. Using the FE model in virtual test is indeed a good way to test the real quality of sealing. The ISO 13679 standard presents a detailed process for the OCTG connection performance testing. It has several series of physical tests. However, certain parts of the OCTG tests can be simulated by FEA models to ensure the safety of oil-well operations [7] reducing the number of physical tests required to certify the application of a particular connection design. The virtual test can save cost and time compared to the real physical tests.

2.2 Parametric FEA Modeling

Abaqus/CAE is the most commonly used commercial finite element code for designing and analyzing premium connections. If a model is only analyzed and calculated once, the interactive development of the FE model is no doubt the most efficient and effective way to produce a result. However, if the FE model is used to adjust the connection design or perform a further analysis based on the computational result, then it usually involves changing the model from the initial model and throughout almost every step in between. In the worst case, a trade-off study could require the development of a new model for every phase of the study. When this process is done multiple times, the work load can be huge since most designs cannot be optimized solely based on a few analyses or experiments. Most FE models developed are one-of-a-kind models. However, model updating, structural optimization and nonlinear predictions are applications using families of models with varying properties [8]. A better method for developing families of models is to develop a parametric model. Fan [9] created a parametric FEA model for vehicle brake and Jiapeng T [10] created parametric FEA models for aircraft wing structures. The parametric FEA modeling of OCTG premium connection is the goal of this research.

2.3 Scripting FE Models

Developing a modeling script to develop a finite element model semi-automatically is sometimes a very good solution. Python is the scripting language used in Abaqus. Python is a dynamic language often used for systems programming. The Python packages from NumPy and SciPy make Python a powerful language for scientific computing. The Abaqus/CAE kernel can be interfaced through the Abaqus Python scripting modules without interacting through the Abaqus/CAE GUI[11]. Once a set of modeling operations or techniques have been determined, they can be modified and stored in the script and the following modeling process can be repeated automatically through the script without using the Abaqus/CAE GUI. It is even more convenient to parameterize the script and use a single Python script to operate on relevant data for a model in order to achieve automatic updates on the model. However, a complete modeling procedure requires several steps, which often makes the script long requiring extensive documentation and effort for understanding. Users may need to spend a lot of time learning to use the scripting tool to find the corresponding code for each operation before modifying the user defined script. This can be difficult for most users.

2.4 Import FE model from CAD programs

In today's work flow, a 3D finite element model can be developed after importing 3D geometry components into the FE software from a CAD program. This process generates a nominal 3D geometric or solid model, which often does not allow any updates in sizes. However, many authors agree [12-14] that complex geometric models need a lot of adjustments, which makes the imported geometry used for the FE model hard to handle. In particular the 3D solid model was developed for purposes of design and/or manufacturing not for analysis. As a result there are many geometric features which may be needed for design or manufacturing that are not needed for analysis. Many of these design or manufacturing features complicate the geometry for meshing and solution. The typical approach in analysis is to "defeature" the CAD model, leaving the geometric features which impact the analysis. The work-flow for importing the CAD model into the FE software was to lower the time of modeling using CAD software which is more powerful and leverage the effort of building the model in CAD rather than the putting effort into more primitive modeling software such as Abaqus/CAE. However, once the adjustments to defeature a model are involved, we have to use CAD software to make adjustments and then import to the Abaqus/CAE again. This is usually more time-consuming. The larger argument to this workflow is that not all FE models need to be developed in 3D. Often times, scale of the geometries allow simpler mechanics to dominate the analysis resulting in much smaller models with significantly shorter runtimes with very good fidelity in the results.

2.5 Current State of the Art in Modeling Premium Connections

The FE models used currently for design and testing of the OCTG premium connections are axisymmetric models. Santi [15] and Hilbert [6] both used the axisymmetric models for their premium connection models. Comparing to the full 360 degree 3D models, axisymmetric models cannot turn the pin into the box to test the work of the helical thread. Despite the commonly accepted approach of using axisymmetric models to simulate the behavior of threaded connections, the make-up stage is not yet fully validated in literature [16]. Ahsan [5] designed a

full 360-degree model of the premium connection. The geometry, boundary conditions, loads, and stiffness distributions of the box and pin can be represented exactly [17].

Jialin Wang [18] designed a plug-in to create the Box Girder Bridge. M. Nesládek [19] designed a plug-in for the fatigue predictions based on the results of Abaqus FE analyses. James Ure [20] designed a plug-in for structural integrity assessments of components subjected to cyclic loading conditions. The plug-in created for Abaqus/CAE is used for modeling and analysis, which gives inspiration to design a plug-in for the full 3D finite element model of OCTG premium connections. An Abaqus/CAE plug-in program can automatically create models based on input parameters by the users and prepare the premium connection model for FEA computation with only a few operations. Developing a plug-in-based parametric finite element model leverages the time and effort put into finite element modeling toward using validated models supporting advanced analysis, design and optimization tasks.

Chapter 3. Design Concept for Connection Parametric FEA/Plugin

Developing a Premium Connection FE model from a script is one of the more convenient approaches among all the methods that are used to create the FE model. Even if users do not write the script themselves, they can run it if they need the same model. However, users need experience with the dominant mechanics associated with the model and script programing if they want a better understanding of the script and/or need to change it to achieve the functionality they want. If a plug-in program in Abaqus/CAE can be implemented to run scripts, and use pictures and instructions from the interface to help people understand the content of scripts, then it would be much easier for users than reading and understanding all the code in the scripts. The plug-in allows FEA experts to share their expertise on these types of models with others who do not have as much experience. The knowledge and experience of the FE analysts and connection designer are embodied in the plug-in and allow others to use that expertise.

Figure 3-1 shows how the Abaqus Scripting Interface commands interact with the Abaqus/CAE kernel.

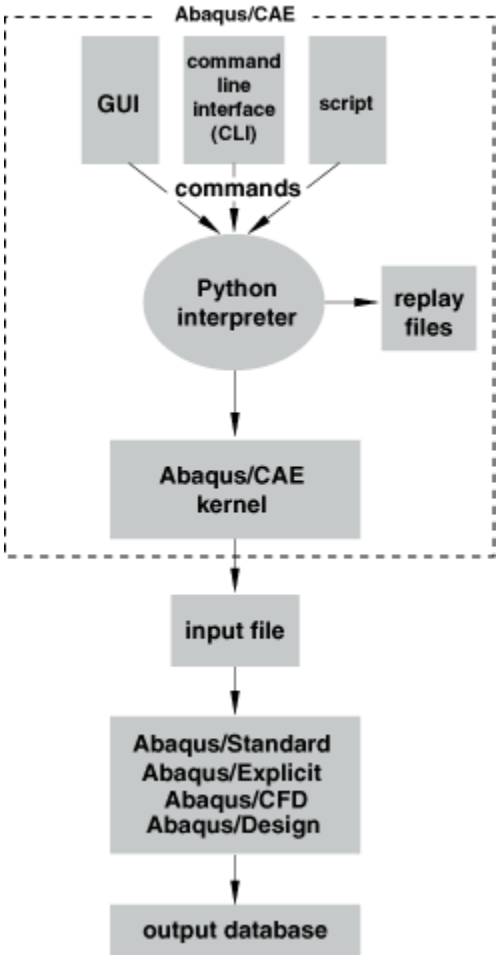

Figure 3-1: Abaqus Scripting Interface commands interact with the Abaqus/CAE kernel[21].

The Abaqus scripting interface provided by Abaqus's secondary development environment is written in Python, a powerful objected-oriented programming language. The syntax and operation of the Abaqus scripting Application Program Interface (API) is also written in Python. The Abaqus scripting manual [21] introduces the Abaqus Python development environment as well as all the Python commands corresponding to the commands in Abaqus/CAE. Depending on the models being analyzed, this project provides commands and code to implement the sequence and condition of instructions using control statements in Python. The commands and code also conform to the manual steps of building a finite element model for premium connections, which allows this plug-in program to replace manual modeling and retrieve parametric geometry that can be further analyzed. This parametric geometry is converted to an FE model in a few steps and used in simulation calculations.

The plug-in is responsible for simple functionality and model creation in Abaqus/CAE. This research focuses on how to program the plug-in for the chosen complex premium connection finite element model.

Before designing the plug-in, it is necessary to breakdown the process of creating the premium connection finite element model. All the steps should be done once to create the model in Abaqus/CAE, so that the Abaqus scripting python interpreter will record all the operations in the 'Abaqus.py' file in the work directory [22]. The 'Abaqus.py' can be modified by deleting useless code. For example, commands that are used to control the view are unnecessary in the modeling process. The remaining operations, followed by selecting and refining the relevant commands based on the design steps, finally result in structured code using the Python langrage.

As stated above and shown in the Figure 3-2, the plug-in should have the following functionalities:

1. Create the pin part and the functions as shown as the Table 3-1.

Table 3-1: The functionalities of the part 'pin'

1	Develop a constrained parametric sketch of the part
2	Assigning parametric values to the part
3	Partition the part for meshing
4	Defining sets
5	Defining surfaces
6	Defining material
7	Defining section properties
8	Instance the part in the premium connection assembly.

2. The box part is developed with the same functionality and process as the pin part.

3. All thread parts are almost the same as the pin part. Except for the subtasks above, the thread part requires a subtask to control the length of the thread which is defined by the thread pullout length on the technical drawing of the pin and the box.

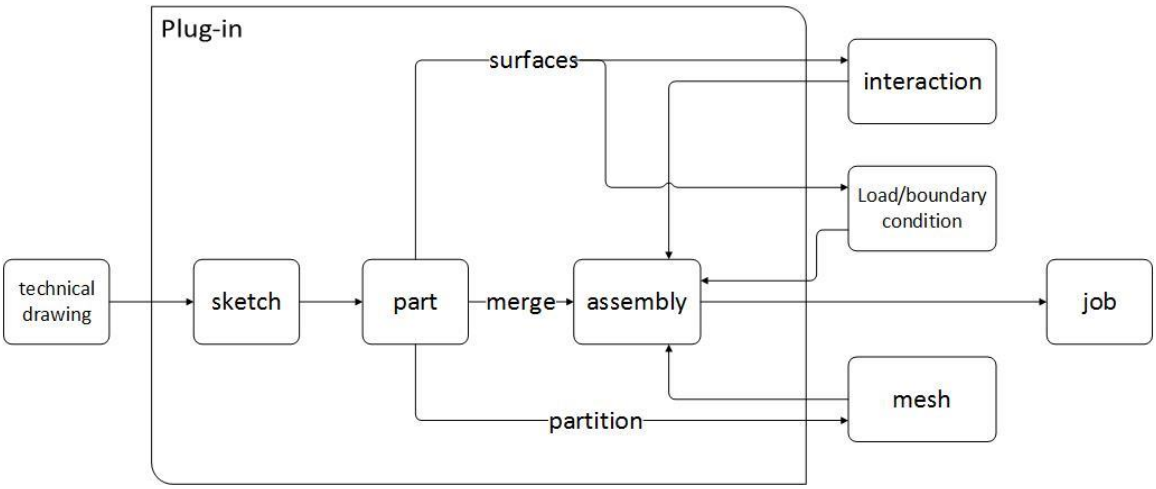

Figure 3-2: The functions that the plug-in implements in creating the 3D finite element mode.

The requirements of the plug-in are:

1. Parameterize all size, position material data. Design the plug-in to accept input from the GUI for further modification.
2. Program each independent function through an individual file as a Python module in order to allow the user to modify specific functions conveniently. Design a main program to achieve all the functions by calling these independent functions, making the plug-in modular and flexible. Each single script file, in this way, would not be too long to jeopardize readability.
3. Design the GUI of plug-in with proper layout arrangement. Add illustrations to define and use a reference to make the program user-friendly and easy to use.

Chapter 4. Detail design

This chapter shows how the plug-in is designed, including design of the interface of the plug-in and the Python code. The code has two parts: the main module which is set as the kernel of the plug-in, and an independent functional module that controls the building of the model.

4.1. Creating the Plug-in

4.1.1. Creating the interface

Before creating the functional modules for the project, the GUI needs to be created to execute all the modules, instead of running the script directly. The Abaqus GUI shows how the developer can use the plug-in functions by the icon image and the descriptive text. Most of the designers working with the GUI do not need to understand how the Abaqus Scripting Interface writes the commands. The important thing is to use the GUI to create and modify the models conveniently.

The GUI plug-in is written using the Abaqus GUI Toolkit and contains commands that create elements of a graphical user interface, which in turn send commands to the kernel. This section is about creating the GUI plug-in for this project.

The tool used in this research to create the GUI plug-in is called the Really Simple GUI Dialog Builder, shown Figure 4-1. The RSG (Really Simple GUI) dialog builder connects the commands written in the Abaqus Script Interface to the dialog boxes.

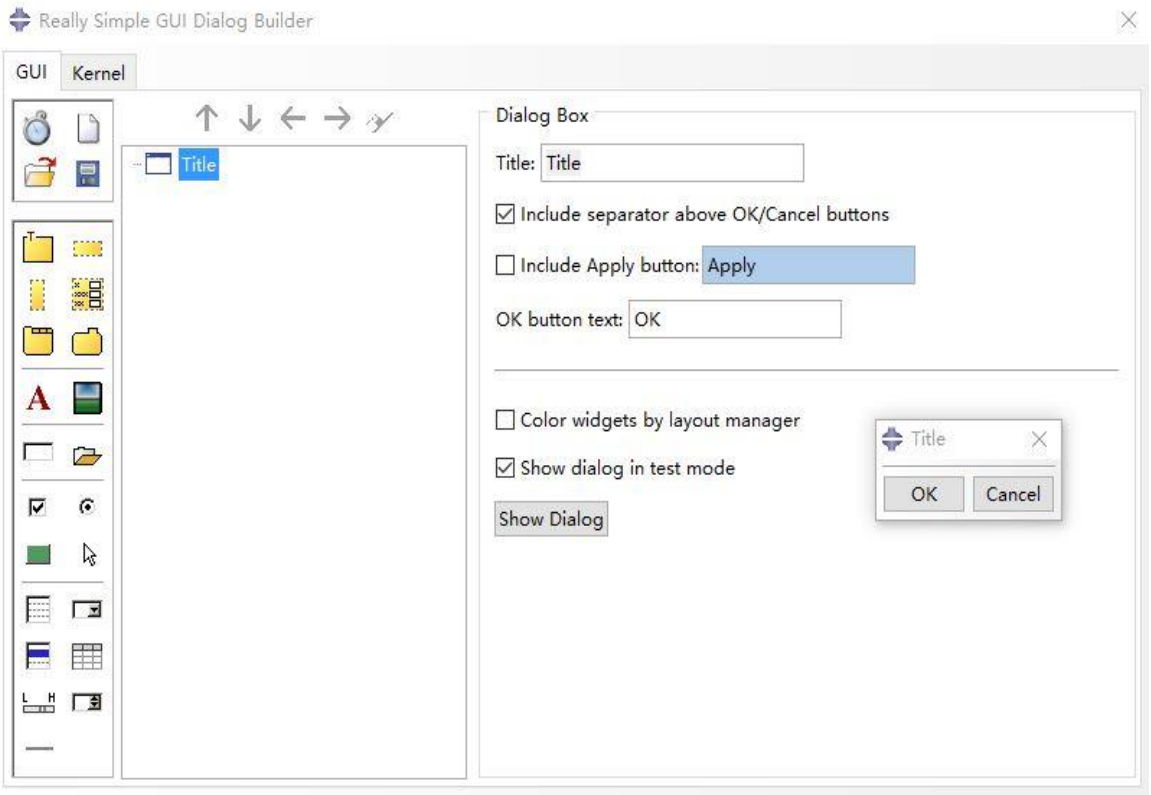

Figure 4-1: The interface of the GUI dialog builder.

There are two parts of a GUI plug-in. One is the GUI that allow the user to input the data and choose the functions. The other is the kernel. Only the main function is put into the plug-in's kernel.

The interface of the plug-in for the project is show as Figure 4-2 and the toolkits used to create the GUI are shown below:

1. The group box and the frame define the direction of the widgets, which can be vertical or horizontal. All the widgets are displayed in a vertical layout by default, unless the horizontal frame is used to change it. An effective arrangement of the widgets can make the interface of the plug-in user-friendly.
2. The tab book contains the tab items which provide a visible frame around a group of widgets that are displayed in a vertical layout. The premium connection plug-in defines a separate tab for each of the parts that make up the pin, the box with illustrations, and auxiliary text descriptions.
3. The text field is a text entry field that can be set to accept text, integers, or floats. This widget is the key of the plug-in. The nominal dimension data obtained from the technical drawings are assigned as the default parameters by the text field widget. The type of all the parameters is set as float, and the names should be the same with the parameters in the kernel module.
4. Check buttons are set to control the modules. The user can run part of the plug-in by using the check buttons. For example, the thread can be rebuilt for changing the size or the linear or angular dimensions independently. The control buttons in the tab of the pin and the tab of the box are "unchecked" by default so the features are not rebuilt automatically, so that the contact between the pin and the box don't need to set again. Figure 4.2 shows the interface of this plug-in.

