Master of Science Thesis

# Simulation of Wear in Turbocharger Wastegates

Hanjiu Lin


Faculty of Aerospace Engineering · Delft University of Technology

TU Delft Delft University of Technology

MSCCONFIDENTIAL

# Simulation of Wear in Turbocharger Wastegates

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

Hanjiu Lin

11.01.2016

Faculty of Aerospace Engineering · Delft University of Technology

The work in this thesis was supported by BMW Group Munich. Their cooperation is gratefully acknowledged.


Delft University of Technology
Faculty of Aerospace Engineering
Department of Aerospace Structures and Materials

**GRADUATION COMMITTEE**

Dated: 11.01.2016

Chair holder:

dr. Sergio Turteltaub

Committee members:

dr. techn. Michael Wibmer

dr. Jos Sinke

dr. Mostafa Abdalla

# Preface

This thesis report is the result of my work at BMW's simulation department for the Development of Turbocharging and Exhaust Systems from March to November 2015. I applied to work on this project the previous year and was very fortunate to be accepted into a team of exceptional colleagues and collaborators.

I would like to express my sincere gratitude to my supervisors Dr. Michael Wibmer and Dr. Timo Schmidt for their extraordinary support and guidance throughout my thesis work. They were extremely generous with their time despite many other commitments, and I could always rely on their effective input and technical expertise. This project could not have been successful without their dedication, good judgement and encouragement in stressful times.

In addition, I would like to thank Mr. Oliver Grabherr for admitting me into the company, and Mr. Adrian Schmidt for assisting me with the compilation of my report. I wish Mr. Schmidt best of luck for the continuation of this project and hope that my work will provide some useful input for his Ph.D. dissertation.

Of course I am also indebted to Prof. Sergio Turteltaub from the TU Delft for his advice and supervision, and to all members of the graduation committee for taking an interest in my work.

Last but not least, I have to thank my X-tremely High Sisters for putting up with me for more than five years. Their unfounded belief in my abilities is the primary reason why I managed to graduate at all. I shall always be grateful for our time at TUD.

Delft, Hanjiu Lin

11.01.2016

# Summary

This thesis report presents the detailed functions of a comprehensive Finite Element - based methodology to simulate sliding wear in dynamic systems with non-uniform contact conditions, such as the wastegate system of an automotive turbocharger.
Though sliding wear occurs due to many different microscopic mechanisms, the simulation aims to depict the resulting macroscopic changes in geometry rather than individual modes of material separation. The geometric modification is generated using Abaqus' integrated *Arbitrary Lagrangian-Eulerian* (ALE) adaptive meshing technique, where the location of nodes in the entire mesh is shifted independently from the underlying material. For that purpose, a user-defined subroutine UMESHMOTION was written to prescribe the the magnitude and direction of node displacement at nodes on a part's contact surface according to Archard's equation for sliding wear. The implemented algorithm for finding the proper node displacement direction is valid for any open surface of a three-dimensional model with arbitrary spatial orientation.
In contrast to most state-of-the-art wear simulation techniques, this study has produced a wear simulation routine that is applicable to dynamic systems, i.e. models that cannot be solved with a static or quasi-static analysis. The difficulty lies in adapting Abaqus' integrated ALE adaptive meshing technique for dynamic analyses. In order to apply this technique (and the established UMESHMOTION subroutine) the results of the initial dynamic analysis have to be passed on to an intermediate static analysis where geometric modifications are possible. This data transfer is accomplished by calculating the wear displacement at each node in a post-processing step after each dynamic cycle, such that UMESHMOTION is able to read in the values in the subsequent static cycle. The modified geometry generated in the static analysis is carried over to the next dynamic cycle by rewriting the node coordinates in the initial input file. The devised "wear simulation loop" can be performed any number of cycles, or can be carried out until a certain total sliding distance has been achieved. It was found that the method is indeed able to qualitatively produce the expected wear profile found in the critical lever/bushing interface of the wastegate device.
The routine also includes a simple extrapolation method to enhance the wear-induced geometric changes produced in each simulation cycle. Since wear damage commonly evolves over an extended period of time - e.g. days or weeks of constant operation - the effects generated in a

few simulation cycles (which at most covers a few seconds of real time) will in most cases not result any significant geometric changes. The extrapolation method in the wear simulation routine linearly amplifies the calculated wear depth in each cycle, until the contact geometry is considered to have changed significantly such that additional load cycles are expected to produce noticeably different results. In practice, all wear depth values are amplified until a prescribed critical difference is reached across any single element. This critical difference is specific to each system and the selected mesh refinement, and must therefore be determined individually. This simple extrapolation scheme makes it possible to represent the effect of thousands of cycles with a single computation cycle, since the overall simulation time (i.e. the total number of cycles that have to be simulated) will depend on the extent of wear (e.g. total wear depth or volume loss) rather than the real number of load cycles.
An evaluation of the derived simulation method has subsequently shown that the results could be affected due to an inherent imperfection in the derived technique. While the modified geometry is transferred from one analysis to the next (i.e. from dynamic to static and vice-versa), the stress state is not. Therefore, if the latest configuration is imported to the next analysis, the solver detects a relaxed state even though all elastic deformations have been imported, and the initial load case could produce an increasingly larger contact area. A simple solution to this problem is to release the contact and all applied loads at the end of each dynamic analysis. It was shown that the results will indeed tend towards the correct solution if necessary precautions are taken.
This subsequent study assessed the impact of Archard's wear coefficient and the material's elastic modulus in a parametric study of a simplified shaft/bushing assembly. The wear coefficient varied according to numbers found in an external experimental study on wear in the wastegate's lever/bushing interface, and the elastic modulus varied according to the properties of a common BMW wastegate material at different operating temperatures. The outcome for increasing wear coefficient was unsurprising: First, the volume loss over one complete simulation progresses linearly for all considered magnitudes of the wear coefficient, as predicted by Archard's equation. Second, the total volume at the end of the simulation is also linearly related to the magnitude of the wear coefficient. This also means that the total simulation time required to produce the total volume loss in each instance is directly proportional to the magnitude of the wear coefficient.
The parametric study on the elastic modulus, on the other hand, produced a rather unexpected result: For most values of the material stiffness, the total wear volume and its progression are extremely similar, with only minor differences in their time-history. However, there seems to be a lower threshold at which the total wear volume suddenly increases significantly. This effect was found to be related to the extrapolation scheme and the mesh refinement: For a certain range of material stiffness, the initial contact area is very small and usually only contains a single node. The calculated extrapolation factor in the first cycle is therefore very large and represents a large number of real cycles. At a certain critical E-modulus, the initial contact area will suddenly contain two or more nodes due to larger elastic deformations, resulting in a smaller extrapolation factor and an earlier onset of significant material loss. It is possible that this phenomenon is also found in reality because a certain number of cycles is required to produce an appreciable contact area, and this number is lower if the initial contact area is larger.
Conclusively, the wear simulation routine developed in this study provides a practical basis for experimental validation, feature extensions and fine-tuning.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

A conventional automotive turbocharger, as shown in Fig. 1.1, increases the power and efficiency of a gasoline engine using the energy of its exhaust gases to add more oxygen to the combustion chambers. A higher oxygen level means that more fuel can be burned and thus more power is created by the combustion and transmitted to the wheels. This process is known as *forced induction* [11]. Fig. 1.2 shows the basic operating principle of a turbocharger.

**Figure 1.1:** Schwitzer-BorgWarner S3 turbocharger [1]

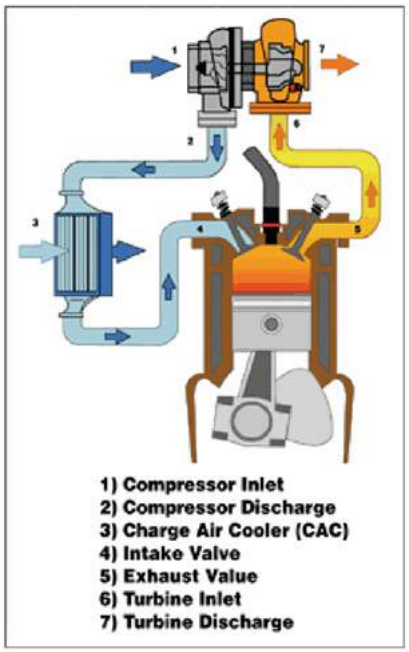


**Figure 1.2:** Operating principle of a conventional turbocharger [1]

As one observes in Fig. 1.2, the exhaust gases flow from the engine cylinder towards the turbocharger and drive the turbine (orange), which is connected via a shaft to the compressor.

The compressor then draws in air from the environment (blue) which subsequently flows through the intercooler back into the engine. An intercooler is necessary to improve the intake air density and to avoid pre-detonation, since pressurization through the turbo's compressor also increases the air temperature. Because the amount of air supplied to the engine depends on its exhaust gases' kinetic energy, the turbocharger tends to provide too much boost at high engine speeds and too little boost at low engine speeds. Automotive turbochargers are usually designed such that even at low engine speeds a small amount of exhaust gas is sufficient to create satisfactory boost pressures. This would however result in excessive boost at high engine speeds [3]. A regulation device is therefore required that is able to control the amount of pressure transmitted by the turbocharger. Conventionally, this is accomplished with a *wastegate device* like the one shown in Fig. 1.3.

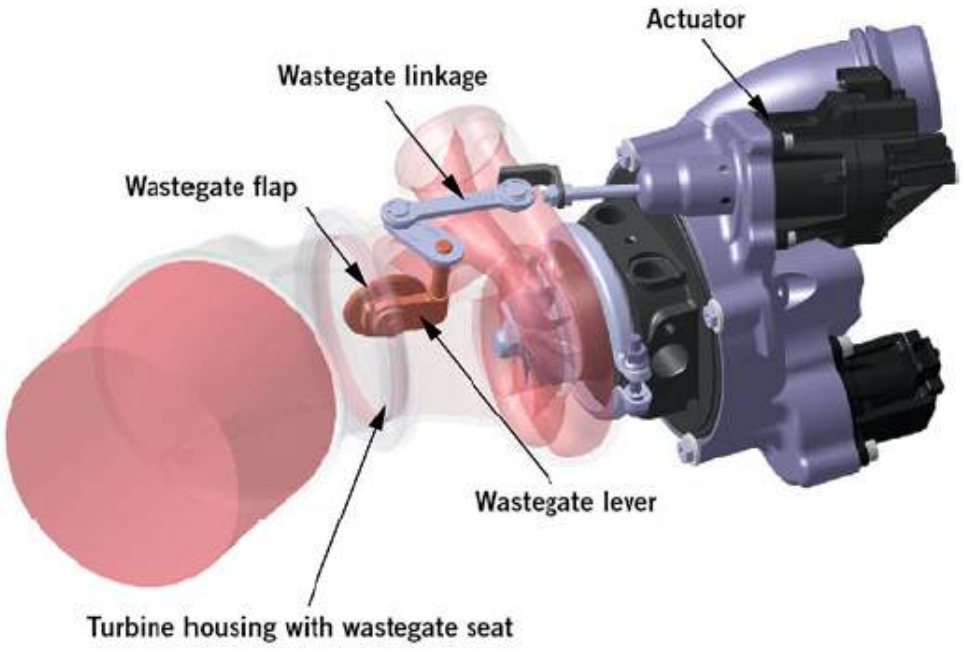


**Figure 1.3:** Wastegate system of a BMW four-cylinder gasoline engine [2]

Essentially, a wastegate is a valve for preventing a certain amount of exhaust gas from entering the turbine. It is situated between the turbine and the turbocharger's inlet. If the *wastegate flap* is opened, part of the exhaust gases that would otherwise act on the turbine instead flow through a bypass directly to the exhaust. The opening angle of the wastegate flap can be controlled very precisely by either a pneumatic or an electric *actuator* such that an optimum amount of boost is created at different engine speeds [3]. The control system detects the requested engine torque (i.e. from the accelerator input), the engine's operating point (engine speed and torque), the ambient pressure and temperature as well as the pressure and temperature at certain locations in the system and, based on those parameters, determines the appropriate opening angle of the wastegate flap [3]. The actuator force is transmitted over the wastegate linkage to the wastegate lever (shown in Fig. 1.3) which then rotates and opens or closes the wastegate flap. The flap's opening angle can be adjusted very precisely such that optimum boost pressure is supplied to the engine at different operating conditions.