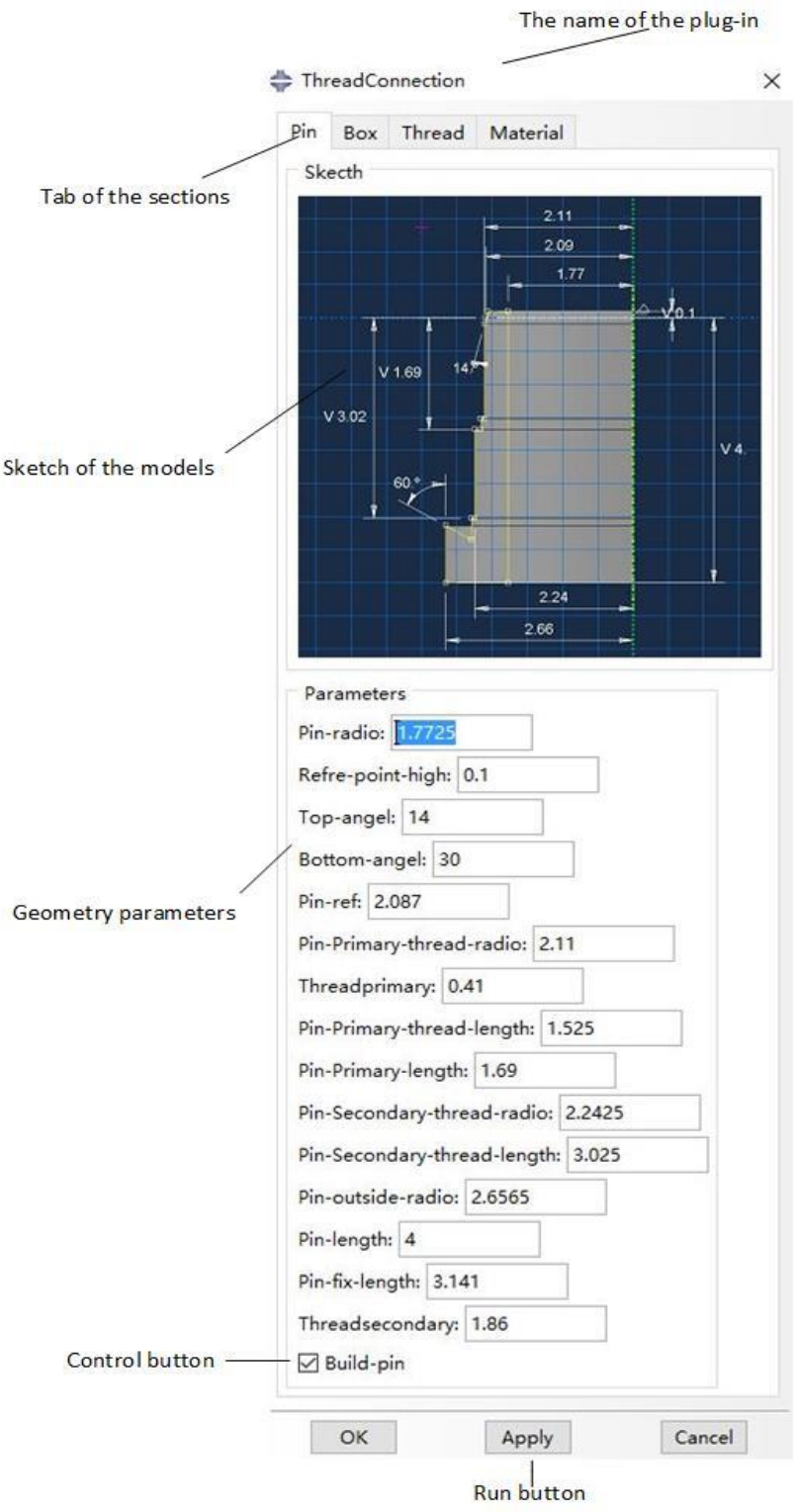

Figure 4-2: Interface of the premium connection plug-in in this research.

4.1.2. Creating the Kernel

The kernel of the plug-in is written in the Python language and has the '.py' extension. The kernel function is executed by dialog 'OK' or 'Apply' buttons.

The premium connection plug-in developed for this research is composed of only the main program in the kernel with a main function. The main program executes each step of the premium connection modeling process by calling the functions from the "premium connection module". The premium connection module must be added to the main program before using its functions by Python importing the functions.

The 'import' statement is used at the beginning of a Python program or module to import all of the Abaqus scripting modules required for modeling. The main program, and every module which call the functions defined in the Abaqus scripting libraries must import the appropriate modules. The parameters input from the plug-in interface are set as global variables. These parameters must have the same name in both the main program and independent modules for helping the users to understand and edit the code easily. The local variables are used only in the modules; these variables cannot affect other modules due to the Python language scoping rules. Variable names can be named simply, even have the same name since the name spaces for the plug-in follow the Python language scope rules for variable names.

The main program of the plug-in, shown in **Error! Reference source not found.**, only has a main f unction, due to the kernel runs only one function each time. The main function contains all of the arguments defined in the GUI building tab. Except controlling the geometry data, six arguments are added to control if the sections of the models are chosen to be built or not. The user is able to decide if only some of the parts, none of them, need to update after testing. For example, the threads can be rebuilt with another size without changing the size of the pin or box parts it belongs to, so that the contacts between pin and box do not need to be set up again. The sequence of 'if' constructions is not forced, the complete set of functions is required to run sequentially through the modeling steps.

4.1.3. Save the Plug-in to the Abaqus/CAE

After the plug-in is created, it should be saved to Abaqus/CAE to run. When the plug-in is saved, the kernel module and the kernel function will be moved to the plug-in directory and can be modified later. The plug-in created by the RSG tool can be saved as a RSG plug-in or as a standard plug-in [21]. Only the RSG plug-in can be edited by the RSG dialog builder. The RSG plug-in is used during programing to make sure the plug-in works as designed. It requires the user to be experienced in programing to edit the standard plug-in by a text editor. That is the reason the plug-in developed in this project is saved as a RSG plug-in, the users are able to add the modules to let the plug-in work as they plan conveniently.

The plug-in is in the menu 'Plug-ins' and has the name that is decided when the plug-in is saved. Every time when the plug-in is edited and saved, Abaqus/CAE should be restarted to renew the updated plug-in files.

Import all the modules

Define the main program 'build'

IF create the part 'pin' **THEN**

DO 'sketchpin'

DO 'buildpin'

DO 'pin_partition'

DO 'setsurface_pin'

DO 'merge_pin'

IF create the part 'box' **THEN**

...

IF create the part 'PinPrimaryThread' **THEN**

DO 'buildputhr'

DO 'setsurface_Pin_ThreadSealPrimary'

DO 'merge_Pin_ThreadSealPrimary'

IF create the part 'PinSecondaryThread' **THEN**

...

IF create the part 'BoxPrimaryThread' **THEN**

...

IF create the part 'BoxSecondaryThread' **THEN**

...

DO 'materials'

Figure 4-3: Pseudo-code of the main function used in the Kernel of the plug-in.

4.2. The pin modules

For the premium connection in this project, there are two physical parts, the pin and the box. Considering the complexity of the module and the difficulty of modeling the assembled connection, the premium connection model is divided to six parts created in Abaqus/CAE. The pin is created as part 'pin' without the thread on it. The thread is created as two parts, named the 'primary thread' and the 'secondary thread'. These three part instances are connected together in the assembly module with contact interactions to realize the same functional performance as the actual physical pin. The box is created in the same way. This section is about designing the pin. The details of creating all the threads are in the section 4.3.

4.2.1. Creating the Sketch of the pin

There are two ways to create a part in Abaqus/CAE, the first approach is sketching the geometric section of the pin and revolving the section to generate the solid during the creating step. The second approach is to create a sketch first and then use the sketch to create the part. The second method is adopted in this project. There are three reasons.

1. Analyzing the design drawing is always important. It takes a long time to understand and confirm the detailed features and size of the parts. Creating defeatured sketches, including pin, box and threads, and then creating the parts by using each of them make the modeling process well-organized and verifiable.
2. The sketch is saved as an independent sketch file, allowing it to be revised or reused as a template. It is convenient to check and update the sketch when the size of the part needs to be changed. The sketch can be used to create a new model which is consistent with the same part.
3. The code of this section can be easily divided into two parts, with each of them in two functional modules, if the sketches are created first, and then the parts are created. For the user that needs to learn the code and adjust the modules, using two modules that one for sketching and the other for creating is better than setting both the code about sketch and the code of the creating the part in only one module.

As the sketch shows in Figure 4-4, the pin is nearly a cylinder and can be created as a whole by using only one sketch to rotate. However, the pin also can be divided into several sector parts. These sector parts can be created independently and be merged as a complete pin part later. Comparing these two methods of creating the part 'pin', using only one sketch is better. There are three reasons:

1. Drawing the sketch of the part 'pin' is not so difficult compared to the real connection model, since the threads on the pin have already been removed.
2. Dividing the pin into several parts is better for meshing later, but creating the partitions when the pin is built by only one sketch has the same effect in meshing.
3. The decomposition of the pin into several geometry components increases the number of the sketches and the Abaqus parts which requires the programming to contend with more entities. The program also needs one more functional module to merge all the pin components, which is not needed when the pin is created as only one part.

For the reasons above, the part ‘pin’ is created as only one sketch shown in Figure 4-4 and the pseudo-code for this part is shown in the Figure 4-5. The part ‘box’ is created using the same approach.

The method used to create the sketch of the pin is to draw all the points needed to outline the part and order these points anticlockwise. The length of each line is controlled by the points whose coordinates should be input as the parameters. Each of the dimensions that are used to completely constrain the sketch needs to be represented by a parameter. The parameters control the update of the dimensions during optimizing the design. It is adequate to create the sketch by only those dimensions obtained from the engineering drawing. However, the methods developed in this project used to create the sketch needs to know the coordinates of the points defining the significant geometric features. Some of these feature points are created by the relative position to other points. All of the coordinates of the points are wrote by parameters in the code.

The python code that is executed by Abaqus/CAE to create the sketch is set up in the file called ‘sketchpin.py’.

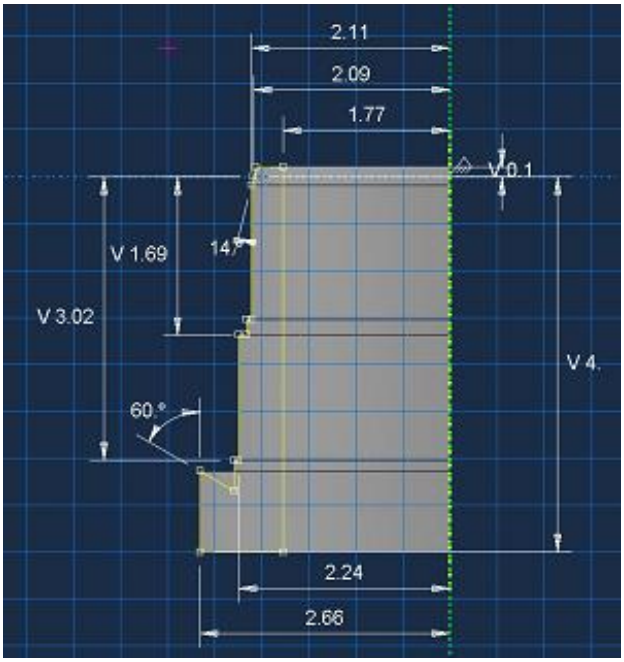

Figure 4-4: Sketch of the part ‘pin’ based on the technical drawing.

Begin SketchPin

Establish a Viewport for drawing the “new” Pin Sketch

Load the NominalConstrainedSketch_Pin

Get the NominalConstrainedSketch_Pin geometry, vertices, dimensions and constraint objects

Define NominalConstrainedSketch_Pin as the PrimaryObject to be operated on

Define a ConstructionLine for “purpose”

...

Save the NominalConstrainedSketch_Pin as ‘Pin_Sketch’

Unset the NominalConstrainedSketch_Pin as the PrimaryObject

End SketchPin

Figure 4-5: Pseudo-code of the module ‘SketchPin’.

4.2.2. Create the part ‘pin’

Both the pin and the box are created as a 90° component which will be merged to a full-360° part later in the assembly module. Compared to the full-360° parts, the 90° component is more convenient to work with when creating the partitions and setting the surfaces. The partitions and the surfaces are inherited after merging the four pin sector components to form a full-360° part. The part ‘pin’ is shown in Figure 4-6 and the code shown in Figure 4-7.

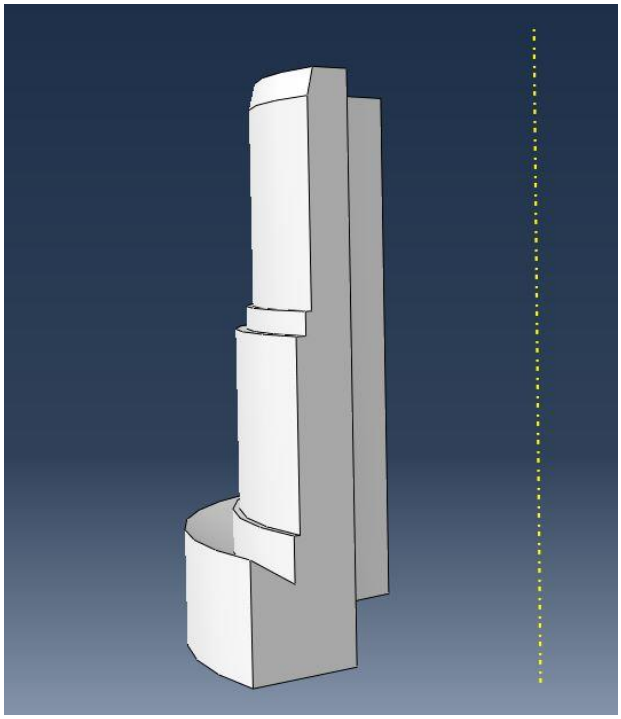

Figure 4-6: The 90° sector component part named ‘pin’.

```
Begin Buildpin
    Establish a Viewport for drawing the Sketch for the part 'pin'
    Load the sketch 'pin-sketch' from the model 'Model-1'
    Define a part project named 'pin', dimensionality is 3D, the type is deformable body
    Define part 'pin' as the PrimaryObject to be operated on
    Create the part 'pin' by using the sketch 'pin', the revolve angle is 90°
    ...
    Unset the part 'pin' as the PrimaryObject
End Buildpin
```

Figure 4-7: Pseudo-code of create the part 'pin'.

4.2.3. Datum plane, partition and set the surfaces

The pin is created by a single sketch. It is necessary to partition the parts into several regions for meshing. For the premium connection model, when the pin and box are connected, the regions where the pin and box parts are in contact are more important for mesh refinement than the other regions during the analysis. These regions should be the high mesh density regions. The other sections can be meshed coarsely. The partitions need to be designed and defined by mathematical expressions with user defined parameters. Designing the partitions this way allows the part partitions to be updated by parameters, the designer does not need to create or update the partitions by hand.

Creating the partitions in Abaqus/CAE requires the designer to choose the part volume, cell, to partition. Interactively this operation is done by using the mouse to pick the entities. The entities in a part, including vertexes, edges, faces and cells, are identified in the code by ID (identification data) numbers. All of the entities have unique id numbers which are issued by Abaqus after they are created. To complete the creation of the partitions by using the script, Abaqus needs to know which cell project is going to be partitioned, which can't be picked. The id numbers can be used to pick the entities to partition in the script. However, the rules that Abaqus uses to number the entities can't be known a-priori by the designer.

When the part has a simple shape and contains few entities, the id numbers of the entities are effortless to determine. Code can be developed to use the id numbers to select the entities directly. Section 4.2.3 shows the details associated with using id numbers to set up the contact surfaces of the thread.

There are many features in a part, if the part has a complex shape. The id numbers generated from the Abaqus.py file can't be used to control the entities directly, it does not matter whether the entities are vertices, edges, faces or cells, for two reasons. The id numbers may be different by changing the size or the location of the parts. The other reason is that different versions of Abaqus could change the rules to number the entities.

There is a function called '*findat ()*' to deal with this situation. This method returns the entities at the given coordinates[21]. In this method, the coordinates of the point that the script is using to locate the edges, faces and cells should not be shared with another entity. If the points are on the intersection of two entities, the method will return the first one that it encounters. This entity may not be the one that is needed. Even though the method returns the right cell at the first test, the result of the method may not always be the same if the geometry is modified. The coordinates can be defined as parameters.

The partitions of the pin and the box are illustrated in the Figure 4-8 and Figure 4-9 respectively. The geometry feature of the models is used to define the partitions used for meshing. The pin should have high mesh density regions, the coarse mesh regions and mesh transition regions. They are both partitioned to nine cells by performing eight partitioning operations. The sequence of the steps for creating the partitions is to divide the pin into two cells from the top surface along the y-axis first and then partition by datum planes seven times from the top to the bottom datum surfaces.