The relative movement of adjacent wastegate components leads to wear on their surfaces. Fig. 1.4 shows a wastegate lever with a very pronounced wear damage profile:


**Figure 1.4:** Wear damage on the surface of a wastegate lever [3]

Damage to the wastegate's lever and other parts impairs the proper functioning of the device, meaning that e.g. the bypass can no longer be completely sealed if required, or that the desired flap opening angle cannot be adjusted precisely. Since the wastegate components are located downstream of the exhaust gases, the operating temperature of the system is extremely high (up to more than 900°C [3]). This precludes the usage of conventional lubricants to decrease friction.
Wear in the wastegate system is a cost and safety critical problem. Continuous damage leads to breakage of the components and might result in engine malfunction. It would thus be very beneficial if one could somehow predict the extent of wear in the wastegate after a certain time in operation, and find ways to reduce the damage.
The purpose of this MSc thesis project is to devise a method to simulate the changes in geometry found in the most critical interfaces of the wastegate system. Since wear depends primarily on the contact characteristics in a certain interface, the most reasonable approach would be to quantify the extent of the damage in relation to different contact variables found in a Finite Element Analysis (FEA).
As the surface geometry in the contact area continues to change due to material loss, the contact conditions will evolve accordingly and thus will have to be re-determined with frequently. This means that even for simple geometries, it is not sufficient or not possible to calculate the necessary variables with analytic equations. A comprehensive FEA software package such as Abaqus is able to solve nonlinear contact problems with great accuracy as well as perform modifications to a part's geometry with integrated and well established techniques. Therefore, it is an extremely effective tool for the purposes of this study. The essential output of this project is therefore a routine or a recipe that uses the Finite Element method to describe the progress of wear in a way that is meaningful with regard to the characteristics of the wastegate system. (In Ch. fundamentalPrinciples it will be shown that there are different ways to approach the simulation of wear depending on the research objective.)
This report presents the findings of the thesis work in a coherent fashion such that the reader will be able to understand why certain techniques and approaches were used. The next chapter first provides a summary of the most essential fundamentals of wear from a scientific perspective in order to establish some necessary background information. Following that, Ch. 3 will present a more detailed explanation on the motion and properties of the wastegate system, which is necessary to clarify the main focus of this study, and Ch. 4 will subsequently

discuss several options for the approach to the wear simulation. The most viable method is then selected with regard to efficiency and flexibility. In Ch. 6, the simulation routine is then used to study a simplified system that is representative of a critical interface in the wastegate known to be susceptible to wear. The simulation results are qualitatively compared to the wear damage observed in the real component. The same chapter also contains a parametric study to investigate the effects of a few selected inputs. Finally, Ch. 7 summarizes the most important outcomes and insights of this thesis project. It will also provide some recommendations with regard to the future development of this subject matter and the most important points for improvement.

# Chapter 2

# Fundamental Principles of Wear and Wear Simulation Techniques

This chapter introduces the most important scientific principles of wear and also provides an overview of some of the more prominent studies on the simulation of wear found in literature. To understand the work done in this project within a broader context, the reader should be familiar with some of the basic concepts of wear as a general physical phenomenon. Though wear is an extensively studied damage mechanism due to its prevalence in industrial machines, it is often not clear what processes or effects are exactly involved. This chapter therefore serves to provide some basic information on the topic and to identify the aspects that are significant, or in fact feasible, to be simulated in a Finite Element analysis. It will be shown that while many properties and mechanisms are relevant to material scientists studying the micromechanical behavior of wear, for engineering purposes with an interest in the macroscopic changes of a component a less detailed and more result-oriented approach might be more useful.
In the second part of this chapter, it is shown how wear can be simulated with a Finite Element approach according to various published papers. The methods and results presented in those papers show that it is indeed possible to numerically replicate basic effects of wear in a part's geometry. Those studies are used as a reference since they contain several useful techniques that can be adopted for the current thesis work.

## 2.1 Essential Physical Properties of Wear

In *Engineering Tribology* Williams defines wear as "the progressive damage, involving material loss, which occurs on the surface of a component as a result of its motion relative to the adjacent working parts" [12]. This description captures the effect of wear (progressive damage), its manifestation (material loss), its location (surface of a component), and the requirement for its occurrence (relative motion to adjacent parts). In other words, wear is associated with

the removal of material from interacting surfaces of at least two bodies that move relative to each other. Wear is traditionally quantified with $W$, defined as the total volume of material loss. The concept of a *wear rate* $w$ was originally defined in the works of Holm [13] (1946) and Archard [14] (1956) on the wear of sliding metallic solids, and was given as the *volume loss per unit sliding distance*. Archard's basic equation predicts that the wear rate is linearly related to the normal contact pressure and inversely related to the hardness of the material surface:

$$w = \frac{W}{s} = K\frac{P}{h} \tag{2.1}$$

where $W$ is the total volume loss in $[mm^3]$, $P$ is the normal load in $[N]$, $H$ is the hardness of the material (usually in units of the Vickers Hardness), $s$ is the sliding distance and $K$ is the so-called *wear coefficient*, a constant that is usually determined by experiment for two specific contact partners under certain environmental conditions.
Archard validated his equation for unlubricated sliding of metals, and since then this simple relation has served as a starting point for more complicated models that describe various wear phenomena. It is also the most frequently used constitutive relation for simple Finite Element wear simulations.
Williams' definition implies that wear occurs due to the interaction of solid "working parts", though generally wear is also assumed to include damage arising from the interaction of solid parts with fluids containing hard particles, most commonly referred to as "erosive wear" [10]. Williams identifies the relative motion between two entities as being necessary for wear to occur, though "relative motion" is a macroscopic concept that does not explain how exactly material is removed. To understand or to visualize the process, the contact surfaces have been investigated by scientists using microscopy, which led to the identification of several basic "wear mechanisms" that are responsible for most of the wear damage found for common types of relative motion. While it must be noted that there is no agreement on an official terminology in the study of wear - researchers are found to differ significantly over the classification of similar micromechanical processes - some basic mechanisms are common to almost all wear-related literature that have been consulted. They occur with unequal likelihood for different types of relative motion, as summarized in Czichos and Habig's *Tribology Handbook*, of which an excerpt is shown in Tab. 2.1. In Tab. 2.1 "×" indicates that a particular wear mechanism can become effective for a given wear type.

| Elements in system | Tribological loading | Wear type | Effective mechanisms | | | |
|---|---|---|---|---|---|---|
| | | | Adhesion | Abrasion | Surface fatigue | Tribo-chemical reactions |
| solid / solid | Sliding | Sliding wear | × | × | × | × |
| | Rolling | Rolling wear | × | × | × | × |
| | Impact | Impact wear | × | × | × | × |
| | Oscillations | Oscillation wear | × | × | × | × |
| solid / particles | Sliding | Sliding wear/ Cutting wear/ Erosion | | × | | × |
| | Rolling | Particle rolling wear | | × | × | × |
| | Griding | Grinding wear | | × | × | × |

**Table 2.1:** Classification of wear types and wear mechanisms according to Czichos and Habig [10]

As shown in Tab. 2.1, for all types of relative motion between solids (i.e. sliding wear, rolling wear, impact wear, and oscillation wear), the dominant microscopic wear mechanisms are *adhesion, abrasion, surface fatigue, and tribochemical reactions*. Those are, in fact, the same mechanisms that have been identified in most other textbooks on wear and tribology by several authors. Therefore, to understand how wear occurs in most industrial machinery, it is necessary that those crucial processes are described to some detail, whether or not they are actually used for the thesis work presented later. The following discussion on each of the four primary wear mechanisms are mostly based on the detailed expositions in the book *Engineering Tribology* by Stachoviak and Batchelor [4].

## 2.2 Wear Mechanisms

### 2.2.1 Abrasive Wear

Abrasive wear refers to the removal of material from a soft surface due to its interaction with hard asperities on the adjacent surface [4]. This can happen in several ways, as illustrated in Fig. 2.1.

1. Microcutting:
   The most obvious way of material removal is *microcutting* where, upon relative tangential movement of the macroscopic bodies, the asperities carve into the softer surface to remove a thin strip of material while leaving behind a groove. The hard asperities may be formed by the peaks and troughs in the microscopic surface topology of the harder surface or may consist of hard particles protruding from the surface. This process is comparable to the manufacturing technique of cutting (e.g. turning, milling, sawing). The largest part of abrasive wear is typically caused by microcutting.

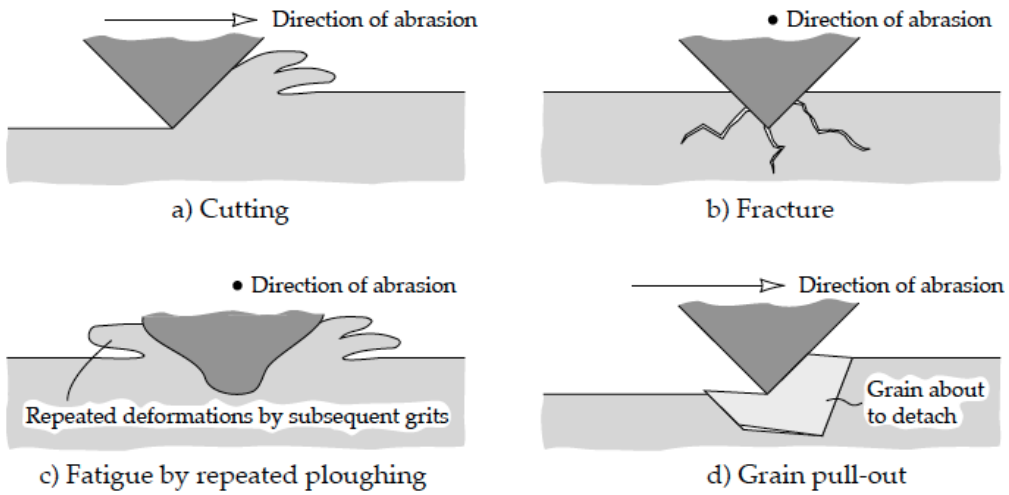


**Figure 2.1:** Mechanisms of abrasive wear: microcutting, fracture, fatigue and grain pull-out [4]

2. Fracture:
   When moving over a surface the hard asperities on the adjacent solid might initiate cracks on the surface and subsurface. These cracks propagate upon repeated interaction with the asperities and eventually lead to the separation of fragments. The extend of abrasive fracture depends on the contact loads (more fracture if loads are high), the shape of the asperities (asperities with sharp edges lead to more cracks) and the hardness of the wear surface (hardened surfaces might experience more abrasive fracture wear due to increased brittleness).

3. Fatigue:
   Abrasive fatigue wear occurs when the micro asperities repeatedly deform the material in a groove, causing fatigue and the associated crack growth and eventual fracture. The effect is therefore similar to abrasive fracture though the cause is different (microscopic cracks caused by cyclic strain rather than high loads). Abrasive fatigue wear is typically less significant than other forms of abrasive wear as it requires several cycles of deformation for a small chunk of material to break off.

4. Grain pull-out:
   Grain pull-out occurs when micro asperities dislodge weakly bonded grains close to the wear surface. This form of abrasive wear is mostly observed in ceramics.

### 2.2.2 Adhesive Wear

Adhesive wear involves a chunk of material being removed from one solid because it sticks tightly to the other contact member [4]. The extracted material therefore migrates from one surface to another. For metal/metal contacts (which is the material combination relevant for this thesis project) adhesion between surfaces is usually prevented by an oxide layer and other contaminants in the contact interface. However, if this oxide layer is rubbed off through relative sliding, the underlying surfaces come into direct contact. Adhesion will eventually occur when a certain minimum gap distance is reached between the micro surfaces (e.g. if

the surfaces are pressed against each other with a normal force). The main cause of adhesion in metal/metal interfaces is attributed to the free electrons inherent to the microstructure of metals. Since valence electrons of metals are not bonded to their atoms, they can easily migrate to a different body if the distance is small enough. The electrons are then able to bond even dissimilar metals. Once adhesion is established, the force required to separate the surfaces is often significantly higher than the initially applied contact force, and often higher than the cohesive force in the weaker material. Relative movement of the bodies will thus lead to the extraction of material from the weaker surface by the stronger surface. The sequence of events associated with adhesive wear is shown schematically in Fig.2.2.