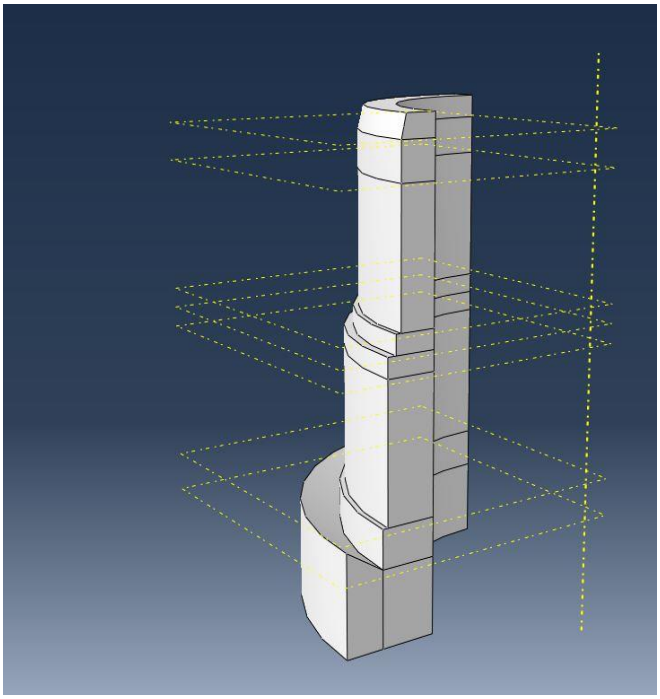

Figure 4-8: The partitions on the part 'pin'

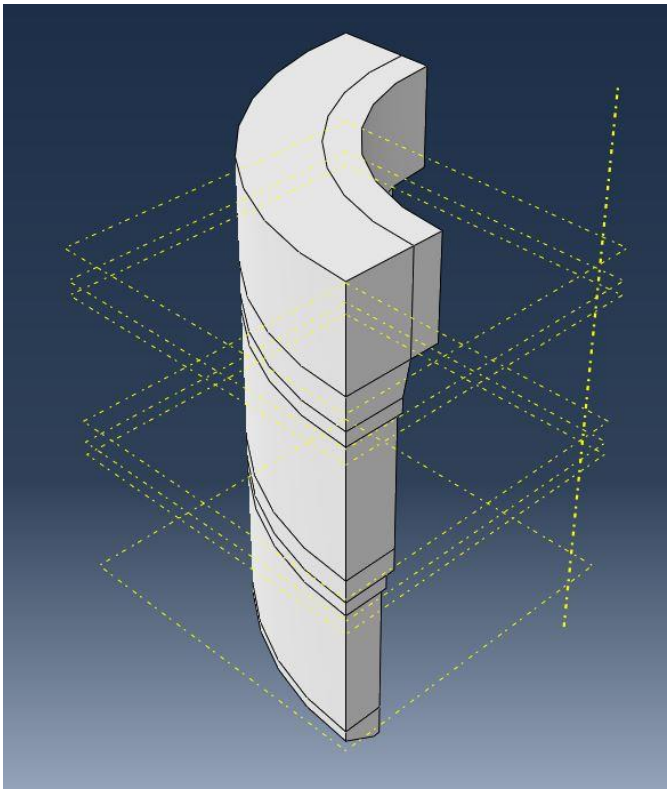

Figure 4-9: The partitions on the part ‘box’

The method of creating a cell by the ‘Extend face’ method has more advantages in the first step, shown in Figure 4-10. Comparing the edges or the points, the face on the part ‘pin’ used to partition is easy to determine by the *‘findat ()’* method.

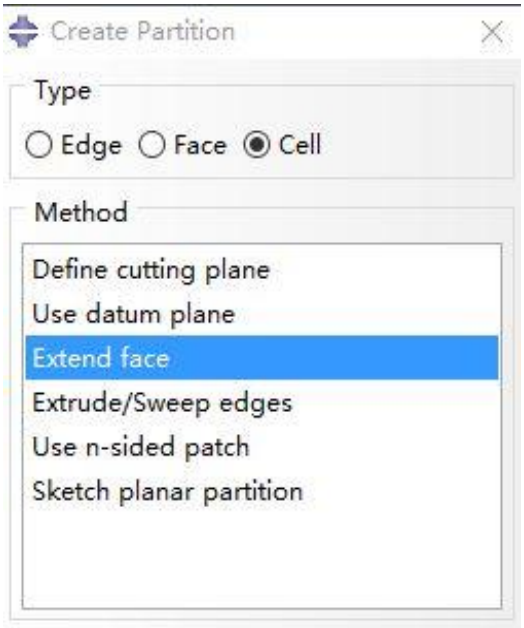

Figure 4-10: Method ‘Extend face’ is used to create the partition

For the later steps, the method ‘Use datum plane’ is better, shows on the Figure 4-11. The datum planes are created using the XZ principal plane and the offset which is set up as a parameters. The datum planes should be created in order, so that their ID numbers can be uniquely referred to later.

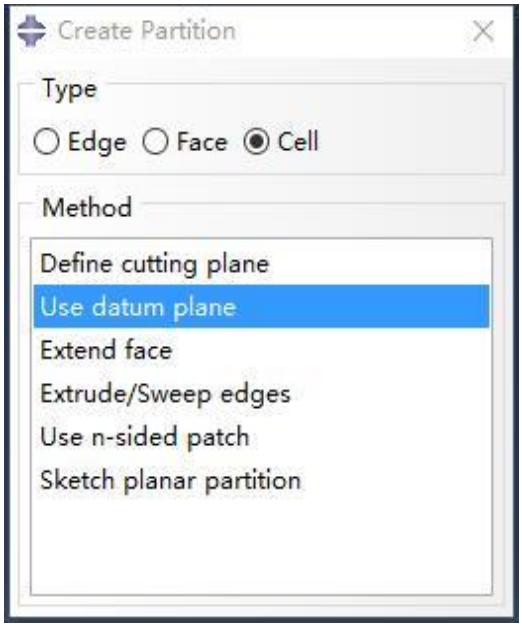

Figure 4-11: Method ‘use datum plane’ is used to create the partition

The local variables established by the geometric parameters from the main program are imported at the beginning of this module. The order of the local variables is according to the value of the coordinates of the datum planes relative to the y-axis, from maximum to minimum. This order is also used for creating the partitions. The pseudo-code of creating the datum planes on the pin shown in Figure 4-12.

Begin creating the datum planes on the pin

Load the pin part

#Define local variables [a1, a2...a7] by input parameters as the y-axis coordinates of the datum planes

Define variable [a1]

...

Define variable [a7]

Define part 'pin' as the PrimaryObject to be operated on

Create the first datum plane by using the principal plane 'XZplane' and setting the variable [a1] to be the offset

...

Create the seventh datum plane by using the principal plane 'XZplane' and setting the variable [a7] to be the offset

Unset the part 'pin' as the PrimaryObject

End creating datum planes on the pin

Figure 4-12: Pseudo-code of creating the datum planes on the pin.

The Abaqus scripting method that partitions a cell by datum plane requires two arguments, the cell and the datum plane, as Figure 4-14 shows.

The script of Figure 4-13 shows that the datum planes are numbered from number 2 to number. 8, datum plane number.1 is the datum axis object which is already defined when the part is created and is accessible. The datum plane id numbers can be used to define the datum plane when creating the partition. Access to the datum planes is defined in Figure 4-14.

As for the cells, the number of the cells increases every time after creating the partitions and Abaqus gives the id numbers to the new cells. The cells used in the subsequent partition steps are often the new one created in the previous step. Because of this situation, the id numbers of the cells needed are hard to verify. Instead of using the id numbers to define a cell, the '*findat ()*' method is necessary to resolve this issue by using the coordinates of the feature points. The command code of the function '*findat ()*' is shown on the Figure 4-14 and the pseudo-code for the script in this section is in Appendix A.

Begin partition the part 'pin'

Define a variable [c] for all the cells in the part 'pin'

Define a variable [f] for all the faces in the part 'pin'

Use the *findat ()* function to define the face in [f] for partition

Use the *findat ()* function to define the cell in [c] for partition

Partition the part 'pin' by extend face

Use the *findat ()* function to define the cell in [c] for partition

Partition the part 'pin' by datum plane [2]

...

Use the *findat ()* function to define the cell in [c] for partition

Partition the part 'pin' by datum plane [8]

End partition the part 'pin'

Figure 4-13: Pseudo-code of creating the partitions on the part 'pin'.

```
mdb.models[name].parts[name].PartitionCellByDatumPlane(cells,datumplane)
```

(a)

```
mdb.models[name].parts[name].datums[i]
```

(b)

```
cells = mdb.models[name].parts[name].cells.findAt(point1, point2, point3)
```

(c)

Figure 4-14: Command for the partition of a cell by datum planes (a); access to the datum planes (b); command of the function '*findat ()*' (c).

After the part is built, the surfaces used for contact need to be defined. There are many regions of the pin and box that require contact properties to be defined in this connection. The surfaces are set up for these contact interactions in the interaction module. If the surfaces are defined on the 90° part, the 360° part will also have these surfaces defined in the same location. The exact surfaces to be used in the interaction module are created after the pin and the box are partitioned. The partitions are defined by the geometric features of the model, the same approach is used for contact. The function of creating the partitions should be run before the function of setting the surfaces in the main function. The order of the functions run to build the fully partitioned pin can't be changed for this reason.

The surfaces on the part 'pin' are shown in Figure 4-15. The numbers used to define the surfaces in the command code regarding creating the surfaces in the part objects are the id numbers. The shape of the pin will not change a lot when the sizes and the angular dimensions are different, as a result the number of the faces in the part 'pin' will be the same. Abaqus will use the same way to name the surfaces of this part every time. There are many coordinates of the feature points need to be calculated if the '*findat ()*' function is used to define the surfaces by picking three different points for each surface. The method '*findat ()*' is not necessary in this section for this reason. The pseudo-code of setting the surfaces on the part 'pin' is shown in Figure 4-16.

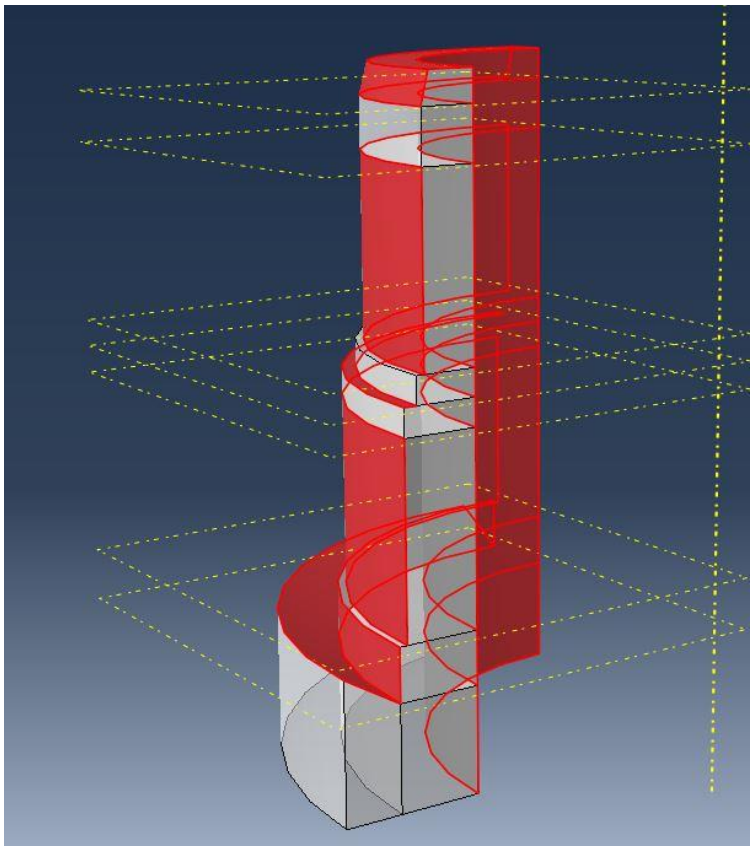

Figure 4-15: Surfaces on the part 'pin'.

Begin setsurface_pin

Define the function setsurface_pin

Define the variable [p] as the part ‘pin’

Define the variable [f] as all the faces that be subject to the [p]

Establish a Viewport for the part ‘Pin’

Pick the faces in the [f] by the ID numbers

Using the picked faces from the [f] to create a new surface and saved as 'Pin_ThreadSealPrimary_base'

Pick the faces in the [f] by the ID numbers

Using the picked faces from the [f] to create a new surface and saved as 'Pin_ThreadSealSecondary_base’

Pick the faces in the [f] by the ID numbers

Using the picked faces from the [f] to create a new surface and saved as 'Pin_shoulder'

Pick the faces in the [f] by the ID numbers

Using the picked faces from the [f] to create a new surface and saved as 'Pin_top'

Pick the faces in the [f] by the ID numbers

Using the picked faces from the [f] to create a new surface and saved as 'Pin_bottom'

End setsurface_pin

Figure 4-16: Pseudo-code of setting the surfaces on the part ‘pin’.

4.2.4. Merge

The operations defined in this section are responsible for turning the 90° part into the 360° part. The full 360° 3D object is named ‘pin-merged’ and is established at the initial position in the assembly module for the later work.

A new full-360°-3D instance is created every time after running the plug-in. The user can rename part names for themselves and this command to change the part name can be added to the plug-in at a later time. However, no matter how the name of the merged part is defined, Abaqus applies an instance number to this new part instance by changing the name to ‘name-1’ in the assembly module, as the Figure 4-17 shows. There will be the part instances named with ‘name-2’, ‘name-3’, and ‘name-4’. The designer should delete the old part instance name by hand every time. Because the instance name has changed, the contact interaction needs to be set up again. That’s why an IF-conditional statement [23] is necessary. This IF-conditional statement is written before the function calls for assembly, shown in the Figure 4-18. If the “name-1” already exists, it will be deleted and the new instance is named as “name-1” after running the script. However, if the delete command is written without the IF-conditional statement, the script will be stopped by the

error that the “name-1” does not exist. This IF-conditional statement is added to all the assembly modules.

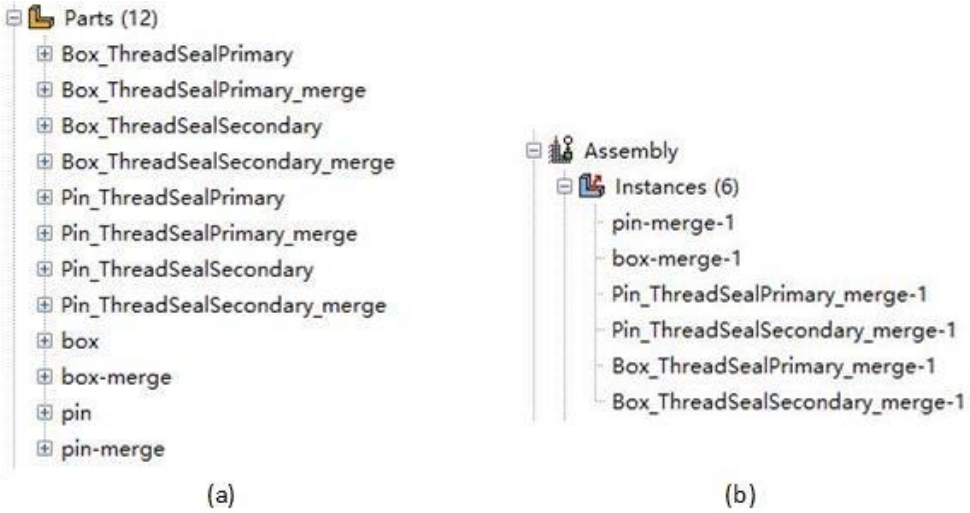

Figure 4-17: Instances’ name in part module (a); instances’ name in assembly module (b).

Begin merge_pin

Define the function merge_pin

Define the variable ‘a’ as the assembly object for the model

Establish a viewport for the ‘a’

Get the ‘a’ features

IF the feature Pin-merge-1 is already existed in the ‘a’ **THEN**

Delete the ‘pin-merge-1’ if it existed

Create a Feature object and a DatumCsys object from the specified default coordinate system at the origin

Define a variable ‘p’ as the part ‘pin’

Create a PartInstance object ‘pin-1’ and put it into the instances repository

Create three PartInstance objects ‘pin’ in a radial pattern around the y-axis and set the total angle as 360°

Pick all the PartInstance objects from the instances repository

Merge all the PartInstance objects and set the name as ‘pin-merge’

End merge_pin

Figure 4-18: Pseudo-code of creating the full 360° 3D part ‘merge-pin’.

4.3. The modules of the Threads

There are four thread instances that need to be created. The difference between each of the thread forms is the sketch and the assembly location, but the steps to create each thread part is the same. This section illustrates the process with the part 'Pin-primary-thread' as the example.

4.3.1. Sketch

The thread is created by drawing a cross-section sketch of the thread first, the same as creating the pin and the box. The technical drawing for the thread in this premium connection is shown in Figure 4-19. There are some modifications required for defeaturing the thread geometry. The fillet radius of the thread needs to be deleted as it introduces a lot of complexity for meshing but provides little information toward the mechanical response of the connection, as Figure 4-21 shows. The number of the elements for the thread with fillets is far more than the defeatured thread, as Figure 4-21 shows. Only the stiffness associated with the thread and the surfaces of the thread used for transferring loads from contact are important in this model. To mesh the full thread with the fillet radii is not necessary. All the threads are defeatured for turning the sketch to a quadrilateral. Compared to the actual cross-section of the full thread, the defeatured thread can be sketched easier and needs fewer commands and parameters in the code, shown in Figure 4-20. The code has the same commands with creating the pin and the box, using parameters which define the coordinates of the points and drawing the lines between the points to make the sketch closed.

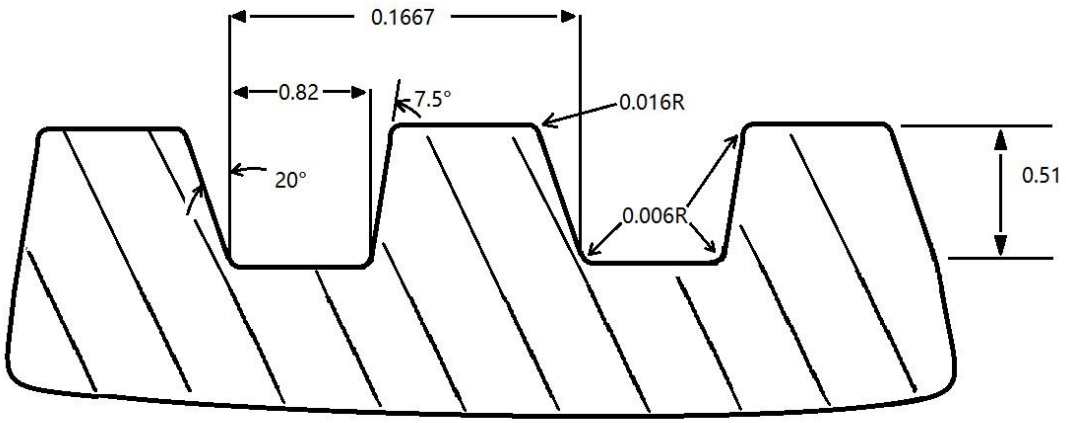

Figure 4-19: Technical drawing of the threads in the premium connection model in this research.

Begin build_pin_primary_thread

Define a function buildputhr

Establish a Viewport for drawing the “new” Pin_primary_thread Sketch

Load the NominalConstrainedSketch_ Pin_primary_thread

Get the NominalConstrainedSketch_ Pin_primary_thread geometry, vertices, dimensions and constraint objects

Define NominalConstrainedSketch_ Pin_primary_thread as the PrimaryObject to be operated on

Define a ConstructionLine for “purpose”

...

Save the NominalConstrainedSketch_ Pin_primary_thread as 'Pin_ThreadSealPrimary'

Unset the NominalConstrainedSketch_ Pin_primary_thread as the PrimaryObject

End SketchPin

Figure 4-20: Pseudo-code of the module create the sketch of ‘pin_primary_thread’.

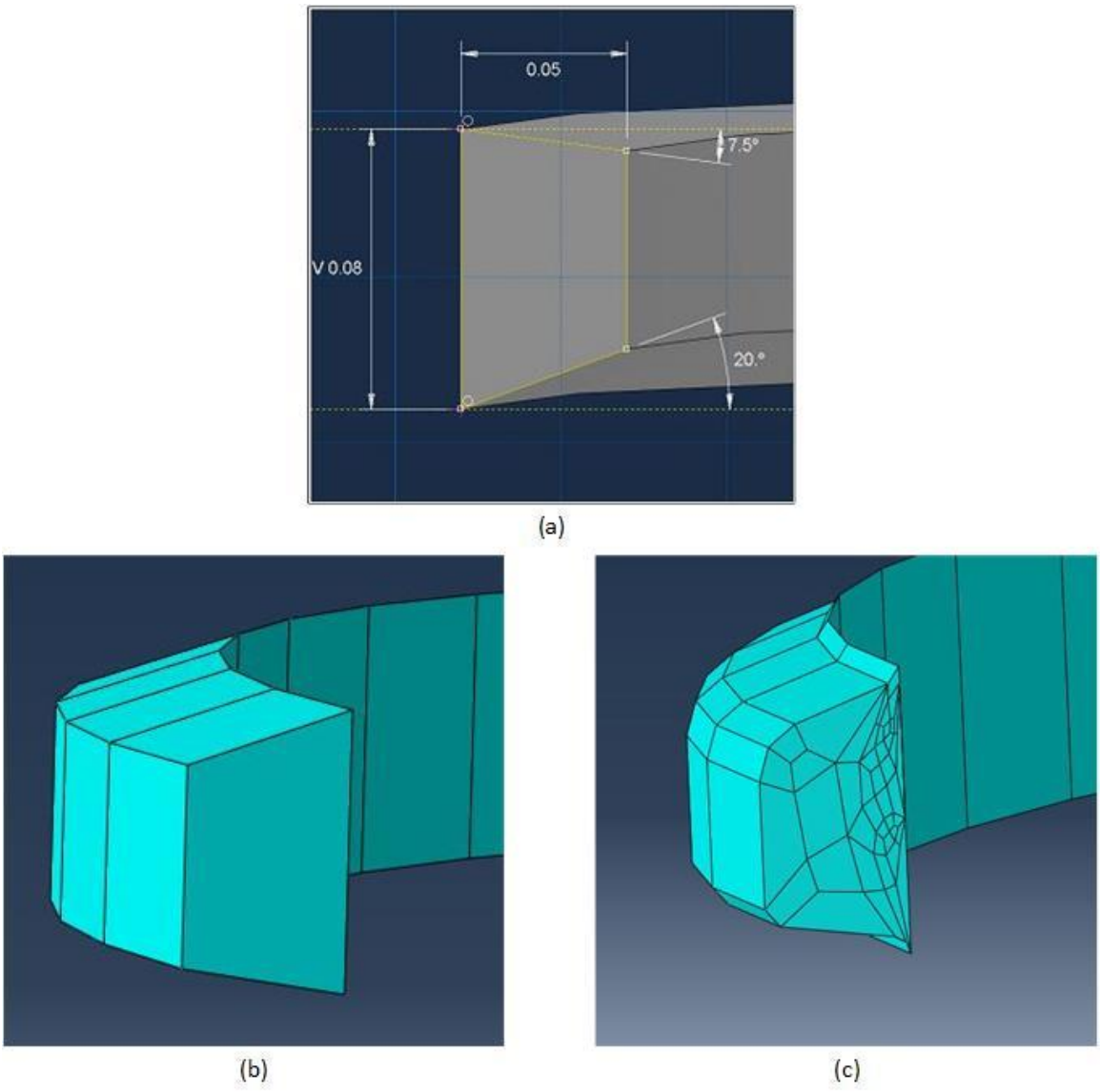

Figure 4-21: Sketch of the threads that is used in the plug-in (a); mesh of the thread without the fillet (b); mesh of the thread when the fillet is included (c).

4.3.2. Part 90 degree

There are two methods for using the sketch to create the 90° sector of the thread. The first one is created using four sketches to fit each thread at different locations in the connection. The commands for this operation require all the coordinates associated with the initial position, therefore, the points should be written by parametric expression. The second method, only one sketch needs to be created. Because the sections for all four threads are the same, all the threads can be created by this single sketch through repositioning the thread part to the desired location on the pin and box in the assembly. However, compared to the first method, the second approach requires more commands to achieve the finished part and the coordinates for the initial position for each thread still need to be calculated. This project uses the first method for the sketch of thread modules.

There are two ways to create a thread by rotating a sketch. One is to rotate the sketch through a prescribed rotation and helix angle to directly define the thread. In this method, the degree of rotation of the thread is derived from the length of the thread. The second way is to create a 90 ° thread as the base unit, and then calculate the required number of units by the length of the required thread, and merge these units into a complete thread in the assembly module. The first method is simple when creating threads. However, the thread created by using the first method has a lot of problems in setting the details and calculation. For example, the surface of such a thread will be divided into many parts by Abaqus, as Figure 4-22 shows. It is difficult to select these surfaces by commands referring to name when setting the thread's contact surfaces. The second method creates a thread that does not cause this problem and is used in this research.

Compared to the modules that create the pin and the box parts, the greatest difference of this module is the command that is responsible for revolving the thread sketch to a 3D solid part. As all the threads start revolving from the shoulder, the primary thread of the pin or the box should revolve anticlockwise and upward. The secondary thread should revolve clockwise and downward. The directions are constant. The pseudo-code in Figure 4-23 defines the process of building the threads.

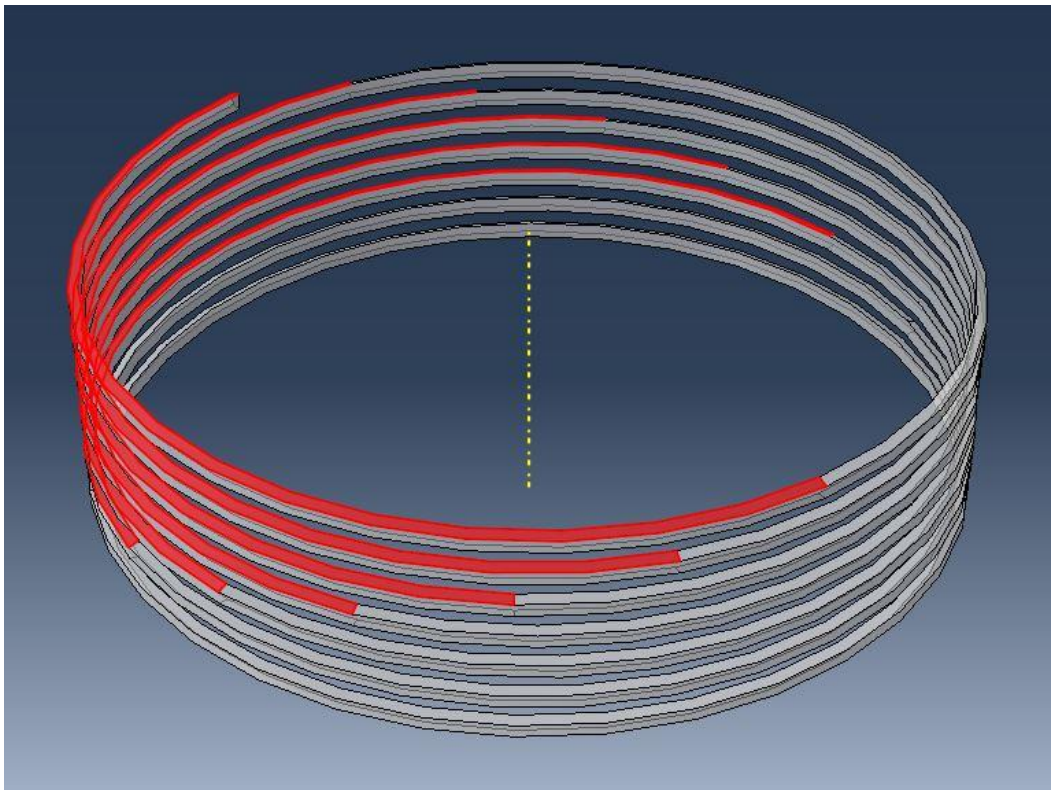

Figure 4-22: Faces in the thread object when it is created by rotate the sketch once

Begin Buildputhr

Define a function buildputhr

Establish a Viewport for building the part Pin_primary_thread

Load the Sketch Pin_primary_thread from the 'model-1'

Get the part Pin_primary_thread geometry, vertices, dimensions and constraint objects

Define Sketch Pin_primary_thread as the Primary Object to be operated on

Define a ConstructionLine for "purpose"

Create the part 'pin_primary_thread' by using the sketch 'pin_primary_thread', the revolve angle is 90°

...

Unset the part 'pin_primary_thread' as the PrimaryObject

End Buildputhr

Figure 4-23: Pseudo-code of create the 90° component part 'pin_primary_thread'.

4.3.3. Set up surface for interaction

There are four surfaces in the thread object that need to be defined for setting the interactions. The surfaces are shown in Figure 4-24. The 90°-basic component of the thread is a hexahedron and only has six surfaces defined as contact surfaces. Since the shape of the thread won't change by a huge difference when adjusting the size, not only the lengths, but also the angles, the surfaces can be easily obtained using the id numbers. The coordinates of the points on the thread are not easy to calculate because of the thread pitch when revolving the sketch. As a result, the '*findat ()*' function is not applicable to the thread module.

After exploring several different approaches, it was discovered that the id number of the surfaces is always the same when the steps to draw the section of the thread in its sketch is immutable. It doesn't matter if the thread is on the pin or the box. As the shown in the code in Figure 4-25, the top surface of the thread's id number is '#2', the bottom one is '#8', and the two flank surfaces' id numbers are '#1' and '#4'. The remaining two surfaces are not considered for use as contact surfaces. The id numbers of these faces in the part 'pin_primary_thread' are used to define the surfaces. This approach to selecting the faces of the thread for assignment for contact does not require ID calculating the coordinates of the points on the face, which reduces errors in the code.

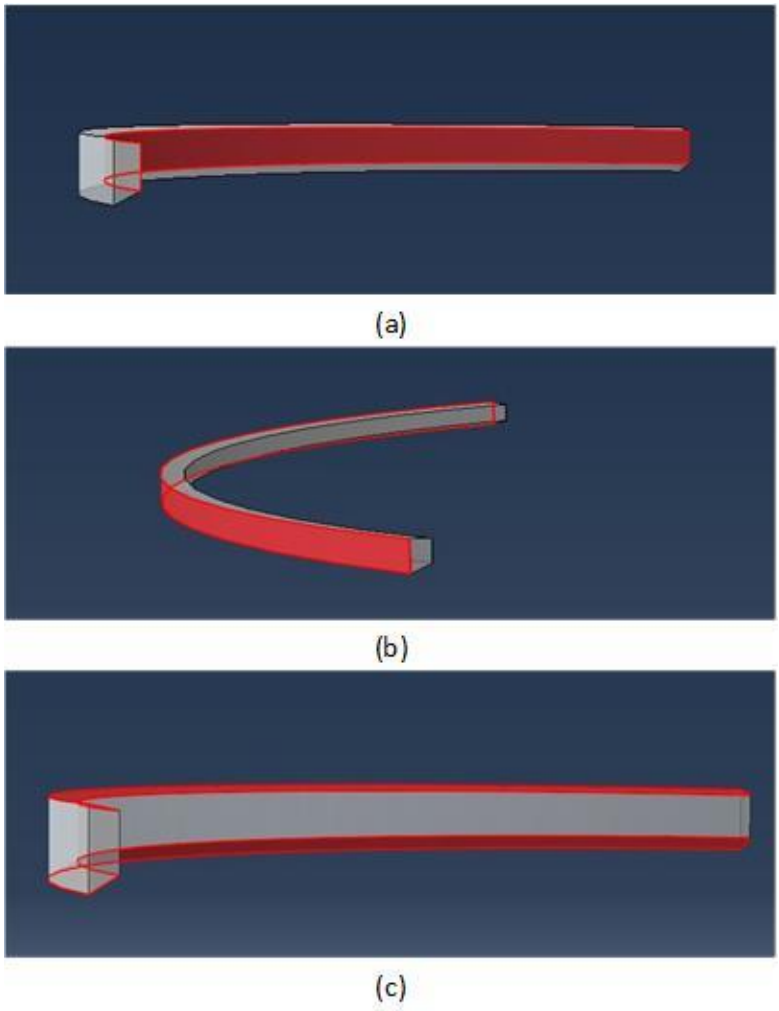

Figure 4-24: Surface ‘Pin_ThreadSealPrimary_top’ (a); surface ‘Pin_ThreadSealPrimary_base’ (b); surface ‘Pin_ThreadSealPrimary_flank’ (c).

Begin set surface of the primary thread of the pin

Define the function setsurface_pin_ThreadSealPrimary

Define the variable ‘s’ for all the faces

Pick the face ‘#2’ in ‘s’ and named as ‘Pin_ThreadSealPrimary_top’

Pick the face ‘#8’ in ‘s’ and named as ‘Pin_ThreadSealPrimary_base’

Pick the face ‘#1’ and ‘#4’ in ‘s’ and named as ‘Pin_ThreadSealPrimary_flank’

End set surface of the primary thread of the pin

Figure 4-25: Pseudo-code of set the surfaces of ‘pin_primary_thread’.

4.3.4. Assembly

The number of turns of the thread should be controlled by the thread pullout length of the pin and the box which is a parameter defined on the technical drawing of the connection. The complete thread is merged from several 90° sectors, so that it should be able to add the new sectors or delete the sectors already existed to change the turns of the thread. Abaqus/CAE doesn't allow the user to edit the merged part. Each sector of the thread needs to be adjusted in position after being added to the premium connection assembly in the assembly module. The merge command can only merge components that already exist. Changing an input parameter to command Abaqus/CAE to create a new-merged thread part with the turns determined and position of the sectors parametrically revised, is not currently supported by the development in this study. This project uses an approximate method to resolve this objective. However, this method has a range on the number of the circles of thread that can be adjusted.

It's important to understand how the assembly module of Abaqus/CAE works. Each instance can be oriented and positioned in the assembly, which includes assembly constraints in the interface. To achieve the goal that the turns of the thread can be revised to fit the pin or the box, the thread basic units should be created with more thread length than what is actually needed. For example, if the pullout length of the thread on the pin requires six full revolutions of the thread, seven revolutions of the thread are created. The code used to create these units and adjust their position and orientation to the designated location is imperative to properly transferring load across the joint and preloading the seals of the premium connection finite element model. A group is created that contains five circles of the thread basic units which is used to be the lower limit of the thread pullout length. These basic units are part of the merged thread no matter what the length of the thread is. This group is set as a variable for the merge command, so that all these basic units' names will not be written in every conditional statement code, As Figure 4-26 shows.