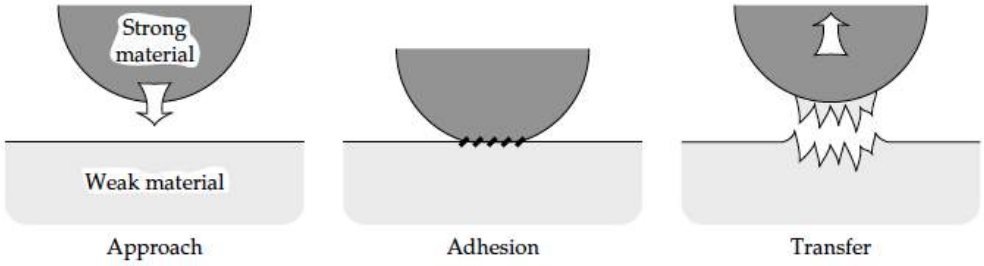


**Figure 2.2:** Process of material transfer due to adhesion [4]

It has been found that adhesive wear is associated with a strong increase in the coefficient of friction $\mu$ between the interfaces. Up to about $\mu = 1.0$ the presence of friction can be explained by adhesion itself, meaning that frictional resistance is caused by asperities coming into contact and adhering to one another. Higher coefficients of friction are explained by the theory of *asperity junction growth* which predicts a further increase in friction due to the plastic deformation of asperities. Initially, the yield criterion is reached due to the application of a normal force. Any additional tangential forces will therefore easily lead to more plastic deformations and thus quickly increase the microscopic contact area between the asperities. The increased contact area in turn is able to carry higher tangential loads, i.e. higher frictional shear stresses which lead to even more plastic deformation. In some cases this vicious cycle may eventually result in an unstable growth of $\mu$ until friction becomes so high that relative motion between the surfaces is arrested, which is known as *seizure*. There exists an analytic equation that expresses the increase in asperity contact area in terms of the friction force, the normal force and an empirical constant with magnitude of about 10. Based on this equation it is possible to assess the likelihood of the onset of seizure, but for brevity's sake this discussion is omitted here. More details can be found in Stachowiak and Gwidon's *Engineering Tribology* [4].
Adhesive wear in combination with seizure is one of the most common failure modes seen in bearings, gears or cams. In a wastegate system the lever/bushing interaction is fundamentally a shaft rotating in a bushing with a small clearance, so one can reasonably expect adhesive wear to be a highly significant wear mechanism.

### 2.2.3 Corrosive Wear/ Oxidative Wear

Corrosive or oxidative wear is said to occur if the wear process involves chemical reactions of the interacting materials with its environment or interface medium [4]. While corrosive wear

refers to any wear process affected by corrosion, oxidative wear is a more specific term describing wear due to a material's surface reacting with oxygen. Naturally these wear mechanisms are commonly found in metallic contacts where the formation of an oxide layer is inevitable in most environmental conditions. Interestingly, depending on the stability of the oxide layer the extend of wear can be increased or decreased. As shown in Fig. 2.3 the development of an oxide layer may lead to different scenarios as relative motion of the bodies continues:

1. The created oxide layer is relatively thin and stable such that direct contact between the mating surfaces is prevented. As discussed in the previous section, this outcome is rather favorable as it counteracts adhesion which often leads to severe wear. The oxide film may also act as a lubricant and decrease friction.
2. The created oxide layer is unstable and easily rubbed off from the surface. Over the duration of the interaction a new oxide layer is repeatedly built and destroyed such that wear is exacerbated by the oxidation.
3. When parts of the oxide layers are removed, a galvanic coupling is established between the broken interface of the film and the underlying material, resulting in additional local corrosion. Severe pitting is consequently observed on the worn surface.
4. If the oxide layer is very weak and removed easily it does not directly contribute to wear. In this case other wear mechanisms act independently and local corrosion on the worn surface may add to the material loss.

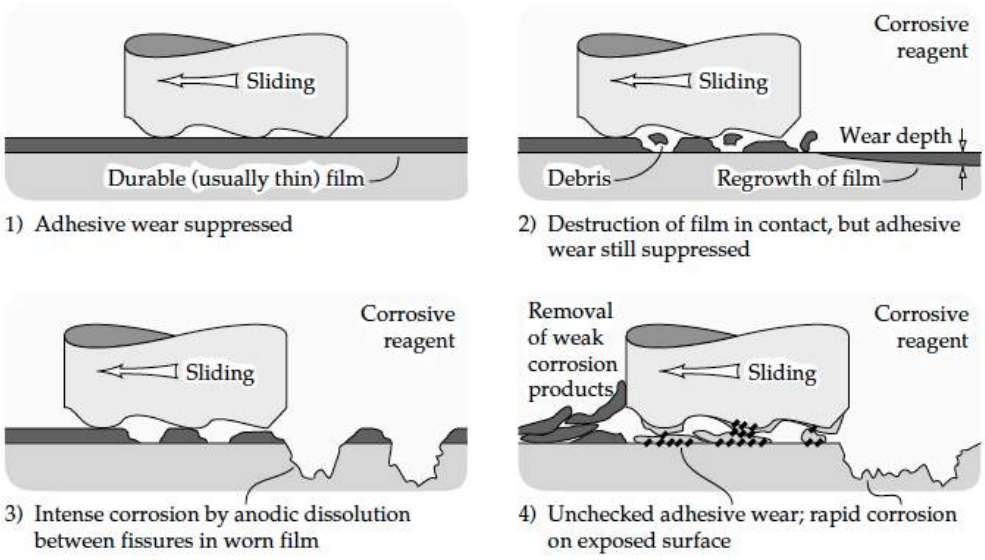


**Figure 2.3:** Models of interaction between a corrosive agent and a worn surface; Possible scenarios 1-4 [4]

Though sometimes oxidation may be favorable for decreasing wear (scenario 1, upper left picture in 2.3), the most common result of oxidation is scenario 2 in 2.3 where oxide layers are continuously formed and ablated. If an existing oxide layer prevents adhesion the wear rate is found to be relatively low and is thus labelled "mild wear". Otherwise if oxide layers

are abraded and adhesive wear occurs the process is called “severe wear". Over the duration of metal/metal interaction one frequently observes a transition from mild wear to severe wear, characterized by a sudden increase in the wear rate [14].

**Temperature Effects on Oxidative wear**

Oxidation and thus oxidative wear is known to be affected by temperature. Since the wastegate valve is located in the turbine housing downstream of the exhaust gases, its operating temperature is exceptionally high. It is therefore appropriate to briefly discuss the general influence of high temperatures on oxidative wear.
At low temperatures (typically up to 200°C for steels) the formation of the oxide layer is enabled by a position exchange between the oxygen and iron atoms in the oxide film's crystal structure. This process is only effective over thin oxide films, so their thickness is limited to a few nanometers. At higher temperatures, however, the atoms have much more kinetic energy and can be transferred effectively over thick oxide films. Upwards of 500°C the thickness of oxide films is therefore not limited and can reach several micrometers. The thin, protective oxide film associated with mild wear is thus found primarily at low temperatures while high temperatures usually lead to severe wear. On the other hand, at high temperatures of a few hundred degrees centigrade fine wear debris might oxidize again and form a second protective layer, called the “glaze layer", over the original oxide layer. In this case a transition back to mild wear may be observed. For example, Nimonic 80A (a nickel-based alloy used for high-temperature engine components) was observed to transition to mild wear at 250°C after a glaze layer is formed.

### 2.2.4 Fatigue Wear

Material loss might also occur directly due to fracture, meaning that chunks of material are separated from the surface by micro cracks [4]. When two solids slide over each other, large deformations and grain reorientations in the direction of sliding are found close to the contact surfaces. The original grain structure is transformed and develops cells separated by accumulated dislocations. The boundaries of those *dislocation cells* are weak spots along which cracks are likely to nucleate (due to the presence of voids) and propagate as a result of cyclic loading. Eventually particles may be removed due to a crack that started either on the surface or in the subsurface of the material, which is known as *fatigue wear*. Surface and subsurface crack nucleation have distinct characteristics which are discussed here briefly.

**Fatigue Wear due to Surface Cracks**
If asperities move repeatedly over a surface, cracks may be initiated at microscopic points of stress concentration that exist due to the surface's inherent roughness (microscopically a surface always exhibits an uneven topology). A crack can easily grow into the subsurface along boundaries of dislocation cells, occasionally splitting into two or more cracks that propagate in different directions. When the crack eventually reconnects with the surface, a block of material is separated and easily removed. A schematic is shown in Fig. 2.4.

**Fatigue Wear due to Subsurface Cracks**
Surface cracks might not be initiated if surface asperities are worn off quickly. Additionally,

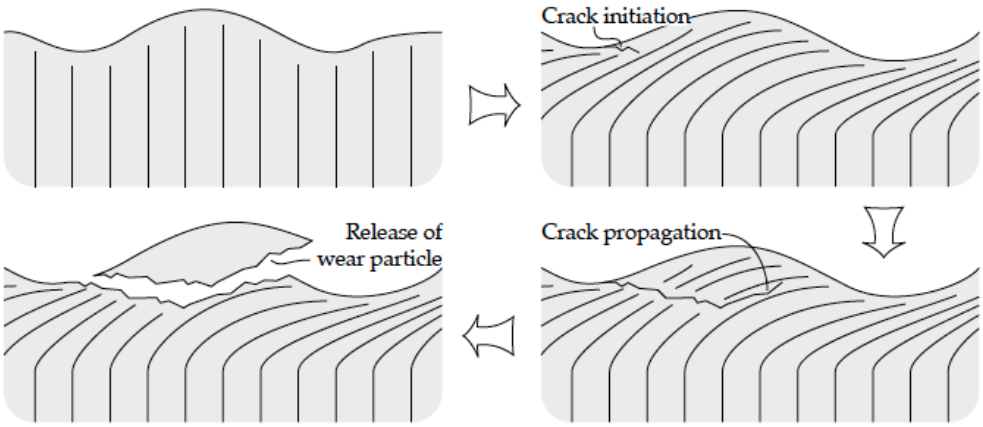


**Figure 2.4:** Wear particle formation due to growth of surface initiated cracks [4]

one often finds a compressive, hydrostatic stress state as well as significant plastic deformations close to the surface, both of which are known to counteract crack growth. At a certain depth from the surface conditions are often more favorable for crack nucleation and propagation. Cracks are often created at points with sufficient tensile stresses or at dislocation cell boundaries where voids have developed due to the accumulation of dislocations. With additional load cycles a subsurface crack can propagate over a significant distance without being visible from the outside, such that the wear debris attains the shape of a long strip once the crack has finally reached the surface. This form of fatigue wear is therefore also known as *delamination wear*. The progression of crack initiation, crack propagation and wear debris formation from the subsurface is shown schematically in Fig. 2.5.

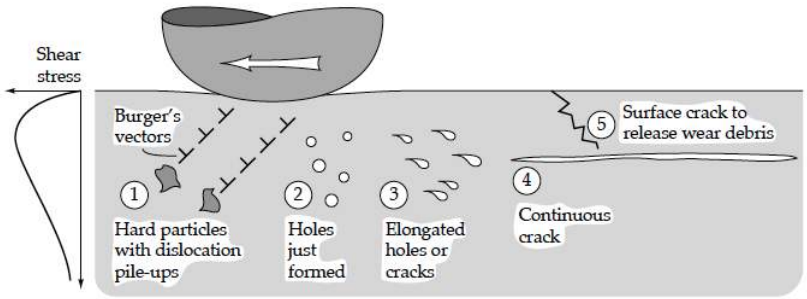


**Figure 2.5:** Subsurface crack initiation and growth [4]

For metals such as copper or iron it was found that higher levels of oxygen in the environment leads to more severe fatigue wear. This can be explained by the fact that rapid oxidation in cracks prevents a natural crack closure process where separated grains manage to reconnect. For more inert metals, such as gold, increased oxygen does not lead to more fatigue wear as the metal is less reactive to oxygen.

Wear mechanisms discussed in the previous sections are the most prominent ones that are observed for the interaction of solids, and there are many others that pertain to e.g. solids and fluids, solids and liquids or solids and gases. For example, *erosive wear* is a well-known mechanism that results from macroscopic particles impinging on a solid part (as e.g. seen in

grit-blasting). In solid/fluid interactions this mechanism is more specifically known as *cavitation erosion*, describing wear of the solid due to impingement of liquid or gaseous bubbles. These processes are not discussed in detail since they are not relevant to the project. Most of the wear observed in the wastegate system result from the interaction of solid parts.

## 2.3 Relevant aspects for macroscopic FE wear simulations

The central issue of this project is to simulate geometric changes in the parts of a wastegate due to wear. Having discussed the most commonly occurring wear mechanisms for solid/solid interactions, one realizes that their distinctions are only apparent on a microscopic scale. If this was a scientific study with the objective to, for example, simulate how a piece of material might be removed if certain wear-inducing forces are applied to a part's surface, it would certainly make sense to consider all the wear mechanisms that have been explained previously. In this case the simulation would be concerned with a very small part of a contact surface and attempt to portray the behavior of both the remaining surface and the piece of separated material.
However, this is clearly not the goal of the current thesis project. Instead of looking at the detailed ways in which material is removed, the focus is on reproducing the changes in the macroscopic shape of wastegate components that are subject to a certain environments and loading conditions. The investigation is therefore conducted on a much bigger scale than would be adequate if one intends to, for instance, distinguish between abrasive or adhesive wear. In conclusion, the wear mechanisms that are essential to e.g. tribological research are of little interest for the purposes of this study. To depict the geometric changes of macroscopic parts, one must concentrate on processes that take place on a similar scale.
In Tab. 2.1 one observes that all wear mechanisms are classified under certain types of relative motion: sliding, rolling, impact, and oscillation. Those interactions can be easily distinguished by observing the motion of real parts. For example, according to camera recordings and dynamic simulations of the wastegate's motion, the wear damage in the wastegate lever occurs primarily due to oscillatory sliding against the inner surface of the bushing, whereas in the flap/lever interface impact wear seems to be more critical. The solution variables pertaining to those two types of relative motion are readily available from the output of an FE contact analysis. Simply by looking at the names of the types of motion one might infer that relative sliding distance and impact velocity are most likely relevant variables. One realizes, therefore, that the macroscopic wastegate wear simulation should be classified on the level of *relative motion* rather than on the level of *wear mechanisms*. The severity of wear will of course depend on the effective or dominant wear mechanisms in a certain interaction, but it is not strictly necessary to be aware of their individual contributions to the damage overall. As shown in Tab. 2.1, most likely all four mechanisms will be effective to some degree. In this case it is more practical to investigate their combined effect and incorporate it into one or two coefficients found by experiment under different loading conditions. Certainly, if it was possible to conduct extensive physical testing, one might eventually be able to separate the effects of different wear mechanisms and devise more detailed models based on those findings, but for this project it is more reasonable to consider on the aggregate effect.
Having established the focus of the wear simulations, the next sections will review several published studies that investigated different simulation techniques and characteristics. The

goal of the discussion is to present a few viable approaches to simulating wear within the possibilities of Finite Element Modelling.

## 2.4 Wear Simulations in Literature

The most essential task of the thesis project is to develop a numerical routine that is able to simulate material loss in various wastegate interfaces. Since the wastegate components form a system with a complicated geometry and load collective, it is not possible to calculate wear analytically by simply applying the equation. A numerical calculation scheme employing the Finite Element Method (FEM) is a much more viable approach. The current chapter presents a compilation of the methods for wear simulation found in literature.

### 2.4.1 Geometry Update by Moving Surface Nodes Only

The most important parameters to calculate and replicate in the simulation are 1. the volume or mass of the removed material and 2. the change in surface geometry due to material loss. The change in geometry in particular will affect the contact conditions for the next simulation cycles (e.g. contact pressure, shear stresses etc.), which will in turn affect the further progress of the wear rate. The change in geometry of an FE model must be accomplished by somehow adjusting the position of individual nodes. The most basic approach is to simply move the nodes on the contact surfaces of the model based on contact solutions found on each node. Several studies that follow this approach have been published in the past and shall be discussed in this section.

#### Podra and Andersson: Sliding Wear Simulation of Pin-on-Disk Tribometer

A standard device for the investigation of tribological parameters is the pin-on-disk tribometer, where a small pin is made to slide on a large circular disk. The set-up is shown schematically in Fig. 2.6.

The uniform sliding performed by the pin is easy to model and experimental validation results are readily available. Podra and Andersson modelled the pin-on-disk sliding wear using the FEM tool ANSYS and a simple assembly: a two-dimensional cross-section of the pin is made to slide at uniform velocity over a flat surface with a constant normal force acting on the pin, as shown in Fig. 2.7 [5]:

In Fig. 2.7, both solid parts are modelled with quadrilateral, 4-node elements with two translational degrees of freedom per node (PLANE42 elements in ANSYS), while special contact elements - 2D point-to-surface elements (CONTAC48) in this case - are used to model the contact surfaces between pin and disk.
After the general contact problem is solved internally by ANSYS, the wear depth (i.e. the displacement of the contact nodes) is calculated in a separately coded post-processor according to Archard's equation (the derivation of an FE-adapted version of Archard's equation for will be shown in Ch. 4).
Podra and Andersson were able to obtain results that agree reasonably well with those obtained by experiment, though a relatively large deviation of $\pm 40 - 60\%$ was found for the

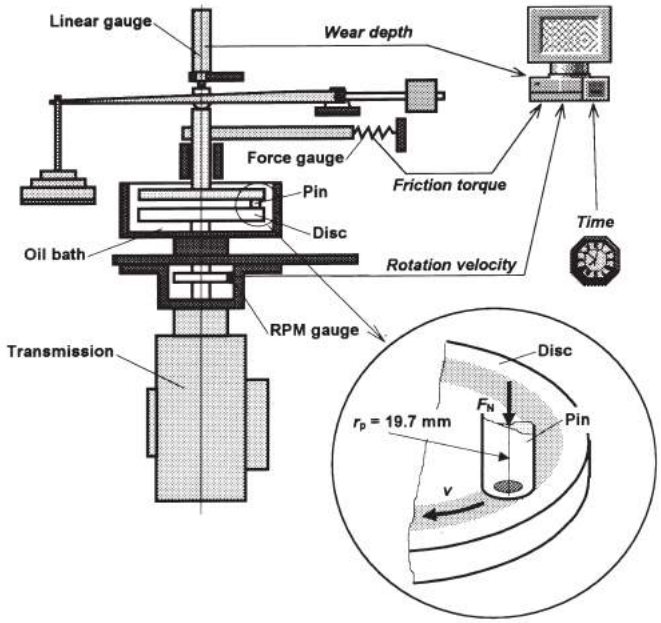


**Figure 2.6:** Pin-on-disk test rig [5]

depth of the wear profile. This discrepancy is not very surprising since a constant wear coefficient was assumed for Archard's equation for the entire simulation. In reality, one typically finds two or more periods where the wear rate changes from one steady value to a different one (and by extension also the wear coefficient). In order to improve the results one could therefore consider either conducting more detailed tests to find the evolution of $K$ with respect to the duration of wear (or the sliding distance), or apply a more advanced analytic model.

### SÃűderberg and Andersson: Wear Simulation at the Pad-to-Rotor Interface in a Disk Brake

Soderberg and Andersson simulated wear on the pad-to-rotor interface in a disk brake using ANSYS again using Archard's equation. This wear simulation considers a more complicated system with multiple components and interfaces. All contact surfaces were modelled with a layer of contact elements on top of the parts' solid elements. Fig. 2.8 shows the model assembly of the back plate, brake pad and rotor with the applied brake force and rotation of the rotor.

In the simulation is was assumed that 1. the rotor wear is negligible, thus only the brake pad wear is modelled, and 2. the rotor rotates with a constant velocity, which corresponds to test conditions rather than real braking conditions where the wheel is slowed down by the braking force. The incremental sliding distance $\Delta s_{i,j}$ in time increment $j$ of each surface node $i$ can therefore be calculated using the rotor's angular displacement and the position of the node with respect to the center of the rotor:

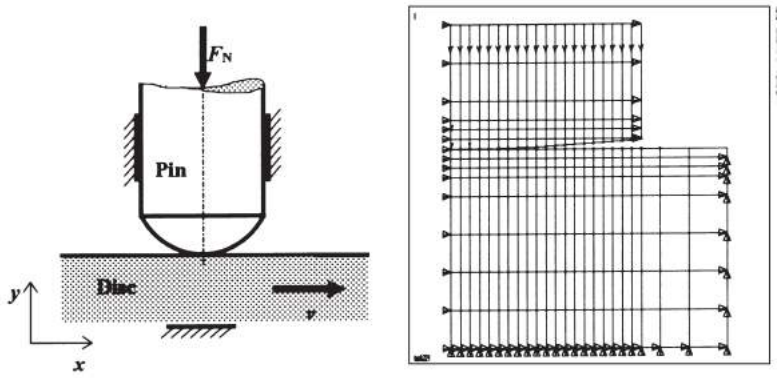


**Figure 2.7:** Pin-on-disk rubbing contact and the FE model structure [5]

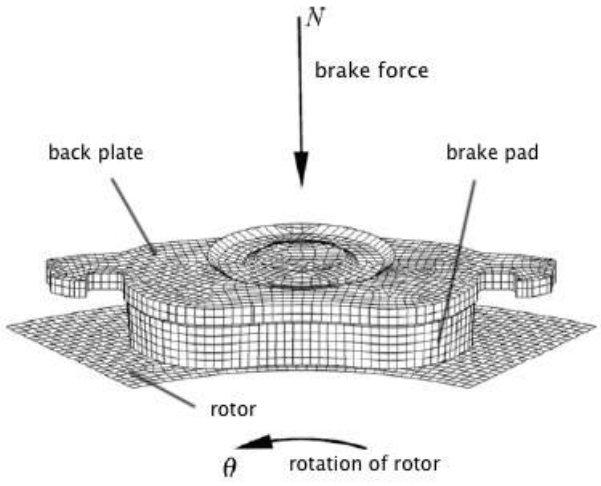


**Figure 2.8:** FE model with applied normal brake force and rotation of rotor [6]

$$\Delta s_{i,j} = n_j 2\pi r_i \tag{2.2}$$

where $n_j$ is the number of rotor revolutions during the $j$th step and $r_i$ is the distance of node $i$ to the center of the rotor. It seems, therefore, that one wear increment may contain multiple rotations of the rotor, which would be reasonable if the rotor spins at a high velocity.
A notable assumption made by SÃűderberg and Andersson in this study is that the brake pad's contact nodes are simply moved perpendicular to the initial surface. The authors note that it would be more accurate to compute the local normal vector at each node and apply the wear depth in that direction, since the local normal direction might change over the course of the simulation. One observes, for example, that the nodes farther away from the center of the rotor should lose more material since they experience a longer sliding distance per revolution. In reality, the surface on the outer side of the brake pad should therefore attain a slightly different orientation as wear progresses. However, SÃűderberg and Andersson justify their assumption on the grounds that 1. the initial surfaces of pad and rotor conform to each other and 2. the wear depths are expected to be extremely small compared to the brake pad's surface dimensions. For the implementation of an external wear simulator, calculating the

local surface normal for each node could significantly complicate the coding, especially in 3D models. One should therefore evaluate for one's own purposes whether simplifications can be made.

As mentioned, the simulation routines discussed in this subsection follow a rather simple technique where only the surface nodes are displaced by a calculated wear depth. This approach is viable as long as the wear depths are small compared to the size of the surface elements [15] [6]. In order to simulate substantial wear with a significant change in geometry one would need rather large surface elements. This, in turn, would be disadvantageous for achieving a good contact solution [16] [7]. In most contact problems it is for example common to apply a finer mesh in the contact region than in the rest of the model such that good approximations of the contact pressure and shear stress distributions can be achieved. The following section presents a studies where this problem is solved by remeshing the part geometry.