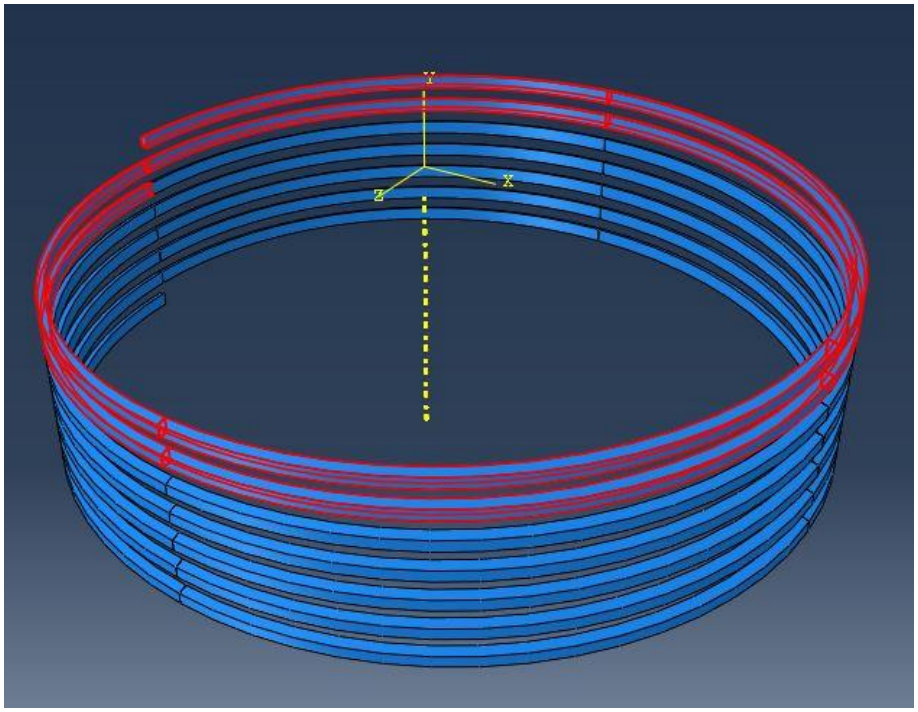

Figure 4-26: Five turns of the thread basic units be set as the lower limit of the range.

A local variable ‘n’ is set as criterion, the quotient of the pullout length and the pitch of the thread, used in the plug-in to determine how many of the basic units are needed to build and merge into a complete thread. Beside the group, there are new basic units added to be a part of the merged thread as the variable ‘n’ getting bigger. Because the basic units of all seven circles are created before, the redundant units need to be deleted after the merged thread is created. The range of the numbers of the thread revolutions is set from five to seven. The users are able to alter this range by editing the module in accordance with their wishes. The pseudo-code is shown in Figure 4-27. The pin-threads’ locations are shown in the Figure 4-28 and the box-threads’ locations are shown in the Figure 4-29.

```
Begin merge_pin_primary _thread:
    Define a variable ‘n’ to decide the length of thread on the part ‘pin’ and the part ‘box’
    #n equal to the length of the pullout of thread divide the pitch of the thread
    IF n≥5.5 and n≤5.75 THEN
        The variable ‘mergeparts’ equal to the instances that already picked before
        Define the ‘mergeparts1’ as the ‘mergeparts’
        Merge the thread
            Set the name as 'Pin_ThreadSealPrimary_merge,
            The picked instances is ‘mergeparts1’,
            The originalInstances is ‘DELETE’,
            The mergeNodes is ‘BOUNDARY_ONLY’,
            The nodeMergingTolerance is ‘1e-06’,
            The domain is set as ‘BOTH’,
            Delete the Features that doesn’t need
            (('Pin_ThreadSealPrimary-1-rad--lin-1-6-1',
            'Pin_ThreadSealPrimary-1-rad--lin-1-6',))
    ELSE IF n>5.75 and n<6:
        ...
    ELSE IF n>6:
        The variable ‘mergeparts1’ is set as the variable ‘mergeparts plus the instances
        'Pin_ThreadSealPrimary-1-rad--lin-1-6', 'Pin_ThreadSealPrimary-1-rad--lin-1-6-1',)
        IF n>6.25:
            ...
            IF n>6.5:
            ...
End merge_pin_primary _thread
```

Figure 4-27: Pseudo-code of creating the primary thread of the pin.

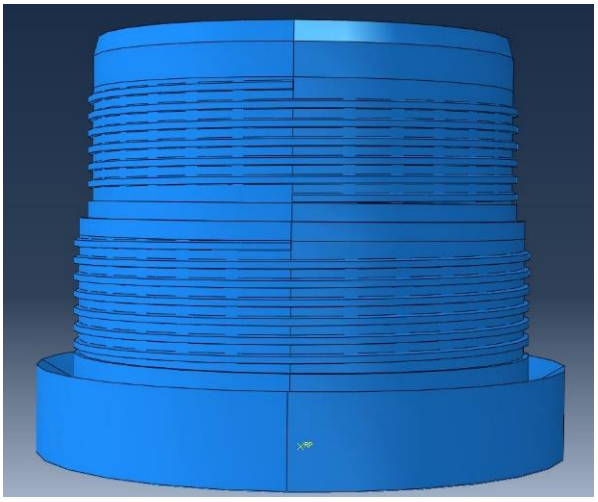

Figure 4-28: The primary thread and the secondary thread created from the shoulder of the pin.

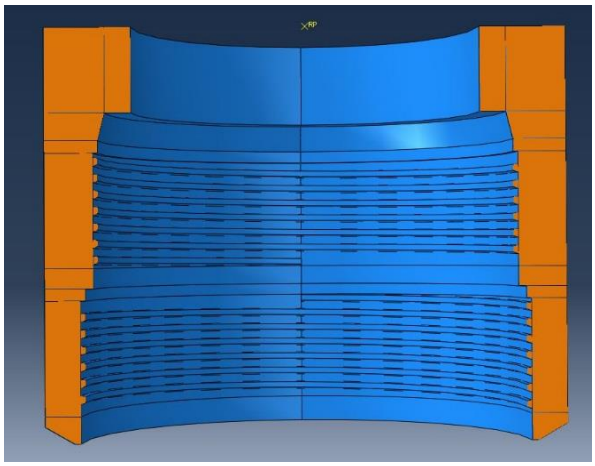

Figure 4-29: The primary thread and the secondary thread created from the shoulder of the box.

4.4. Material

After the model is assembled, the material data of the model is assigned to the parts. The analysis for this model is elastic. The material data used for linear elastic analysis are Young's modulus and Poisson's ratio. According to Hooke's law, the stress is proportional to the strain in the elastic region of the material response. The change in the stress over the change in the strain in the linear elastic range is called the elastic, or Young's, modulus of the material. Young's modulus is a material parameter which is an indicator of the rigidity of the material, the greater Young's modulus, and the less prone to deformation. Poisson's ratio is a measure of the Poisson effect, the phenomenon in which a material tends to compress in mechanics directions perpendicular to the direction of tension[24], as the equation (4-1 shows.

$$\nu = -\frac{\mathrm{d}\varepsilon_y}{\mathrm{d}\varepsilon_x} \qquad (4\text{-}1)$$

The overall focus of the project is parametrical modeling. There is no physical experimental data to validate the material performance for this particular premium connection. The data

representing the material properties are provided. The plug-in is required to allow the user to change input data to represent the material behavior of the connection being studied.

The material properties are defined through the Abaqus/CAE Material Module interface. Two section properties are created and assigned the premium connection material model, one for box and the other for pin. These sections were set for each part. It is necessary to use the 'if' statement to determine whether the part exists before performing the assignment of the material properties to each part. In some cases the plug-in is only used to create one part, the function will report an error when the module runs without the 'if' statement in this situation. The code for defining the material data can be put in the creating material module of each part. However, this is different from the original intention of this project that each module only involves a specific function. The code associated with setting the materials is shown in the Figure 4-30.

Begin materials

Define the function 'materials'

Define the name of the Material of the parts

Create the elastic objects of the material and named as 'mn'

Create a HomogeneousSolidSection object for box and named as 'box', set the material as 'mn' and the thickness as 'none'

Create a HomogeneousSolidSection object for pin and named as 'pin', set the material as 'mn' and the thickness as 'none'

IF the part 'box' is created **THEN**

Define the variable 'p' as the part 'box-merge'

Define the variable 'c' as all the cells in the 'p'

Set the section named 'box' to the part 'box'

IF the part 'pin' is created **THEN**

...

IF the part 'pin-primary-thread' is created **THEN**

...

IF the part 'pin-secondary-thread' is created **THEN**

...

IF the part 'box-primary-thread' is created **THEN**

...

IF the part 'box-secondary-thread' is created **THEN**

...

End materials

Figure 4-30: Pseudo-code of the materials module.

Chapter 5. Application

To verify that the model created by the plug-in is a model that can perform analysis tasks, it is necessary to observe whether the test results are consistent with the experience with the make-up and loading of actual tested premium connections. It is also necessary to modify the dimensions of the model and change the data that are input from the plug-in interface to demonstrate that the plug-in can achieve the desired functionality.

The model created by using the plug-in is only a geometric model. There are additional modeling steps before the FE model is completed for execution:

1. Setting the interactions between the parts,
2. Setting the loads and boundary conditions,
3. Effective meshing the model.

The plug-in developed in this study setups up the premium connection model so that the focus can be on analysis steps, loads and boundary conditions to be analyzed through FEA.

5.1. Material model

In the simulation test, the material properties of the connection model need to be set. The module of adding material properties to the model has been designed in the plug-in. The user can input the data based on the real model properties. The focus of this project is to verify the feasibility of the model, so a specific material property is selected for demonstrating the analysis of this premium connection. The specific data are defined in Table 5-1.

Table 5-1: Material properties of the model in this project

Box/Pin Material Properties		
Elastic Properties	Young's Modulus(psi)	Poisson's Ratio
	3.00E+07	0.292

The material property is used to define two sections which are named ‘pin’ and ‘box’, and then the sections are assigned to the corresponding part objects, show in Table 5-2：

Table 5-2: Sections of the part object that contain material property

Name	Type	Part
Box	Solid, Homogeneous	Pin_merge-1, Pin_ThreadSealPrimary_merge-1, Pin_ThreadSealSecondary_merge-1
Pin	Solid, Homogeneous	Box_merge-1, Box_ThreadSealPrimary_merge-1, Box_ThreadSealSecondary_merge-1

5.2. Analysis steps

For models that have already been completed, some final settings are required before performing the simulation calculations. Once the user executes the plug-in, the next step is to define the analysis steps. The analysis step typically involves analysis of process selection, load selection, and output request selection. Each analysis step can use different load, boundary conditions, analysis process and output requirements. To verify the premium connection FE model created by running this plug-in works as expected, at least two steps for testing are required.

1. Testing the function of the thread. The pin is rotated into the box with a constant angular displacement and whether the rotation of the pin is affected by the thread can be validated by calculating the displacement change of the pin in the y-axis.
2. Testing the sealability of this connection model. The changing curve of the stress on the seal section of the model, the moment and the displacement on the y-axis of the pin can verify if the model works as expected.

The complete testing process consists of three steps:

1. The initial step
2. The rotation step
3. The sealability testing step

The first step is the initial step of the model, the contact between the various parts need to be set to make the model ready for testing in this step. The second step is to test whether the model works according to the thread in the rotation. The third step is to verify the sealability of this model. The loads and boundary conditions are different in the second and the third step, and as a result need to be set separately. The option 'Nlgeom' is selected in both steps for the nonlinear effects of large deformations and displacements and the option 'Time' is set as one time period. The steps is show as the Table 5-3

Table 5-3: Steps in the virtual test.

Name	Procedure	Nlgeom	Time
Initial	(Initial)	N/A	N/A
Rotation	Static,General	ON	1
MakeUp	Static,General	ON	1

5.3. Interactions

The next step is to set the interactions between the part components of the premium connection model. After the creation of the geometry model is complete, the model is set in accordance with the initial position in the assembly module, but only the relative position between components are determined. The model should be working as the real physical connection. The pin and the box should connect their own threads. Contact interaction must be defined for all the surfaces within the connection assembly that interact with each other at any time during the analysis [5]. Interactions need to have three subsections:

1. Tying the threads to the base. Including the primary and the secondary threads of the pin and the primary and the secondary threads of the box. The thread should be set as *Tie' as shown as the Table 5-4.

Table 5-4: Detail of setting the constraint 'Tie'

Constraint Type	Tie
Master surface	Pin_merge-1.Pin_ThreadSealPrimary_base
Slave surface	Pin_ThreadSealPrimary_merge-1.Pin_ThreadSealPrimary_base
Discretization method	Surface to surface

2. The contact interactions between the pin and the box including:
    a. The contact at the primary seal,
    b. The contact at the secondary seal and the contact at the shoulder.

The contact needs to be set as 'surface to surface'. The contact interactions are defined as Table 5-5 shows.

Table 5-5: Detail of setting the constraint 'surface-to-surface contact'

Interaction type	Surface-to-surface contact(Standard)
Step	initial
Sliding formulation	Finite sliding
Surface Smoothing	Automatically smooth 3D geometry surface

3. Coupling the reference points on the surface of the part 'pin' and 'box'. For easy analysis of the model, and represent the real situation of the premium connection model, reference points are created for the loads and boundary conditions to be set on instead of on the part 'pin' and 'box' directly. The coupling constraint couples the motion of the nodes on a surface to the motion of a reference node allowing boundary conditions to be applied [5]. The location of reference points are shown in the Figure 5-2 and the couplings are set as the Table 5-6

Table 5-6: Detail of setting the constraint 'coupling'

Constraint Type	Coupling
Control points	Reference point of Pin
Surface	Pin_merge-1.Pin_coupling
Coupling type	Kinematic
Constrained degrees of freedom	U1,U2,U3,UR1,UR2,UR3

There are three contact interactions between the pin and the box, primary seal, shoulder seal and the secondary seal. The threads are constrained to the pin or the box that they belong to as 'tie' and contact to the other one. The threads also contact with each other on the load and stabbing flank surfaces.

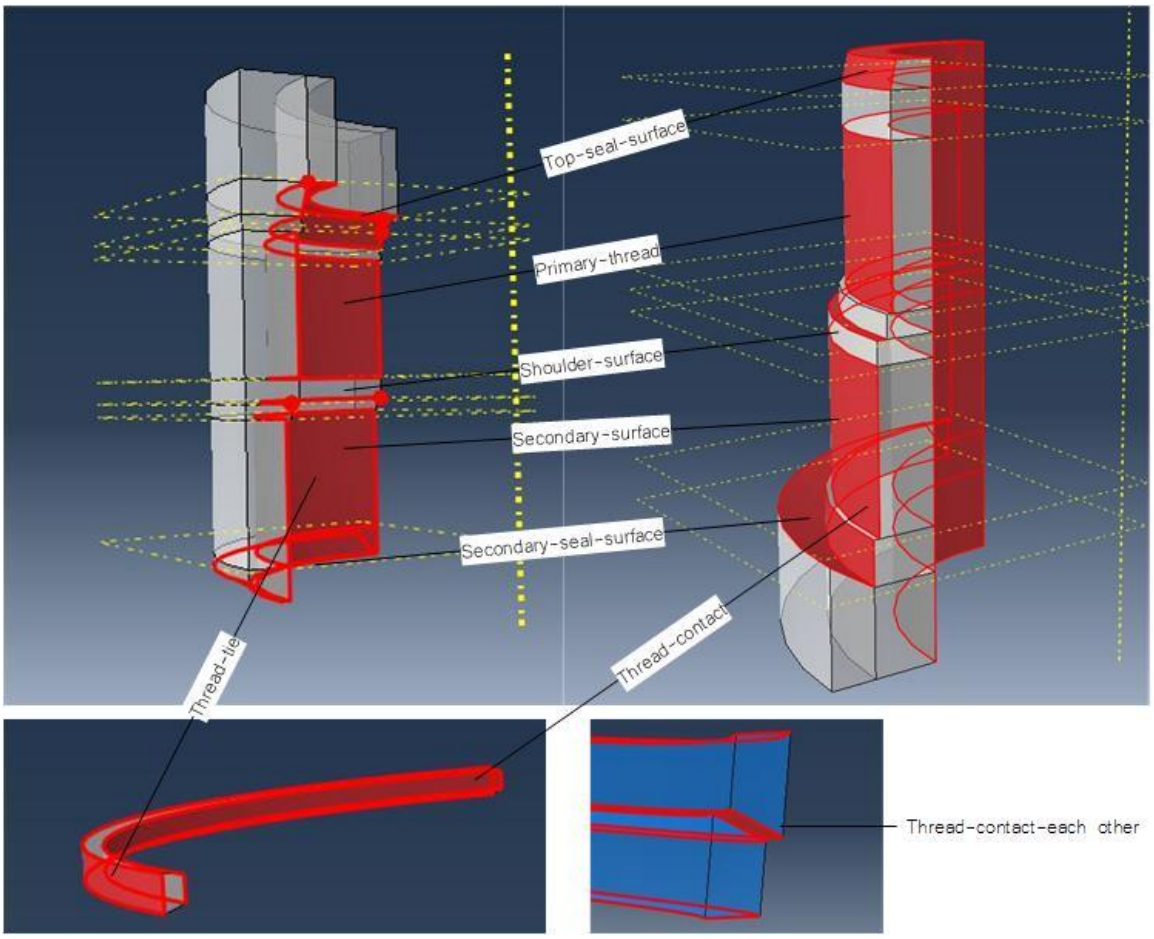

Figure 5-1: Interactions, including contacts, ties, and couplings, are set between the part ‘pin’, the part ‘box’ and the threads

5.4. Loads and boundary conditions

It is necessary to simulate the model in the actual use of force conditions to test the work of the thread. The box needs to be fixed, and then the pin is threaded (rotated) into the box. Because the threads are right-handed, the pin should rotate in the same direction. The reference points are created for the pin and the box as the point to add the boundary conditions. These points are constrained with the pin and the box as ‘coupling’, a multi-point constraint written between the displacement degree of freedom (DOF) between two nodes, so that they remain consistent in kinematics. Since the box is fixed, the boundary condition on its reference point is set as ‘encastre’, fixing all of six degree of freedom. During rotation, only the rotation and displacement of y-axis is allowed on the pin, so that the boundary conditions on the reference point of the pin is that the rotation angle of y-axis is given different specific value according to different test steps, the displacement freedom on the y-axis is released, the other 4 degrees of freedom are set to zero.