## 2.5 Geometry Update with Part Remeshing

If one attempts to simulate a large number of wear cycles, at some point the surface nodes of the contact elements will be displaced so much that these elements become severely distorted or are even completely consumed. This, of course, will lead to inaccuracies in the solution and eventually nonconvergence. To maintain an adequate mesh quality it is therefore necessary to also adjust the positions of the nodes lying beneath the contact surface. A straightforward way to accomplish this is to generate a new mesh if a certain wear depth is reached. The following paragraphs are going to address two studies that employ such a remeshing approach. The paper by Hegadekatte et al. provides an in-depth discussion on the applied technique and will thus be discussed in detail.

### Hegadekatte et al: Simulation of Dry Sliding Wear

Hegadekatte et al. developed a post-processor routine (the so-called "wear processor") that works in conjunction with the commercial FE software Abaqus to simulate dry sliding wear in both a 2D and a 3D model [7]. The 2D model depicts a curved ring in steady rotation against a flat surface, while the 3D model approximates the pin-on-disk tribometer set-up shown in Fig. 2.6. The FE models are shown in Figs. 2.9 and 2.10.

The simulation routine used by Hegadekatte et al. can be summarized as follows: Like in Podra and Andersson's study, the basic contact problem is first solved using Abaqus/Standard, whereupon the contact pressure and sliding distance are obtained at each contact node. The wear depth is subsequently calculated using the wear processor in a post-processing step. The wear processor then checks if the wear depth has reached a critical percentage $\delta$ of the surface element height; if that is the case, then the part is remeshed, where the calculated wear depth is applied as a *displacement boundary condition* to the contact nodes (this method is known as the *boudary displacement method* and is traditionally used for structural optimization problems. A detailed explanation can be found in [17] and [18]); if the critical wear depth has not been reached, the wear simulation proceeds to the next step and a greater value of the

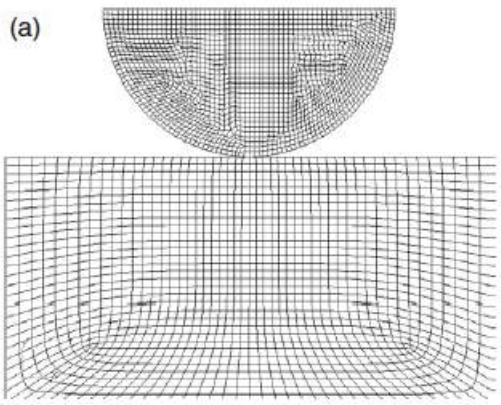


**Figure 2.9:** FE mesh of a hemispherical ring on a flat disk [7]

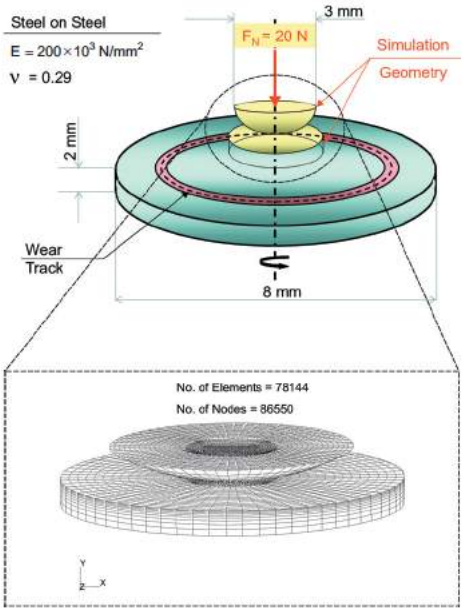


**Figure 2.10:** Modelling of a spherical pin revolving over a disk [7]

wear depth is calculated. After remeshing, the static contact problem is solved for the next increment and the simulation continues until the maximum sliding distance $s_{max}$ is reached. The flow chart for the whole simulation routine is shown in Fig. 2.11.

The remeshing technique used in this routine allows for an arbitrarily large wear depth and a good mesh refinement such that realistic contact stress distributions can be obtained. However, remeshing obviously adds significantly to the computational expense of the wear simulation. One observes, for example, that the nonlinear contact problem has to be solved each time the part is remeshed. Therefore, the smaller the contact elements are the more frequent the remeshing becomes, and consequently the more contact problems have to be solved.
In conclusion, while remeshing solves the problem of having a “wear limit", it is computationally expensive and difficult to implement. Thus it is fortunate that Abaqus provides an in-built function that is able to perform automatic mesh smoothing in accordance with user-prescribed node shifting. This techniques is known as *Arbitrary Lagrangian Eulerian* (ALE) adaptive meshing and is going to be discussed in the following section.

### 2.5.1 Wear Simulation with UMESHMOTION and ALE Adaptive Meshing

In Abaqus/Explicit and Abaqus/Standard the ALE mesh smoothing algorithm is generally used to improve the mesh quality in response to distortions for whatever reason. While it is used in Abaqus/Explicit to maintain the mesh quality for large material deformations (for example in part forming simulations), in Abaqus/Standard it is especially well suited to simulate surface material ablation (the other main application being the adjustment of acoustic meshes). As the name suggests, ALE adaptive meshing in Abaqus/Standard employs both the Langrangian and the Eulerian method of analysis. In each increment the solver first applies the conventional Lagrangian method, where the mesh follows the underlying material,

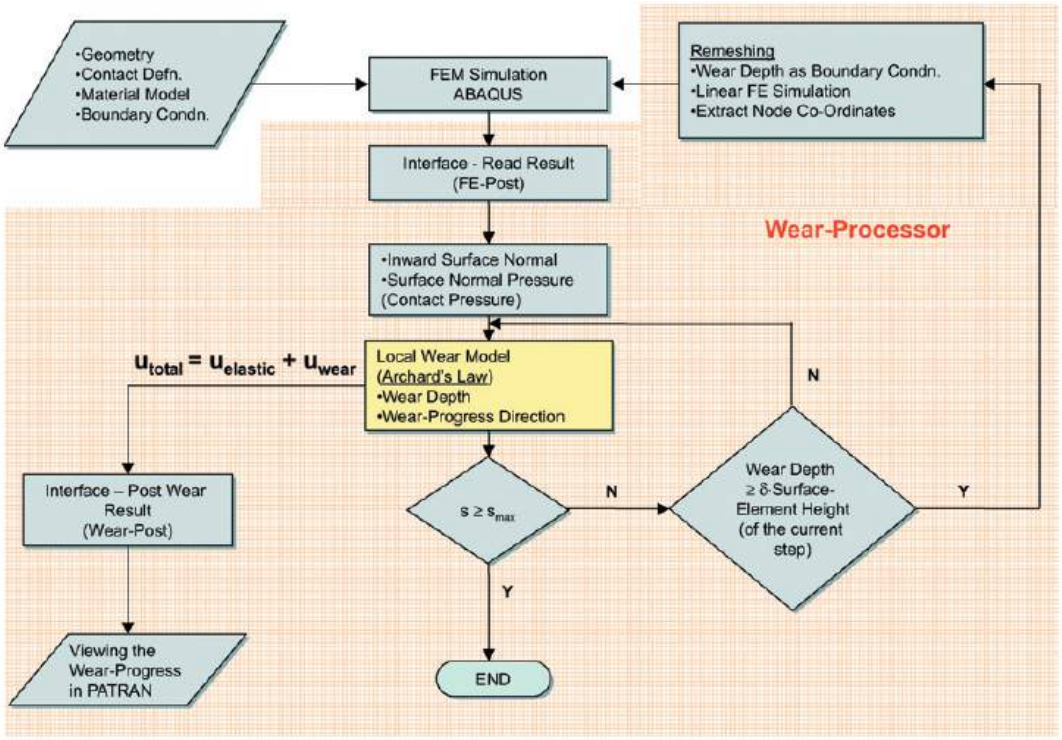


**Figure 2.11:** Flow chart of the Wear-Processor [7]

in order to obtain an equilibrium solution. The mesh is subsequently updated in the Eulerian step where the nodes are able to move independently from the material. The Eulerian analysis is further divided into a *mesh sweeping* step and an *advection* step. *Mesh sweeping* describes the actual procedure of shifting the nodes to recover the mesh quality. Each node in the adaptive mesh domain is adjusted according to Eq. (2.3):

$$x_{i+1} = N^N x_i^N \tag{2.3}$$

where $x_{i+1}$ is the new position of a node, $x_i^N$ are the positions of the neighboring nodes from the last mesh sweep step, and $N^N$ are weight functions that are obtained using either the *Original configuration projection* algorithm (where the node is repositioned while trying to minimize its distance to a projection of the mesh back to its original configuration), or the *Volume smoothing* algorithm (where the repositioning is performed based on a weighted average of the volume of neighboring elements), or a weighted combination of those functions. The number of mesh sweeps per increment may be specified by the user. Multiple mesh sweeps per increment might be beneficial if large element distortions are expected. Mesh sweeping in ALE adaptive meshing only modifies the position of the nodes, meaning that the type, number, and connectivity of the original elements remain unchanged.
Following the mesh sweeping step, the solution variables obtained in the Lagrangian analysis are mapped onto the new mesh in a process known as *advection* [19]. The mapping is done using the Lax-Wendroff method, where the nodal point quantities are transformed to material point quantities and subsequently mapped by relating their spatial derivative to their time derivative. More details on the Lax-Wendroff method can be found in [20].

The ALE adaptive meshing technique in Abaqus/Standard allows the user to prescribe special nodal displacements at the boundaries of the adaptive mesh domain by defining so-called *adaptive mesh constraints*. Simple constraints - such as Lagrangian constraints to prevent node repositioning, or specific displacement or velocity values - can be specified in Abaqus/-CAE (Abaqus' GUI). To define more complicated constraints - such as wear depths that have to be calculated based on nodal solution variables - one must code the node shift with a separate user-defined subroutine, known as UMESHMOTION. The operating principle of UMESHMOTION is discussed briefly in the following.

### Defintion of Wear using UMESHMOTION [21]

UMESHMOTION is a simple subroutine that makes it possible for the user to freely define the magnitude and direction of the displacement of certain nodes, which is exactly what is necessary for the simulation of material ablation at contact surfaces. In a wear simulation one would commonly obtain certain nodal point quantities (e.g. contact pressure and sliding distance) by calling different inbuilt utility routines, calculate the wear depth according to an analytic equation, and finally apply the wear depth in a certain direction as nodal displacements. The nodal displacements are expressed in the vector ULOCAL, which is defined in the local coordinate system ALOCAL (a 2×2 or 3×3 matrix listing the unit vectors in global coordinates).
The wear simulation approach provided by Abaqus can therefore be summarized as follows: Subroutine UMESHMOTION prescribes surface node displacements according to analytic wear models, such that the ALE adaptive meshing algorithm can adjust the nodes in the rest of the part domain in order to maintain a good mesh quality. This method has been applied by many researchers in recent years for wear simulations in simple geometries. The following section shall discuss a small selection of those studies.

### Bortoleto et al: 3D Wear simulation of flat pin-on-disk sliding

UMESHMOTION combined with ALE adaptive meshing was used by Bortoleto et al. to simulate wear on a disk surface resulting from uniform sliding of a flat pin [8]. The model set-up and simulation flowchart are shown in Figs. 2.12 and 2.13:

As one observes in Fig. 2.12, the mesh size of the model is smaller at locations where the pin is expected to be in contact with the disk, which is possible because the mesh in the regions comprised of 8-node brick elements is adjusted continuously. The flowchart in Fig. 2.13 shows that in principle, it is only necessary for the user to specify the analytic wear model at the contact nodes in UMESHMOTION. Abaqus will access the same UMESHMOTION for each individual contact node and adjust the rest of the mesh according to the ablation depth.
Bortoleo et. al were able to approximate the changes of the disk's surface topology with this approach and using Archard's wear equation. The wear coefficient $K$ was calibrated with pin-on-disk experimental data for three different normal loads. However, at higher normal loads (35N, 70N and 140N) the numerically predicted wear volume deviated from experimental results by approximately 100% (overestimated). The authors attribute this error to inaccuracies in the estimation of the wear coefficient $K$, which was calibrated without accounting for the running-in period, where the global wear coefficient is typically higher than during

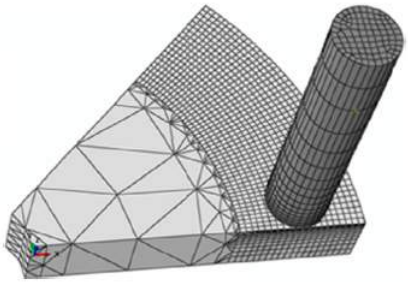

**Figure 2.12:** Model set-up and mesh of pin-on-disk model [8]

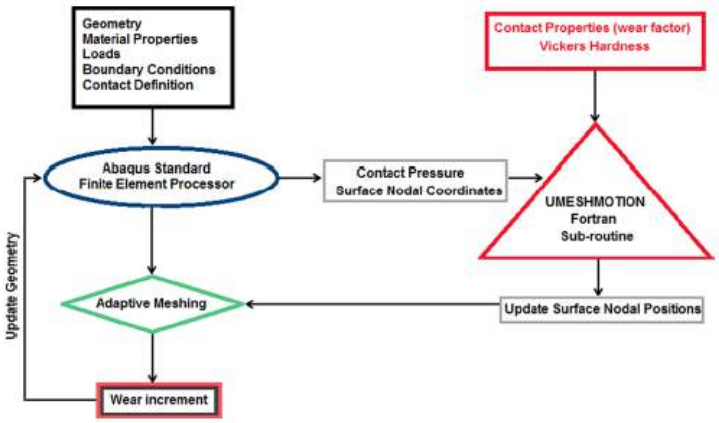


**Figure 2.13:** Simulation flowchart of with UMESHMOTION subroutine [8]

steady material ablation.