Table 5-7: the rotation boundary conditions

Rotation Boundary Conditions	
Box	Encastre
Pin(Global system)	U1=U3=0; UR1=UR3=0; UR2=4π

Table 5-8: the makeup boundary conditions

Makeup Boundary Conditions	
Box	Encastre
Pin(Global system)	U1=U3=0; UR1=UR3=0; UR2=0.5π

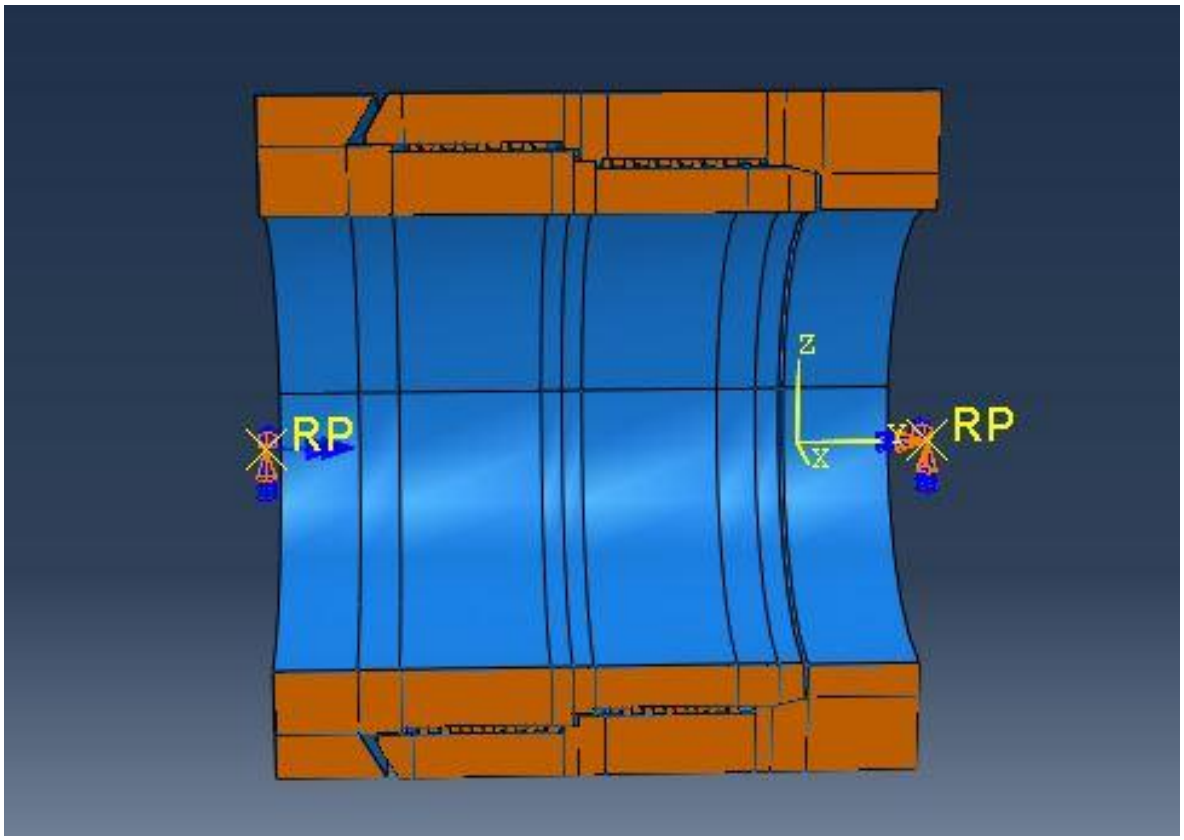

Figure 5-2: Make-Up loads and the boundary conditions, fixed displacement BCs on the box (right reference point, RP) and prescribed rotation displacement on the pin (left reference point, RP).

5.5. Meshing

After the establishment of assembly geometry for the premium connection FE model, mesh is the next very critical step. It can be said that whether the model can be properly meshed determines the correctness of the analysis results. For complex models like threaded connections, it is not possible to simply mesh all the parts together as a whole. If the number of elements is too few, the results will be inaccurate and unable to properly represent the resulting displacement as well as the stress and strain field, particularly in areas of rapid change. If the elements are too numerous, the required solution time is too long, affecting the work efficiency. The overall mesh partitioning scheme discussed in section 4.2. is key to an effective mesh on the connection and therefore a representative solution for the displacement, strains and stresses in the connection. The partitions for the parts were completed in the plug-in to enable an effective meshing scheme for the user. The partitions were created based on model features which are changed by the data get from the interface of the plug-in. This means the partitions are reliable.

It is inferred that, in the actual connection, there are three sealing areas between the pin and the box are where a fine mesh is needed, while the other areas require only a coarse mesh. Figure 5-3 shows that all the contact areas are high mesh density regions. Transition regions are required to connect the fine meshed regions with the coarse mesh regions. Partitions of the pin and the box created by the plug-in divide the regions along connection features. The primary seal and secondary seal regions have one transition regions.

The element selection associated with the mesh has a significant influence on the accuracy of results and the run time of the model. Because of the model in this project is only test of its elastic-plastic properties, the linear 3D brick elements (Abaqus C3D8 family of elements) are selected. There are two kind of element type for the pin and the box, the linear 3D, reduced integration elements (C3D8R) and the linear 3D full integration elements (C3D8). The first one only use 1 integration point and the second one use 8 integration points. It means using the C3D8R can reduce the model runtime. However, using the C3D8 can get more accurate results. Since the fillet radius of the threads has already been removed, the thread mesh is only one element of the section, the Abaqus C3D8I elements are used for all thread components. The elements selected for the different regions of the connection model were chosen for the mechanics in that particular region of the model. The user of the premium connection plug-in is able to select the elements types for the meshed regions depending on needs of the of FE model. All of the regions in the part ‘pin’ and ‘box’ are meshed with the same element type and element selection for the different regions of the premium connection model are displayed in Table 5-9.

Table 5-9: Abaqus element types used in this model

Part	Element Type
Pin	C3D8
Pin Primary Thread	C3D8I
Pin Secondary Thread	C3D8I
Box	C3D8
Box Primary Thread	C3D8I
Box Secondary Thread	C3D8I

The geometric model created by the premium connection plug-in is a partitioned geometry model, where the mesh of the model can be changed to accommodate an effective meshing scheme. The mesh model can't be modified. If modification is needed, the model should be re-create. It's a waste of time to make the user do a lot of duplication work. So that the plug-in does not contain the operation of the mesh. If Mesh refinement operations need to be applied to the premium-connection plug-in model to adapt and/or refine existing meshes, to support mesh convergence studies, the user can mesh the model according to the solution response and the needs of the analysis. The Figure 5-3 and Figure 5-4 shows the finial meshed model in this research. The testing result in section 5.6 shows that it is a good mesh for testing. It is significant that this geometric can be modeled complexity with bricks

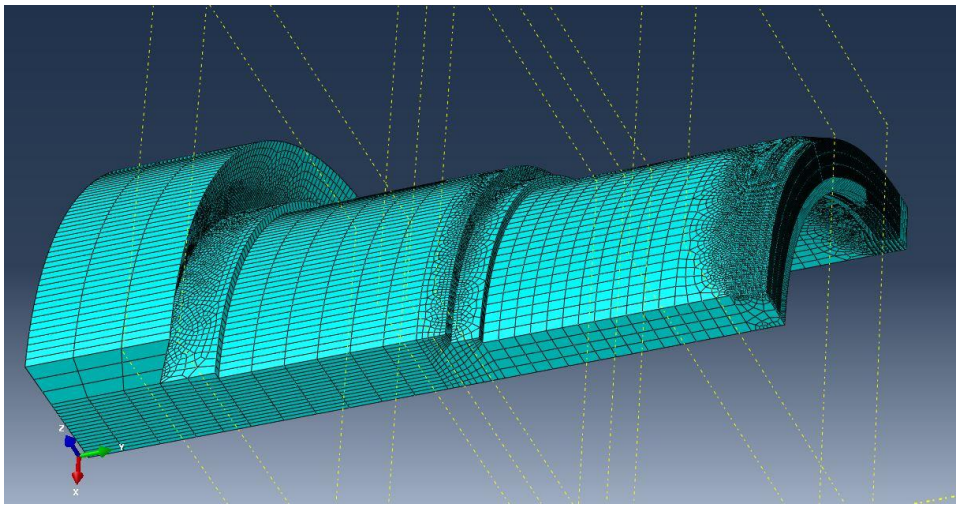

Figure 5-3: The full meshed part 'pin'.

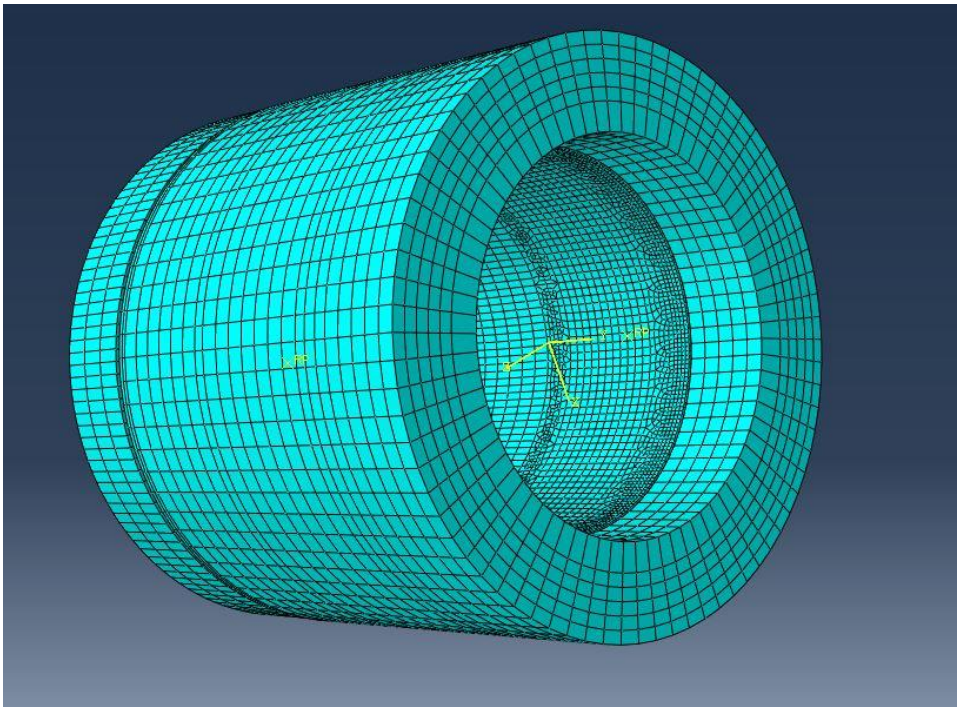

Figure 5-4: The full meshed premium connection model.

5.6. Test Cases

The most important thing in the project is to make sure that the Abaqus plug-in generated model works well and capable of enabling a user to generate results for the virtual test that are reasonable. In order to verify the reliability of the model created through the plug-in, the entire test session is divided into two parts. For the first part of the test session, after the pin is rotated by a specific angle, if the pin's displacement in the y-axis changes positively correlated with the pitch of the thread to test whether the thread is reliable. In the second part of the test session the pin is rotated alone the thread from the assembly position. This test explores the stiffness of the connection by analyzing the result of the entire connection.

5.6.1. The rotation testing

The first part of the model test is to test the reliability of the thread. The thread is a right-handed thread. When the box part is held stationary, and the pin part is rotated to the right about the y-axis, the two parts should gradually approach each other by the pitch of the thread until the surfaces of the primary seal between the pin and the box contact each other. The threading of the pin into the box preloads the primary seal to produce a contact pressure and an elastic stress in the pin "nose". The contact pressure developed over a length of the of the pin and box makeup a metal –to-metal seal, responsible for sealing the premium connection under load. This process can be decomposed into two parts, the first part is the rotation of the pin into the box.

The first part of the evaluation process focuses on if the thread in this model works well. The relationship between the displacement of the pin on the y-axis after rotation and the pitch of the thread is the main point. Because there is a gap between the threads engaged, as the Figure 5-5 shows, the displacement is not just the product of the rotated angle and the pitch. But as long as the displacement and the pitch are nearly in a positive correlation, difference between the product and the displacements is in the error range which is decided by the gap, as the equation (5-1) shows, the threads in this model is reliable.

In the first part of the test, in order to see the experimental results clearly, the pin and its own thread together as a whole move down two-pitch distances in the assembly module. The rotation of the pin is set to a right-hand rotation through an angular displacement of 720°, imposed on the pin as a prescribed rotation displacement boundary condition.

Some variables are defined for calculation. The length of the thread crest is defined as 'l_1', the thread groove width is 'l_2', and the pitch of thread is set as 'p'. The gap of the thread engagement is '$(l_2 - l_1)$'. The final displacement of the pin in the y-axis is considered acceptable in the range:

$$(\mathrm{p} \times 2 - (l_2 - l_1), \mathrm{p} \times 2 + (l_2 - l_1)) \qquad (5\text{-}1)$$

The exact number depends on the relative position of the threads at the start of the rotation. If the pin thread engages the box thread in the positive direction of the y-axis when the rotation starts, the final displacement is relatively short. If in negative direction of the y-axis, the result should be relatively long. The variables are show on the Figure 5-5.

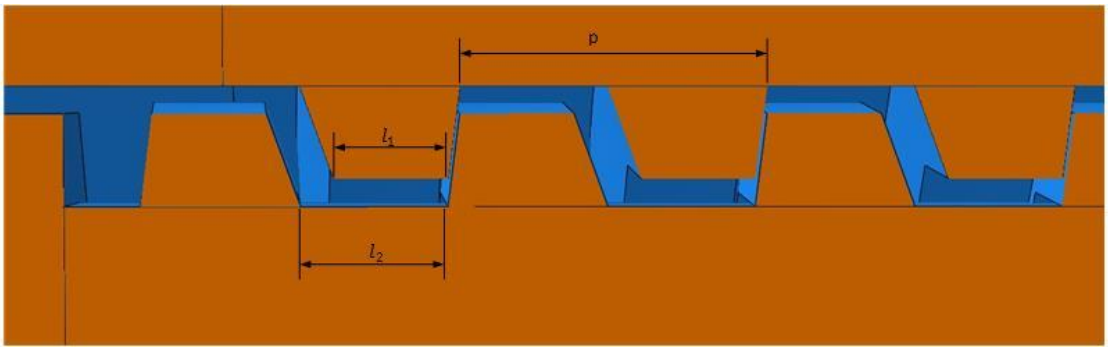

Figure 5-5: The variable that used to calculate the distance after the rotation of the model.

As the Figure 5-6 displayed, Pin in the right-hand rotation along the y-axis is gradually increased. In this model, the value of the variables as show in the Table 5-10.

Table 5-10: The actual number that set to the variables

variables	Value(in)
l_1	0.06202
l_2	0.799
p	0.1667

As the Figure 5-6 shows, the displacement variation in the Y-axis direction is 0.34in, within the result range (0.3155in, 0.3512in), as the equation (5-2) shows.

$$(0.1667 \times 2 - (0.799 - 0.06202), 0.1667 \times 2 + (0.799 - 0.06202)) \qquad (5\text{-}2)$$

The result in the Figure 5-6 shows that the pin is in the rising state on the axial displacement through the right-hand rotation. The threaded distance is positively related to the product of the angle of rotation and the pitch. Fig. 5.6 shows that the premium connection model properly threads the pin into the box under a nonlinear general contact condition. Ideally the lead vs. angle would be a line of constant slope equal to the pitch. The 'bumps' in the plot is because the gap existed between the threads, as the Figure 5-5: The variable that used to calculate the distance after the rotation of the model.Figure 5-5 shows. Both of two flank surfaces on the thread tooth are possible to contact with other tooth which makes the 'bumps'. These 'bumps' in the plot don't influence the testing result.