### 2.5.2 Martinez et al: 3D Wear Simulation of Polymer Cylinder Sliding on Steel

Martinez et al. used Abaqus' ALE adaptive meshing technique and UMESHMOTION to simulate wear of a thermoplastic polyurethane elastomer (TPU) sliding on a steel block, and validated their results in a tribometer test. The TPU is modelled as a linear elastic material since Abaqus has difficulties with advection of hyperelastic material parameters in adaptive meshing [22]. The 3D model set-up and the structure of the implemented UMESHMOTION subroutine are shown in Figs. 2.14 and 2.15.

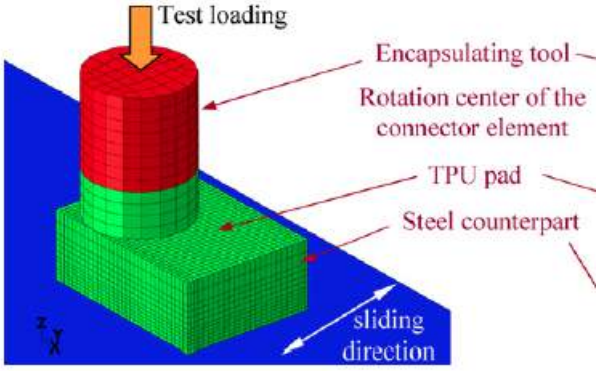


**Figure 2.14:** FE model of the validation wear tribotests [9]

With carefully calibrated wear and friction parameters, Martinez et al. observed in their validation an average deviation of 15% between the simulated and experimental results. It was also observed that the accuracy of the numerical predictions are very much dependent on how well the initial running-in phase is replicated, since the wear rate in that regime varies considerably. The authors therefore recommended to use more simulation steps in the

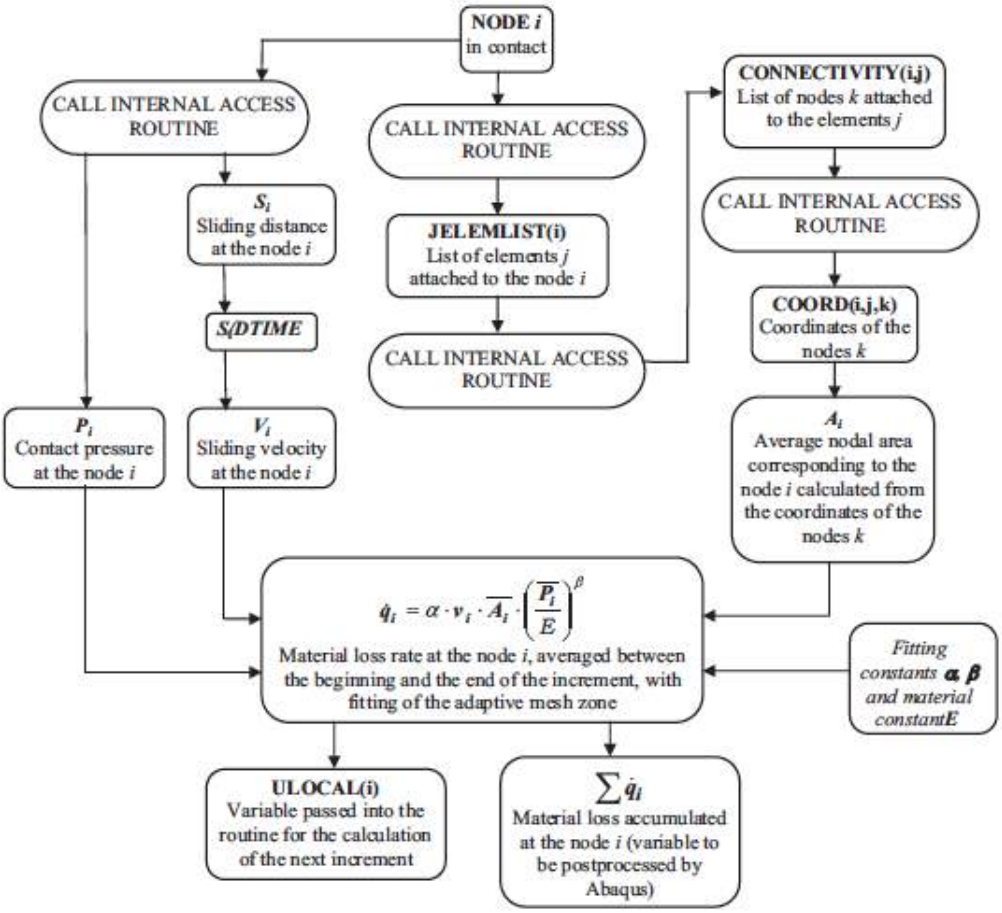


**Figure 2.15:** Implementation of the wear model for node i [9]

running-in phase to properly characterize the wear constants at that stage.
The short overview of simulation methods were presented in this section to show some examples of state-of-the art wear simulations studies that were conducted by professional researchers. A brief discussion on the ALE adaptive meshing technique provided in the Abaqus software package explained how it can be used to change a part's geometry based on solution variables obtained during the analysis. It turns out that the implementation of a basic wear simulation program based on adaptive meshing is very simple and efficient compared to other possible approaches: The displacement of surface nodes, which is the fundamental process through which the geometry is modified, does not have to be coded directly to be consistent with the rest of the mesh domain. Instead, the user is only required to specify the magnitude and direction of node shift, and the algorithm of adaptive meshing will map all relevant solution variables to the new node locations. Also, a good mesh quality is maintained throughout the analysis since the entire mesh is adjusted to accommodate the surface node displacements. Compared to other techniques, adaptive meshing is also much more efficient since the part does not have to be reconstructed in each cycle. Its effectiveness was demonstrated by several wear simulation studies published in recent years. Due to the simplicity and efficiency, the adaptive meshing technique is selected as the basic procedure for geometric modifications.

## 2.6 Summary

This chapter introduced some of the most fundamental principles and mechanisms of wear. To understand the focus of the work done in this thesis project, it is necessary to be somewhat familiar with its scientific context. Therefore, the primary goal here was to provide the reader with some basic information on the most common processes that induce wear damage. It was eventually concluded that the most effective and sensible approach for the current study is to focus on the macroscopic types of relative motion that lead to wear, rather than the actual microscopic wear mechanisms that are more prominently featured in Tribology textbooks and articles. The primary reason for that distinction is that the client company is more interested in reproducing the changes in the wastegate components' geometry due to wear. The detailed ways in which small amounts of material can be separated are less important. Consequently, Archard's simple wear equation was found to be a suitable analytic model for calculating the volume of material loss due to wear. This simple relation can be easily implemented in an FE simulation routine because the result depends on output variables that are directly obtained from the numerical contact solution.
The second part of this chapter discussed different wear simulation routines presented in several scientific papers. It was found that the basic approach for simulating wear was to modify the position of contact nodes where relevant output variables had been registered. While the node displacement can be accomplished in different ways, it was concluded that the Arbitrary Lagrangian-Eulerian adaptive meshing technique is the most efficient and practical method. With this integrated Abaqus capability, the mesh is adjusted rather than reconstructed in each turn to accommodate the geometric changes at the contact surface. To accomplish a simple wear simulation it is only necessary to specify the magnitude and direction of node displacement in a user-defined subroutine. The work presented in the following chapter of this report is therefore fundamentally based on the adaptive meshing technique.
Having introduced the fundamental principles of wear as well as some conventional wear simulation techniques, the next chapter will investigate the motion and wear characteristics of the wastegate system itself in more detail.

# Chapter 3

# Wastegate Motion and Wear Characteristics

To understand what is required of the wear simulation as well as what kind of meaningful results can be produced with the available resources, it is important that the wastegate's system characteristics are explained in more detail. This chapter will thus take a closer look at the loads acting on the wastegate and the behavior of the most critical interfaces in the system. The discussion also serves to provide the reader with more background information such that they are able to gain some insight into a rather obscure automotive device.

## 3.1 Wastegate components assembly and wear-inducing load cases

The FE model in Fig. 3.1 shows the complete assembly of the wastegate system with all essential components, as well as the primary load case during operation:

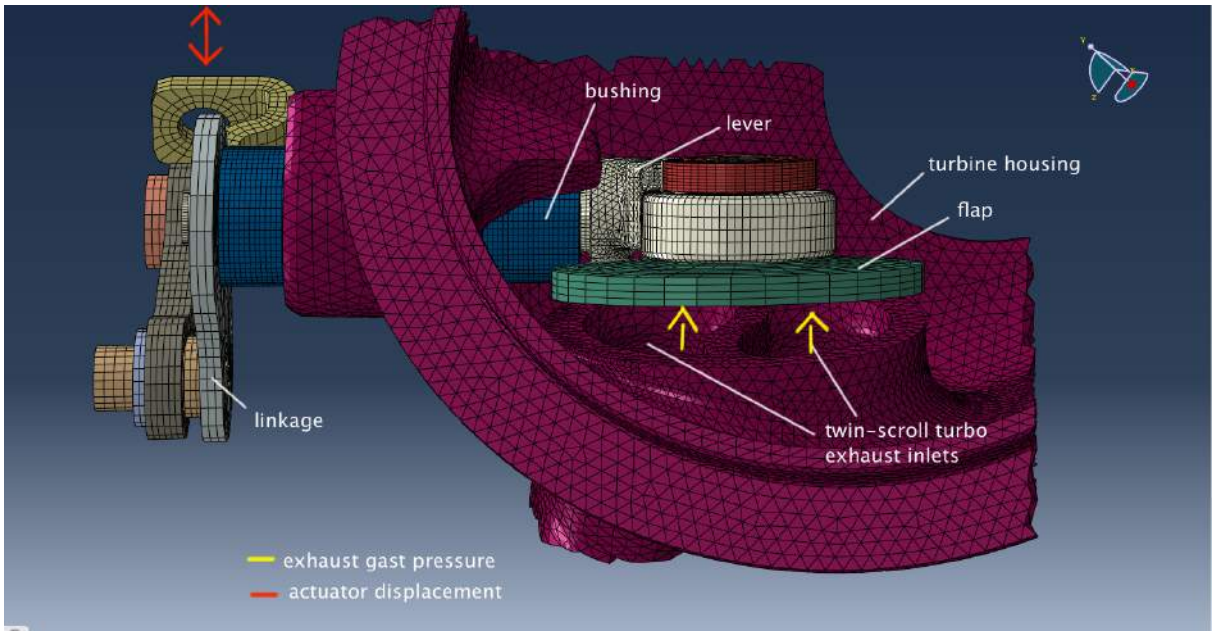


**Figure 3.1:** Wastegate assembly with most important parts

As shown in Fig. 3.1, the primary components of the wastegate are:

1. the exhaust gas inlets, where the exhaust gases are transferred to the turbocharger's turbine. For a twin-scroll turbocharger, the exhaust gases are supplied in alternating pulsations from the two channels.
2. The wastegate flap, whose opening angle determines the amount of exhaust gas transmitted to the turbine
3. the wastegate lever, which serves to lift or lower the flap according to the requested amount of boost pressure
4. the wastegate bushing, which serves as a casing for the lever
5. the linkage system that transmits the displacement of the control actuator to the lever (a detailed description is omitted here)