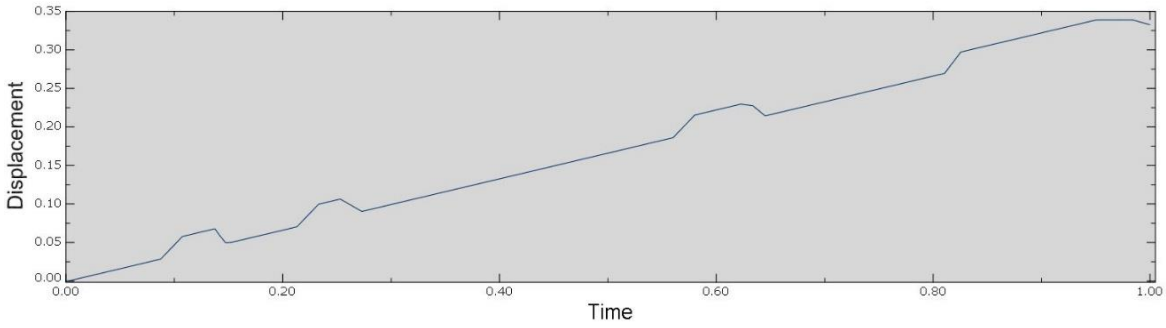

Figure 5-6: The changing curve of the displacement on the y-axis when the model rotated 720°.

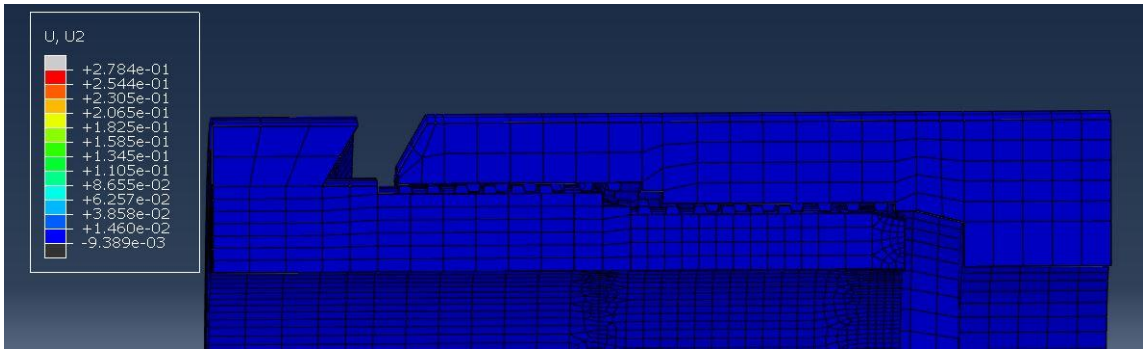

Figure 5-7: The starting location in the connection model assembly when the rotation starts.

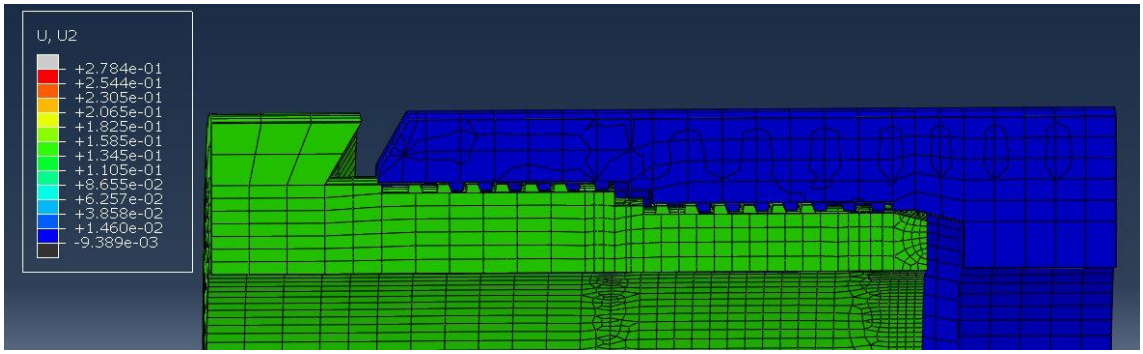

Figure 5-8: The location of the connection model when the pin rotated 360°.

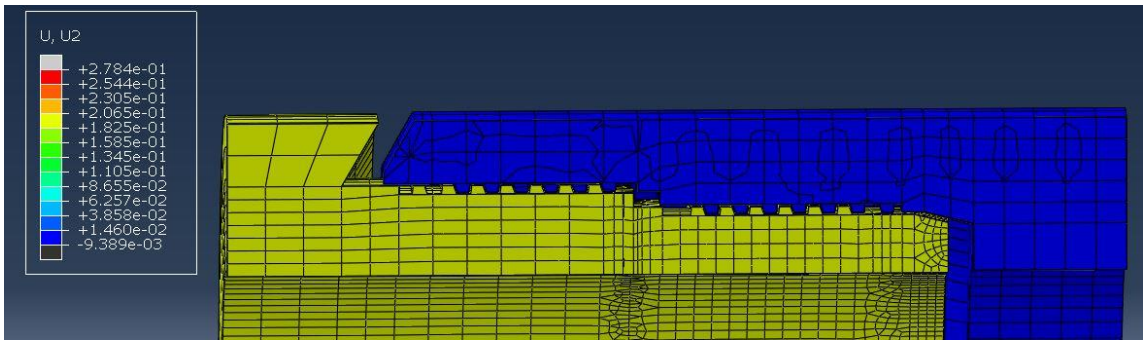

Figure 5-9: The location of the connection model when the pin rotated 540°.

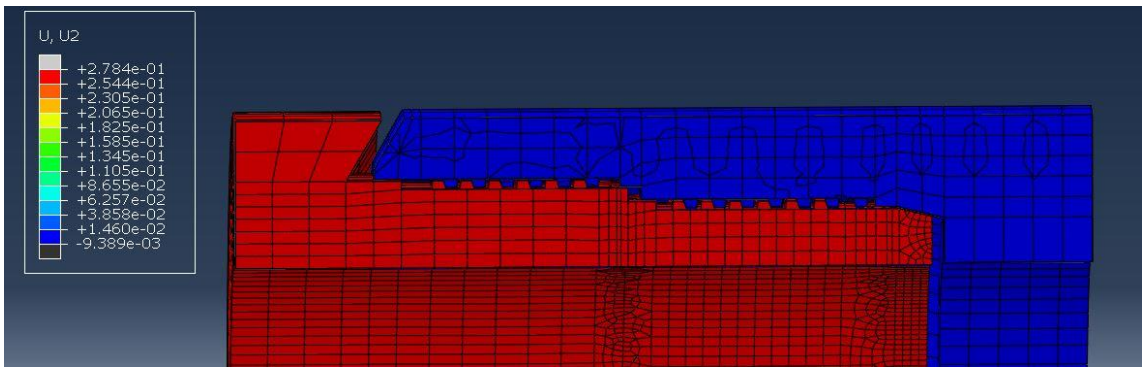

Figure 5-10: The final location of the connection model when the 720° rotation finished.

5.6.2. Sealability testing

The second part of the evaluations focused on the force state after the pin and box are gradually tightened. When the connection reaches the hand tight state, if the rotation continues, there should be a significant stress change in the contact area, including the sealing area and the contact surface of the screw. The pin should also require a significant torque to further preload the connection. Because no physical test data is available to be used for comparison on this particular premium connection, only through the results of data changes in FEA chart to observe the trend of change. The importance of the chart is the emergence of a major turning point.

The moment that the pin and the box are in full contact there are two conditions that must be established; (1) the connection make-up conditions, (2) the preload state for the metal-to-metal seal. The purpose of this phase of evaluating the premium connection finite element model is to predict the pin and box makeup state. This test is about developing significant torque and further "seating" the pin nose into the box. This results in bending, axial and shear deformations which result in plastic deformation and large strains. Once the pin nose reaches the base of the box, there is no place left for it to advance upon continuing to rotate the pin. Further rotation of the pin results in very large torques and displacement which stretch the connection. Since the pin and the box are elastic members further rotation tries to move the pin forward, but it has nowhere to go so it starts stretching the pin and box along the threads and deforming the pin and box in the primary and secondary seal regions. The test end of with regions of the model that see stresses in the plastic region, particularly in the seal regions. This is where preloading the connection comes from. When applying external loads, operational loads, internal pressure, bending, thermal loads, the connection responds based on the preload state of the connection. This is why it is so important to get the make-up conditions right. In this section, a gradual increase in the angle of rotation, as the Table 5-11 shows, is used to analyze the variation of the distance, stress and torque in the connection.

Table 5-11: The pin's rotation degree in the sealability testing.

Test steps	Pin's rotation degree
The first test	45°
The second test	90°
The third test	135°

First, the pin is rotated at 45 ° from the initial position. The result shows that the rate of displacement of the pin in the y-axis direction is the same as that of the previous section, that the pin and the box are not sealed.

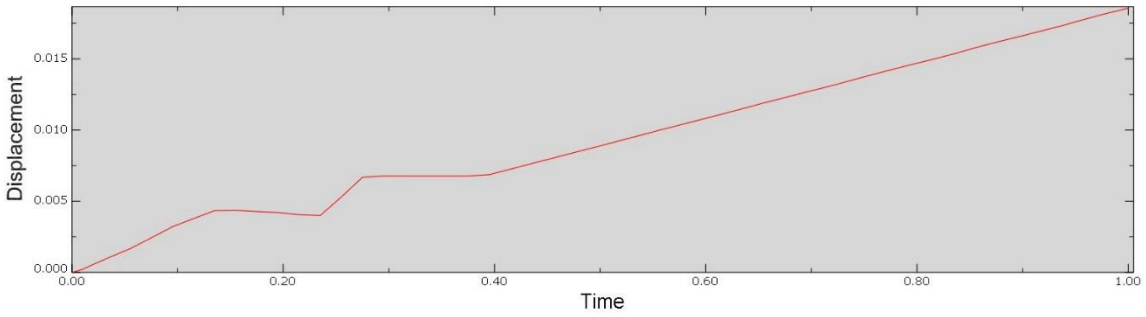

Figure 5-11: The changing curve of the displacement on the y-axis on the sealability testing when the pin rotate 45°.

In the second test, the pin is rotated at 90 °, the results remain unchanged and the connection has not yet reached the makeup state.

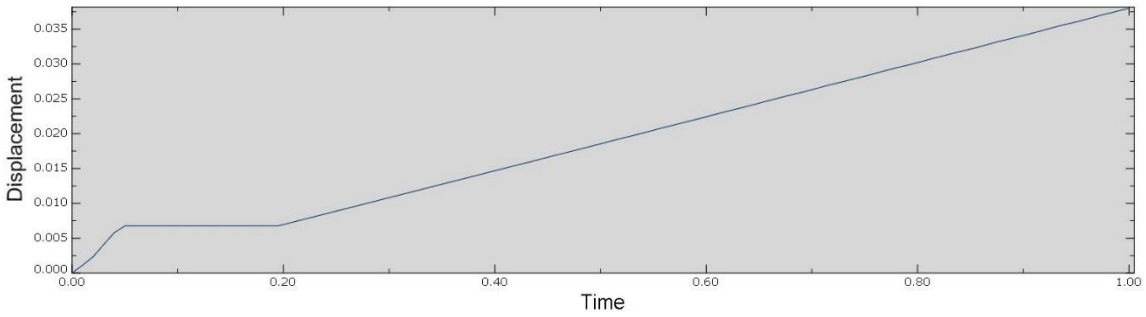

Figure 5-12: The changing curve of the displacement on the y-axis on the sealability testing when the pin rotate 90°.

In the third test, the pin is rotated at 135 ° and the results shown on the Figure 5-14. To reach about 68.5% of the whole testing process, the rate of displacement of the pin in the y-axis suddenly slowed. The stress at the top surface of the pin that in contact with the box and the torque of the pin are suddenly increased at the same moment. It can be considered that the pin and box are make-up from this moment. The thread begins to be damaged when the pin continues to rotate after this moment.

$$68.5\% \times 135° = 92.475° > 90° \quad (5\text{-}3)$$

Equation (5-3) also verified that the previous 90 ° rotation did not achieve the purpose. The end results shows on the Figure 5-13, Figure 5-14 and Figure 5-15, which proves that this model is reliable in performing seal contact analysis.

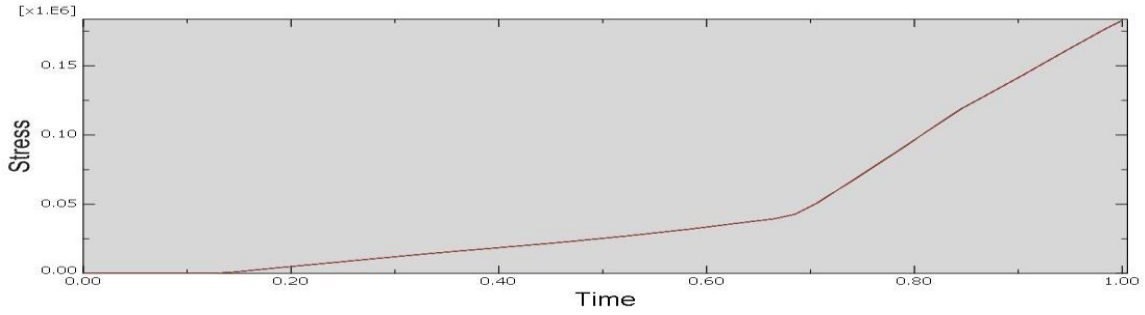

Figure 5-13: The changing curve of the stress on the primary sealed area of the part 'pin' on the sealability testing.

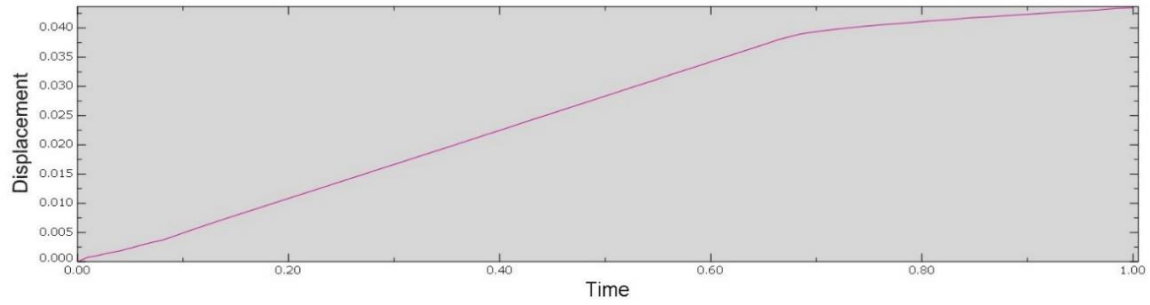

Figure 5-14: The changing curve of the displacement on the y-axis on the sealability testing.

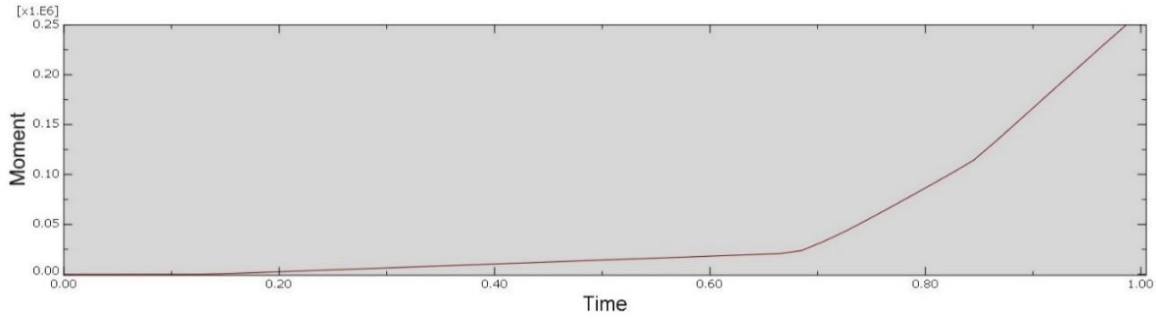

Figure 5-15: The changing curve of the moment on the part 'pin' on the sealability testing.

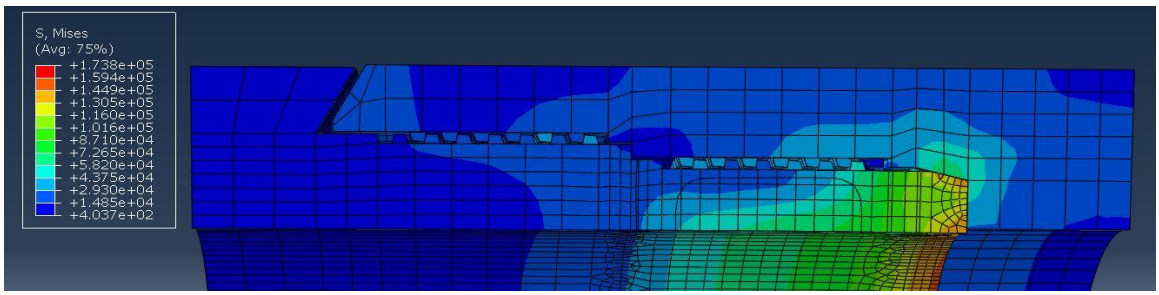

Figure 5-16: The result of the sealability testing.