The wastegate is evidently a complicated system consisting of many individual subcomponents. The dynamic gas pressure loads and actuator displacements act on opposite ends of the device and are transferred through the entire assembly. For example, the amplitudes of the gas pressure loads acting on the flap over approximately two crankshaft rotations (720°) are shown in Fig. 3.2:

One can imagine that variations in the external loading (such as the gas pressure in Fig. 3.2) will induce a relative motion between all parts shown in Fig. 3.1 that are not rigidly attached to each other. As explained in Ch. 2.4, this relative motion can induce different wear mechanisms that lead to a loss of material from the contact surfaces. The most pronounced wear damage occurs in the following interfaces:

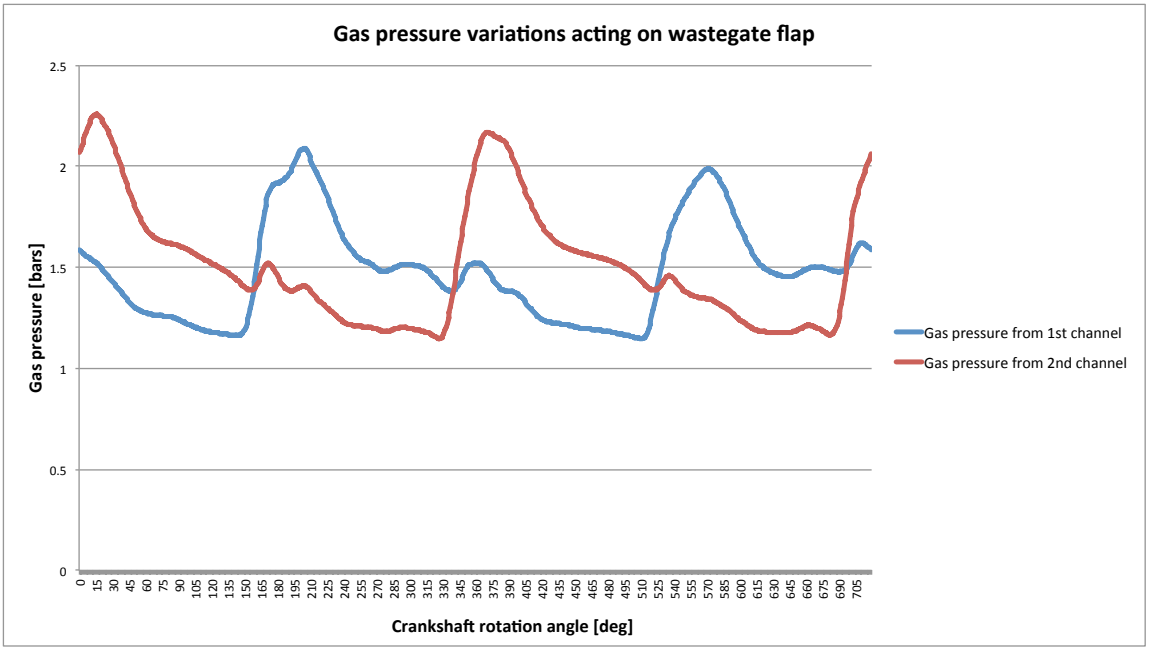


**Figure 3.2:** Gaspressure loads acting on the wastegate flap from the two channels of a twin-scroll turbocharger (engine torque = 215 Nm, engine speed = 5980 rpm)

1. The interface between the flap and the lever, primarily due to repeated collisions
2. The interface between the lever and the bushing due to relative sliding.

While impact wear damage in the contact surfaces of flap and lever is quite noticeable, it is less severe compared to the sliding wear damage that occurs between lever and bushing. In fact, wear between lever and bushing is a critical damage mode that tends to disrupt the wastegate's motion such that the control actuator often has to accommodate the change in geometry by adjusting its position to achieve the required flap opening angles.
In 2011 Meyer published an extensive empirical study on wear in the wastegate's lever/bushing interface and was able to derive wear characteristics for different material pairings. Figs. 3.3 and 3.4 show some of the wear profiles documented in Meyer's dissertation:

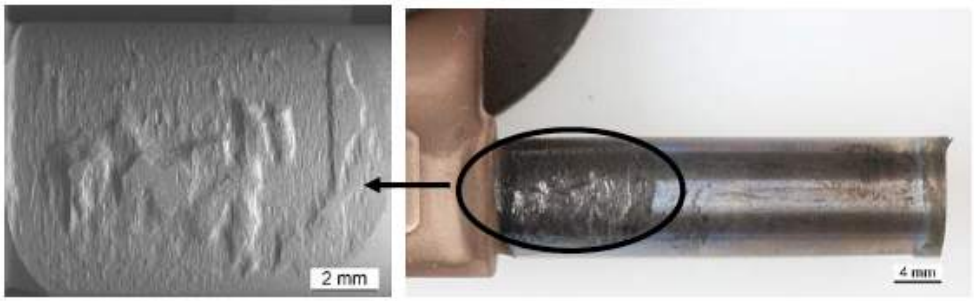


**Figure 3.3:** Wear in lever/bushing interface showing pronounced adhesion [3]

As shown in Figs. 3.3 and 3.4, the wear profile generated by relative sliding can be rough

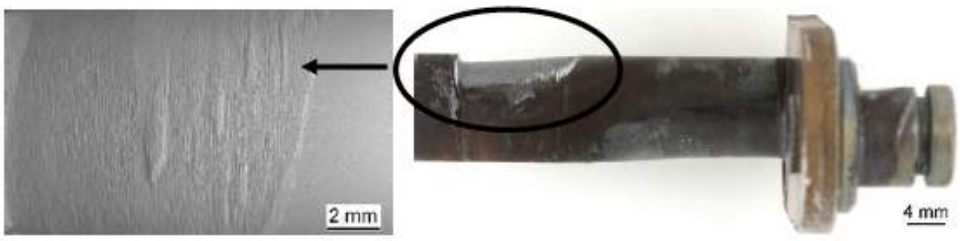


**Figure 3.4:** Wear in lever/bushing interface leads to a significant change in the lever's geometry [3]

or smooth, depending on the dominant wear mechanisms that were effective in the process. For example, the rough profile in Fig. 3.3 is characteristic of adhesive wear as one observes distinct pits in the surface where patches of the lever's material was spontaneously removed. Fig. 3.4, on the other hand, shows a relatively smooth profile that was most likely a result of extensive abrasive wear. Those findings are certainly very interesting from a material science perspective and might be useful for designers to improve the wear resistance of certain material pairings. However, as discussed in Ch. 2.4 the microscopic topology is simply too intricate for the purposes of the current project, so they will not be addressed in more detail for the remainder of this report.
The result that is of interest for an FE wear simulation is the overall change in geometry (or the shape of the part) that is produced by the primary external load cases. Fig. 3.5 shows the critical relative motion that leads to wear in the lever/bushing interface in more detail:

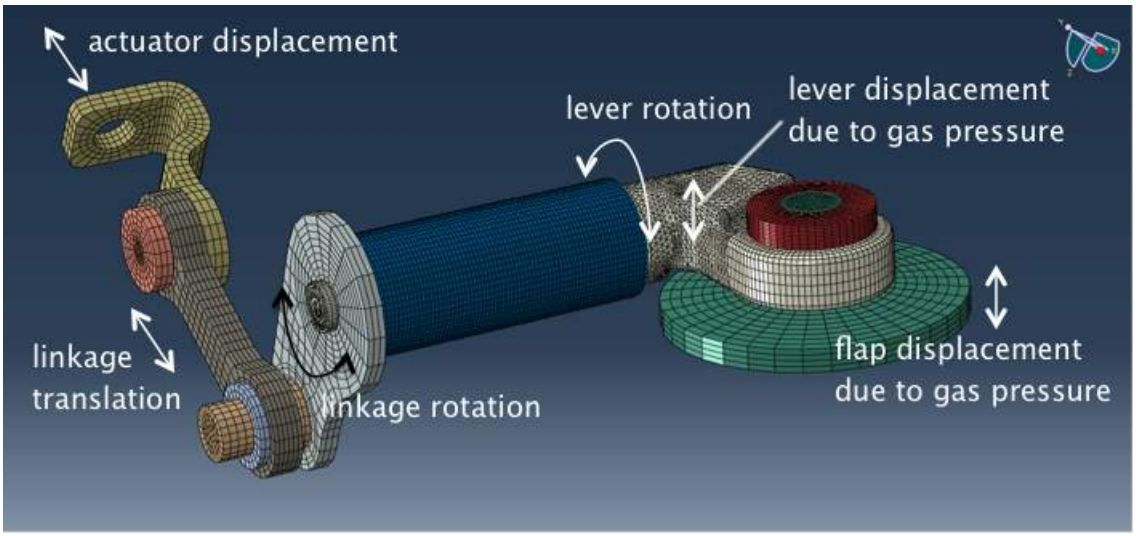


**Figure 3.5:** Relative motion that leads to wear in the lever/bushing interface

One observes that the actuator displacement induces a translational motion in the linkage, which is then transformed into a rotational displacement in the lever such that the opening angle of the wastegate flap can be adjusted. In addition, the gaspulsation loads acting on the bottom surface of the flap will push the lever against the bushing such that the lever is tilted counter-clockwise until it is in contact with the bushing's interior surface at the ends of the interface.

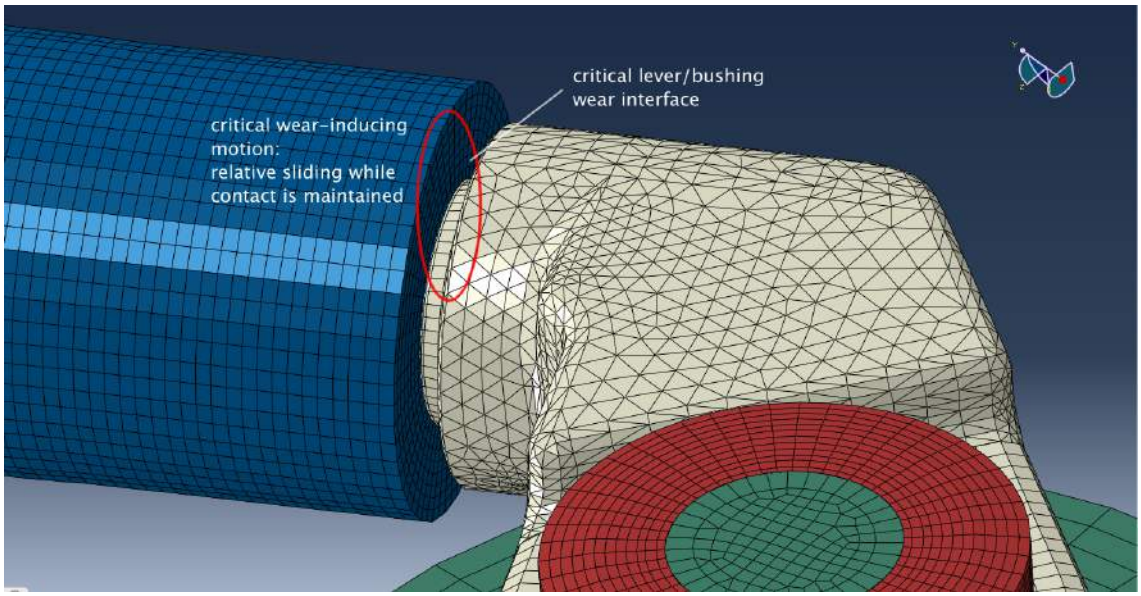


**Figure 3.6:** Critical lever/bushing interface

The sketch in Fig. 3.7 (obtained from Meyer's paper) illustrates the contact conditions more clearly:

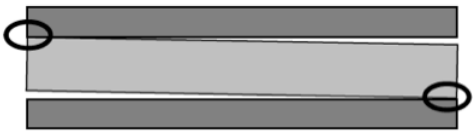

**Figure 3.7:** Cross-section of lever/bushing assembly shows contact at both ends [3]

By observing Figs. 3.5 and 3.7 one realizes that wear in the lever/bushing interface is the result of the combined effect of the gas pressure and the actuator displacement. The gas pressure acts on the flap to create contact between the two parts and the actuator displacement induces the relative sliding. In fact, for the wastegate of a BMW B48 turbocharger model it was found that lever and bushing are in constant contact at the engine operating point that generates the most severe wear damage [2] (this happens to be the engine load that produced the gas pressure variations shown in Fig. 3.2, i.e. at a 215 Nm torque and 5980 rpm engine speed).
The goal of this thesis project is therefore to derive a simulation method that is able to reproduce a similar change in geometry as shown in Fig. 3.4. Since it is not possible to perform extensive testing in order to obtain wear coefficients and other relevant inputs, the results obtained in this study can only be assessed in a relative or qualitative sense, meaning that it will not be possible to directly compare or validate the simulations with experimental data. Conclusively, the essential outcome of this project is to demonstrate that the derived simulation method has the potential to generate the expected geometric changes due to wear if the correct inputs are provided. A detailed investigation to determine those inputs and to adjust them to the simulation is left to future projects.
Using existing gaspressure and actuator displacement boundary conditions, Wibmer et al.

have performed a simulation of the dynamic motion of the entire wastegate system which can be used to derive essential inputs for the interaction between lever and bushing [2]. Certainly, the most obvious approach is to use the same dynamic simulation to generate the necessary contact outputs for the wear simulation. However, the time period that can be reasonably covered by the dynamic simulation proved to be much too short for a conventional wear simulation (as will be explained in more detail in Ch. 4.5 and 6). In order to produce appreciable wear damage with a manageable computation time it is therefore also important that the derived wear simulation routine contains some method to expedite the progress of damage.