5.7. Application summary

The premium connection model created by running the Abaqus Plug-in generates a complete geometric model, the geometry of the model does not need to be adjusted in the application. To complete a full finite element analysis of the premium connection, the following operations need to be completed:

1. analysis steps specific to the type of analysis being performed need to be defined,
2. loads and the boundary conditions specific to the analysis being performed need to be set,
3. The premium connection finite element model needs to be meshed and converged.

In this chapter, a detailed description of the Abaqus premium connection plug-in is described for a connection make-up analysis. The parameters defined for the model are presented in tables 5.1 through 5.10. The user of the premium connection plug-in can reference changes to the model based on the needs of the required analysis.

Two steps are used in the analysis to set the make-up conditions for threading the pin into the box. The complete rotation of the pin into the box is completed over two steps. The reason for the multi-step analysis is to decompose the threading process into simpler operations making the data easier to handle in the analysis. Each Abaqus analysis step corresponds to a single analytical purpose. The output state of the first step is fed into the second step as initial conditions.

When setting up the interactions, the surfaces of each part that are created through running the plug-in are necessary in setting up the contact interactions used in the finite element analysis, eliminating the need for the users to create them by hand. Each surface has its own name, which is defined by the plug-in and can be referred to uniquely. The users without much experience can also be easily to set. The difficulty of using plug-in is reduced.

Reference points, sets and surfaces associated with the pin and the box are created by the plug-in and used to define loads and boundary conditions.

The plug-in also defines partitions created on the part ‘pin’ and the ‘box’ to enhance meshing. However, meshing the parts is done entirely by the users. Each user is required to mesh the model according to the needs of the analysis. Partitioning the mesh targets develops many smaller cells; the users can define the regions that can benefit from a finer or coarser mesh. Some of the regions are effectively used as the mesh transition regions.

The analysis of the experiment is divided into two parts. In the first step, the Pin is rotated clockwise by 720 ° along the y-axis, with the result that the pin is displaced in the positive direction of the y-axis, indicating that the direction of rotation is the same as the helical direction of the thread. The displacement is positively correlated with the pitch, within the error range, indicating that the threads between the pins and the boxes are in contact with each other. The threads work well and the model generate the desired response. The model along the spiral direction shows that the creation of the thread is reliable.

The second step of the test is starting from the pin and the box’s design position, since the thread has been proven to be reliable, when the pin is rotated by the direction of the thread, the contacts

between the pin and the box are closed properly. By observing the test results, the stress in the primary sealed area of the pin and the box suddenly becomes larger when the pin is rotated beyond the hand-tight position, the displacement speed of the pin in the y-axis is abruptly reduced, and the moment on the pin increases suddenly. The results indicate that the pin and box are in a sealed state. All of these changes of the data testing shows that the reliability of the seal connection test. Both of these tests demonstrate that this model is a reliable model that can reflect the working state of the real threaded connection, and demonstrates the feasibility of using the plug-in instead of modeling by hand every time.

Chapter 6. Summary and conclusion

6.1. Summary

The process of demonstrating the applicability of a premium connection for use in an off-shore drilling operation requires a great deal of development effort including testing time, manpower and material resources. Premium connection manufactures are committed to designing and studying improvements as well as verification and validation of premium connection designs. International standard organizations and government regulators are looking to further develop elements of the testing process for premium connections.

Finite element analysis is widely used on practice to predict the performance and sealing of oil & gas connections. The power of the Finite Element Method for handling complicated geometries, loading and boundary/initial conditions yielding reliable results prove that FEA is an appropriate method to reduce the number of physical tests on the premium connection. The accuracy of the result for FEA depends on many modeling decisions requiring significant expertise if the FE model is to adequately represent the behavior of the connection under the ISO 13679 test protocol or in-service loads. The completion of a representative finite element model of a premium connection design requires modifications to the model. Several parts of the model, even the entire model, needs to be rebuilt or meshed many times, when the connection is being studied for the effects of tolerances and worst case design states. Instead of creating the model by hand every time, using a scripting or plug-in approach to building the model becomes an effective method to support advanced model studies.

A script can be developed which contains nearly all the steps to create a FE model. However, the more modeling functions the script embodies, the more complex the code becomes. The complexity can reduce the script readability, requiring the designer to spend more time studying the script before using it. Making the model building script understandable and easy to edit is key to further enhance the work efficiency and extended use of the FE model. Creating a plug-in with illustrations and auxiliary text can help the designer quickly understand and interact with the model parameters and become an enabling “best practice” for model refinement, design trade-off studies, model verification and validation leading to design optimization.

An accurate understanding of the complete FEA process is necessary for project optimization. The model needs to be created with the purpose of being an “operational” model from the start, the development of functions within the plug-in that support operations in each module should be done to support this goal.

Methods developed in this research can create components in the part module of Abaqus/CAE to explore and develop a “best practice” to creating sketches of the pin, the box and each of the thread forms for the purpose of developing a parametric operational finite element model. The parametric premium connection components are created as separate parts by developing parametric part sketches. The modules controlling the sketches are independent from the main function of the premium connection plug-in. The user can edit these part sketches or functions to change the shape of the model or reuse them in a plug-in for another model containing the same component. Errors can be developed during the analysis if the full 360° 3D model was created by revolving the part section as one complete 360° part in Abaqus/CAE. This appears to be a problem

with the Abaqus/CAE scripting API, the merging of 90° component to develop a full 3D model is a workaround. The partitions and the surfaces that are set in the 90° sector part are also inherited after being merged. The premium connection FE model builders are able to select the mesh technique and element shapes themselves. They can also define the fine and the coarse meshing regions independently. The surfaces helps setting the interactions. The code that partitions the parts are explained in section 4.1.3.

The operations required to position the thread in the connection assembly are adjusted by the pullout length for the thread on the pin and the box. The assembly operation is significant and difficult to program a-priori. There are two ways to create thread forms. One method studied was to revolve the section of the thread once, setting the angle as 2π times the turns of the thread. The other way is to create a 90° sector as the base unit of the thread using the radial and linear pattern operations to build the thread followed by merging the 90° sectors together. The reason for choosing the second way to create the thread as the "best practice" is explained in section 4.2.2. The thread module has its limitations. The turns of the thread can only be adjusted in a range (5 circles to 7 circles). The FE model builder is able to change the range for each design situation. The 'merge thread' function is relatively easy to program due to its independence of other code for the premium connection plug-in.

The plug-in is built by the Abaqus/CAE RSG (Really Simple GUI) dialog builder. The functions of the plug-in are well explained in the illustration and the explanatory text on the interface which can help the user of the plug-in to quickly understand and start work. The details of the interface, are shown in section 4.3.1. Since the kernel of the plug-in can only execute one function each time, the premium connection plug-in program is divided into several independent Python modules, which executes one step in the modeling process. The main program and all other functions can call as many functions as needed. The main program contains only the commands of calling out the functions in each module and executes them in order. All the modules work independently and are uncoupled from each other. This makes the code easy to change or extended other cases.

Use of the Abaqus premium connection plug-in requires some modeling steps to complete the finite element model after the geometric model is created. The contact interactions in the model, the loads and boundary conditions of the specific analysis to be performed need to be set. These tasks are not completed by the plug-in because the analysis requirements specific to a particular study could be different in each design, even for the same model. Meshing is another important task for the user to complete. Choosing the type of the elements and the size for seeding the edges vary depending on the type or stage of the analysis to be performed. The design of the plug-in has been developed to allow for different stages of premium connection model development or analysis task.

The premium connection finite element model created by the plug-in needs to be verified before it can be used to predict the response of a practical connection. Premium connection thread forms are typically designed as right-handed threads, the pin should translate according to the right-hand rule when it is rotated. The strain and stress produced in the primary and secondary seal contact zones of the pin and the box are used to compute seal metrics, quantities that are used to determine if the connection is sealed or not. The result of the thread testing and sealability testing in section 5.4 demonstrated that this model works well. This plug-in is effective and efficient.

6.2. Conclusion

All the operations that the users executed in Abaqus/CAE, are journaled. Running tailored scripts can achieve almost all the work carried out by a user in Abaqus/CAE. For an experienced engineer, the script can be used to complete most of the work, without requiring the Abaqus/CAE interface being used. Engineers can also modify the completed script to develop the plug-in for similar models. This is very helpful in improving work-flow efficiency and is one of the most significant features of running the script to complete the modeling and analysis task. An additional benefit is that the scripts/plug-in embody the finite element knowledge and know-how of an experienced premium connection modeler. Less experienced users can use the premium connection plug-in models incorporating the experience of more analysts, producing more standardized models. Experienced engineers with premium connection, finite element modeling and programming experience can modify and develop additional models for other connections leveraging the knowledge, experience and effort embodied in the plug-in. Modifying the script to support extended analysis with finite element models required for a specific task is difficult for an inexperienced finite element user. From this point of view, scripting finite element models is difficult for users to learn and modify. The method that running the script to create FE models can't be extended to a wide range of projects.

The premium connection model developed in this research uses the Abaqus/CAE's plug-in architecture to produce a script to develop a parametric, operational finite element model. The plug-in is developed to execute the script rather than the user executing the script file directly. The plug-in is characterized by a user interface that can achieve human-computer communication and codifies "best practice" modeling operations. The premium connection Abaqus plug-in developed in this research completes most of the modeling process. The premium connection plug-in developed in this research can be used to automatically generate a large number of the same type of model or the similar model for experiments, saving a lot of work time.

Compared to using scripts, the plug-in interface has graphical and descriptive text. The plug-in interface embodies a "best practice" modeling approach to and illustrates the model parameters, relieving the user from the need to master the modeling technology so he/she can focus on the analysis task and use the model directly. The engineers using the premium connection plug-in are not required to have a wealth of finite element experience, as long as they have a basic understanding of premium connections, the model and analysis requirements.

Each specific plug-in is targeted to a particular type of premium connection model but can be used for a family of premium connections of that type. Users are able to modify the plug- to complete their own specific functions to accomplish a specific modeling task. The Abaqus plug-in kernel and the modules responsible for each function divide a complex and lengthy script file into separate parts compared with running the script files directly. The decomposition of the premium connection models increases the readability of the program, and enables the users who have a programming foundation an easier path to modifying the code. Through study of the kernel program, users can expand the functionality of the plug-in. Development of an auxiliary mesh function can provide several sets of mesh solutions for a certain type of model for users to choose, of course, users can also choose to complete their own meshing operations. The plug-in is an extension to script-based modeling, reducing the difficulty of use. The models created by the plug-in are consistently developed, embody "best practices" and work as well as those created by hand step by step.

Even though all the operations in Abaqus/CAE can be implemented by executing the command code in an executing script or plug-in for a particular the target, many operations are more efficient by the user's interactive operation, such as picking a specific point, a line, a face, or a cell. The code used to determine the specific target is about calculating the coordinate of the picked points. It is much simpler for the users to pick them by mouse. These kind of modeling operations are easier to do interactively rather than scripting.

6.3. Recommendations

The plug-in of for the premium connection model is still in the early stages of development and shows promising results. Several features of the plug-in require additional exploration to create a FE model instead of the geometric model by running the plug-in. The operations of turning the geometric model into a FE model in chapter five can also be added to the plug-in as independent modules.

The ID numbers are used to define the surfaces in this plug-in. Because of the objects' ID numbers may be different when the version of Abaqus changes, using the '*findat ()*' function to define objects is more strict than using their ID numbers. All the objects, include lines, faces, and cells, need to be defined by using the *'findat ()*' function in the future work.

More surfaces can be set in the plug-in, not only for interactions, but also for loads. This research's tests emphasized testing the thread and seal contact and finding the hand tight location. The premium connection plug-in is not only developed for these tests. For example, the sealability of the premium connection needs to be ensured when the tubes working with the gas and oil. The surfaces inside the pin and the box are used for add the pressure for testing the sealability. These surfaces should be defined in the plug-in.

The plug-in can be developed to add the interactions in the model. The designer knows the interaction details. The surfaces and reference points are also defined by the plug-in. It is possible to create a module to set the interactions, including contacts and constrains, after the model created.

The designer can mesh the model as an example and add it into the plug-in. Users can test the model directly based on this mesh. As the model created by the plug-in is a geometric model, the mesh in the finite element model can also be changed later if the user has experience and wants to mesh the model himself.

Future development of the plug-in can focus on incorporating many functions developed into separate plug-ins. Each of functional modules in this plug-in can be developed to an independent plug-in with more details and operational choices. A main plug-in can also be developed to control all of the specific purpose plug-ins. Compared to the approach developed in this research; the new method will give users more options during use. For example, a plug-in can be developed only for creating the partitions on the part 'pin'. The user can run this only for changing the partitions on the model, instead of starting the work from creating a new part. Creating a partition plug-in can offer the users more options for creating the partitions in their way. Inexperienced users will be more convenient to use if the plug-in can be developed this way.

The plug-in can also be extended to add a post-processing module. After doing the test through the FE model, the user can analyze the test results based on their selections in the interface of the plug-in to generate data plots and reports.

Bibliography

[1] Vespa M, 2015, "Introduction to Oil Country Tubular Goods(OCTG)," EdX.Web. March, 2015.
[2] API Technical Report 17TR8, 2015, "High-pressure High-temperature Design Guidelines."
[3] Bradley, A., Nagasaku, S., and Verger, E., "Premium connection design, testing, and installation for HPHT sour wells," Proc. SPE High Pressure/High Temperature Sour Well Design Applied Technology Workshop, Society of Petroleum Engineers.
[4] ISO13679, 2002, "Petroleum and natural gas industries-Procedures for testing casing and tubing connections "Switzerland.
[5] Ahsan, N., 2016, "OCTG Premium Threaded Connection 3D Parametric Finite Element Model," Virginia Tech.
[6] Hilbert Jr, L., and Kalil, I., "Evaluation of premium threaded connections using finite-element analysis and full-scale testing," Proc. SPE/IADC Drilling Conference, Society of Petroleum Engineers.
[7] Assanelli, A., and Dvorkin, E., 1993, "Finite element models of OCTG threaded connections," Computers & Structures, 47(4-5), pp. 725-734.
[8] Balmès, E., 1996, "Parametric families of reduced finite element models. Theory and applications," Mechanical systems and signal Processing, 10(4), pp. 381-394.
[9] Fan, J., Yang, Z., Wang, J., Ding, S., Liu, C., and Shi, Z., "Parametric finite element modelling and nonlinear analysis of vehicle brake," Proc. Mechatronics and Automation, 2009. ICMA 2009. International Conference on, IEEE, pp. 1762-1766.
[10] Jiapeng, T., Ping, X., Baoyuan, Z., and Bifu, H., 2013, "A finite element parametric modeling technique of aircraft wing structures," Chinese Journal of Aeronautics, 26(5), pp. 1202-1210.
[11] Luo, L., and Zhao, M., 2011, "The Secondary Development of ABAQUS by using Python and the Application of the Advanced GA," Physics Procedia, 22, pp. 68-73.
[12] Jones, M., Price, M., and Butlin, G., "Geometry management support for auto-meshing," Proc. Proceedings of 4th International Meshing Roundtable, Citeseer, pp. 153-164.
[13] Haghighi, K., and Kang, E., 1995, "A knowledge-based approach to the adaptive finite element analysis," Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations, Springer, pp. 267-276.
[14] Robertson, T., Prasad, B., and Duggirala, R., "A knowledge based engineering method to integrate metal forming process design and simulation," Proc. Proceedings of the 1994 ASME Database Symposium, Engineering data Management: Integrating the Engineering Enterprise, ASME, p. 41ą50.
[15] Santi, N., Carcagno, G., and Toscano, R., "Premium and Semi-Premium Connections Design Optimization for Varied Drilling-with-Casing Applications," Proc. Offshore Technology Conference, Offshore Technology Conference.
[16] Galle, T., De Waele, W., Van Wittenberghe, J., and De Baets, P., "Optimal make-up torque for trapezoidal threaded connections subjected to combined axial tension and internal pressure loading," Proc. ASME 2014 Pressure Vessels and Piping Conference, American Society of Mechanical Engineers, pp. V002T002A022-V002T002A022.
[17] Ostergaard, E. B., 2013, "A Refined Methodology for Calibrating Premium Connection Make-ups," Virginia Tech.
[18] WANG, J.-l., and LI, P., 2009, "Secondary Development for GUI of Box Girder Bridge Based on ABAQUS [J]," Journal of Chongqing Jiaotong University (Natural Science), 6, p. 006.

[19] Nesládek, M., and Španiel, M., 2017, "An Abaqus plugin for fatigue predictions," Advances in Engineering Software, 103, pp. 1-11.
[20] Ure, J. M., Chen, H., and Tipping, D., "Development and implementation of the ABAQUS subroutines and plug-in for routine structural integrity assessment using the Linear Matching Method," Proc. SIMULIA Community Conference 2012,(Formerly the ABAQUS Users Conference).
[21] Abaqus, V., 2014, "6.14 Documentation," Dassault Systemes Simulia Corporation.
[22] CHENG, L., and LI, H., 2009, "Second development of Abaqus based on the scripting interface," Modern Machinery, 2, pp. 58-65.
[23] Zelle, J. M., 2004, Python programming: an introduction to computer science, Franklin, Beedle & Associates, Inc.
[24] Ullman, D. G., 2002, The mechanical design process, McGraw-Hill Science/Engineering/Math.