## 3.2 Summary

This chapter introduced the individual components of the wastegate and the most critical load cases that induce wear damage in their interfaces. It was found that the interface between the wastegate's lever and bushing is especially susceptible to sliding wear because it is subjected to a contact pressure and a simultaneous sliding motion. The discussion in this chapter led to the conclusion that the simulation routine developed in the thesis project should be able to qualitatively reproduce the wear damage found in the lever/bushing interface. If this outcome is accomplished, it should in theory be possible to also quantitatively predict or estimate wear in the same interface once applicable wear coefficients become available.
The next chapter will thus present the most essential work done in this thesis project.

# Chapter 4

# Wear Simulation Methodology

This chapter presents all methods and techniques that make it possible to carry out a wear simulation that is appropriate for the properties of the wastegate system. Starting with basic considerations for achieving an accurate representation of geometric changes as a result of sliding wear, the discussion then moves on to the derivation of a global simulation routine for dynamic systems, and concludes with the proposal for a simple wear extrapolation scheme as a means to accelerate the simulation.
As shown in Ch. 2.4, Abaqus' integrated ALE adaptive meshing capability is the most efficient and accessible wear simulation technique that provides a reliable mathematical foundation for geometric modifications and mesh adjustments. ALE adaptive meshing in combination with the UMESHMOTION subroutine is therefore used as the starting point for all additional features. Therefore, all methods discussed in this chapter serve to augment the basic adaptive meshing technique such that it can be applied (in theory) to more complicated systems such as the wastegate device.
But before simulation routines are discussed in more detail, it is first necessary to establish an equation that describes the magnitude of nodal displacements based on an analytic model for sliding wear. The following section shows the derivation of the node displacement equation.

## 4.1 Archard's wear model applied to Finite Element simulations

Archard's simple model for sliding wear has been introduced in Ch. 2, where it was also shown that it is used for most Finite Element wear simulations found in literature. As explained, this model is very favorable for FE applications due to its simplicity and its dependence on contact solution variables rather than micro-mechanical parameters. The most recognizable form of Archard's equation is shown in Eq. 4.1:

$$W = K\frac{P}{H}s \tag{4.1}$$

where $W$ is the volume of lost material in [$m^3$] (the "wear volume"), $P$ is the normal load in [N] acting on the contact surface, $H$ the hardness of the softer material (most commonly in

units of the Vickers hardness), $s$ is the relative sliding distance in [m], and $K$ is a dimensional constant that is usually determined by experiment.
In most cases, the hardness of a material cannot be determined in a straight-forward manner. But since $K$ is in any case a proportionality constant that is found through curve-fitting, it is reasonable to simply incorporate $H$ in $K$, such that the wear volume is linearly related to the product of the normal force and the relative sliding distance between two contact surfaces.
For FE simulations, Archard's model must be transformed to describe the one-dimensional displacement of the contact surface, i.e. the displacement of nodes on the contact surface. This can be accomplished by dividing both sides of Eq. 4.1 by the contact area $A$. Though this is not exactly accurate for three-dimensional bodies whose top and base surfaces are not parallel (as compared to e.g. cuboids, cylinders or prisms), on a small scale - that is if the mesh size is sufficiently small - it is valid to approximate the lost volume as the product of the contact area and the *wear depth*. The nodes on the contact surface must therefore be moved by an amount equal to the wear depth to replicate said volume loss. This approximation is used in most wear simulation related studies found in literature. However, to keep errors limited it is necessary to ensure that the mesh is sufficiently refined such that the contact area is not too irregular.
Dividing both sides of Eq. 4.1 by $A$, one observes that while the wear volume becomes the wear depth, normal force $P$ becomes normal pressure $p$ acting on the contact surface. Eq. 4.2 shows the form of Archard's model that is useful to an FE simulation:

$$h = K_H \frac{P}{A} s = K_H p s \tag{4.2}$$

where $h$ is the wear depth in [m], $p$ in $[\frac{N}{m^2}]$ is the contact pressure, and $K_H$ is Archard's wear coefficient corrected by material hardness $H$. From Eq. 4.2 one also observes that the unit of $K_H$ is $[\frac{m^3}{N}]$, which may be interpreted as the volume of lost material per newton.
The contact pressure and relative sliding distance are calculated as nodal quantities in Abaqus' output database and can be obtained during the analysis by calling utility routines GETVRN or GETVRMAVGATNODE from UMESHMOTION. Contact pressure CPRESS is defined in Abaqus as the current magnitude at each increment, and the relative slip in two directions, CSLIP1 and CSLIP2, are defined as the total relative slip accumulated up to the current increment. The wear depth incurred in each increment i should thus be calculated using the average contact pressure between two increments and the incremental vector magnitude in the two slipping directions:

$$\Delta h_i = \frac{CPRESS_{i-1} + CPRESS_i}{2} \sqrt{(CSLIP1_i - CSLIP1_{i-1})^2 + (CSLIP2_i - CSLIP2_{i-1})^2} \tag{4.3}$$

This basic equation shall be used in all wear simulations to calculate the displacement of contact nodes due to sliding wear. If additional wear models are used - such as models for impact or fretting wear - the methods discussed in the following are still valid. Essentially, it is a matter of obtaining and combining relevant nodal output variables specified in the corresponding wear equation.
In Abaqus, surfaces are discretized with either nodes or elements. If a node-based surfaces is

defined, Abaqus will use the node-to-surface formulation to calculate contact variables, while element-based surfaces allow for both node-to-surface as well as surface-to-surface contact formulations.

## 4.2 Wear Simulation Techniques

The methods discussed in the following have been considered with regard to their feasibility, flexibility and general validity. In order to understand the motivation behind the discussed procedures, the main problems or difficulties are first explained briefly.
In Ch. 2.4 it was shown that most studies on wear simulations so far have considered relatively simple systems where simple geometries are subjected to quasi-static load cases. For those systems, such as the cylinder sliding on a steel plate investigated by Martinez et. al [9] or even the adapted pad-to-rotor model studied by Soderberg and Andersson [6], UMESHMOTION and ALE adaptive meshing can be applied directly with very few adjustments. The wastegate system, on the other hand, consists of many parts with arbitrary orientations in space that are in addition expected to change with time. As will be explained shortly, this complicates the definition of an appropriate direction for node shift as compared to a simple part with constant surface orientations.
Also, the components in the wastegate are subjected to irregular, high-frequency gas pressure loads that cannot be reasonably converted to a quasi-static load case. While it might be very difficult to relate the original dynamic load case of the wastegate to the wear simulation (due to reasons explained later in Ch. 6), the goal here is to derive a procedure that is potentially applicable to any dynamic system, such that it can still be used once suitable adjustments are found.
As mentioned, Abaqus does not support ALE adaptive meshing for dynamic analyses. However, if one intends to ensure that wear occurs due to the exact solution-dependent variables as the ones generated by the actual dynamic load case, it is necessary to include the original dynamic simulation in the routine. An approximation of the dynamic loading with a quasi-static load might in theory be possible, but would in itself require extensive investigation. It is in particular very difficult to predict the impact of the changes in geometry on the progression of the stress state and kinematics of the system, and to then transform it into some quasi-static condition. Conclusively, it perhaps simpler to just use the original dynamic analysis to obtain the correct outputs.
The two central problems that have to be solved can thus be summarized as follows:

1. Devise a way to accurately replicate the changes in geometry as a result of material loss from surfaces with arbitrary spatial orientations.

2. Find a method to apply wear-related effects from the original dynamic simulation in a static step that allows for ALE adaptive meshing, preferably in such a way where the accuracy is not inherently limited and can be adjusted according to the needs of a specific system. To clarify: Since geometric modifications cannot be performed simultaneous to the dynamic analysis, it is necessary to obtain wear-related outputs and apply those separately with a certain frequency. As will be shown in Ch. 5.2, if the resolution is very high, i.e. if geometric modification is performed many times throughout the analysis, the

simulation should approximate the case where mesh movement is performed in parallel to the actual load case.

The next section will first address the challenge of finding a proper direction for node shift.

## 4.3 Geometric Part Modification with UMESHMOTION

As dicussed in Ch. 2.4, while ALE adaptive meshing is used in many different applications to maintain a good mesh quality, the change in part geometry is accomplished in Abaqus/-Standard by imposing adaptive mesh constraints. With UMESHOMTION the user is able to describe the movement of nodes in the adaptive mesh constraint region based on solution dependent variables. In wear simulations, the part geometry is changed by moving appropriate surface nodes a certain distance away from the contact surface based on a calculated wear depth. According to this displacement, the mesh in the rest of the part domain is then adjusted according to the algorithm of ALE adaptive meshing such that severe element distortions are prevented. The essential function of UMESHMOTION is therefore to prescribe the magnitude and direction of the surface nodes' displacement vector.
If a user-defined adaptive mesh constraint is activated for a given surface, UMESHOMTION will access each node individually and move it by a certain distance according to the user's specifications. The magnitude of the displacement can be defined in many different ways. If one intends to simply apply the geometric modification in the same analysis step, the procedure is usually rather straight-forward: In each increment UMESHMOTION will call integrated utility routines (GETVRN, GETNODETOELEMCONN and GETVRMAVGATNODE) that will deliver relevant solution variables for the particular node under consideration, and using a certain analytic equation the amount of displacement is then calculated in UMESHMOTION itself and applied to the vector ULOCAL. The ULOCAL vector is defined in the local coordinate system ALOCAL, which Abaqus calculates internally for each individual node.
A UMESHMOTION file with a set-up as shown in Fig. 4.1 had been developed previously by Schmidt [23] at BMW and was consulted as a basis for the the UMESHMOTION code for this project:

As one observes in Fig. 4.1, an important parameter that needs to be defined in UMESHMOTION is the direction of node displacement, i.e. the direction that results in the most reasonable change in geometry. Defining this so-called “wear direction" is not trivial, as it involves approximating the loss of material from a continuous surface of a real part through rearranging nodes that make up a discretized surface. Problems occur primarily at macroscopic surface discontinuities such as edges and corners. Consider for instance the change in geometry of a real part as a result of relative sliding. Fig. 4.2 shows the 2-dimensional cross-section of a solid cube.

As shown in Fig. 4.2, the surface that is produced by sliding the cube against a flat plate is expected to conform to the orientation of the flat plate (if sliding wear is considered). This should be the logical outcome in almost all cases if contact between plate and cube is maintained by an external load (indicated as contact pressure in Fig. 4.2). If the plate's surface is parallel to the cube's lower surface, as shown in the images at the bottom of of Fig. 4.2, it is easy to see that material loss should occur perpendicular to that interface.

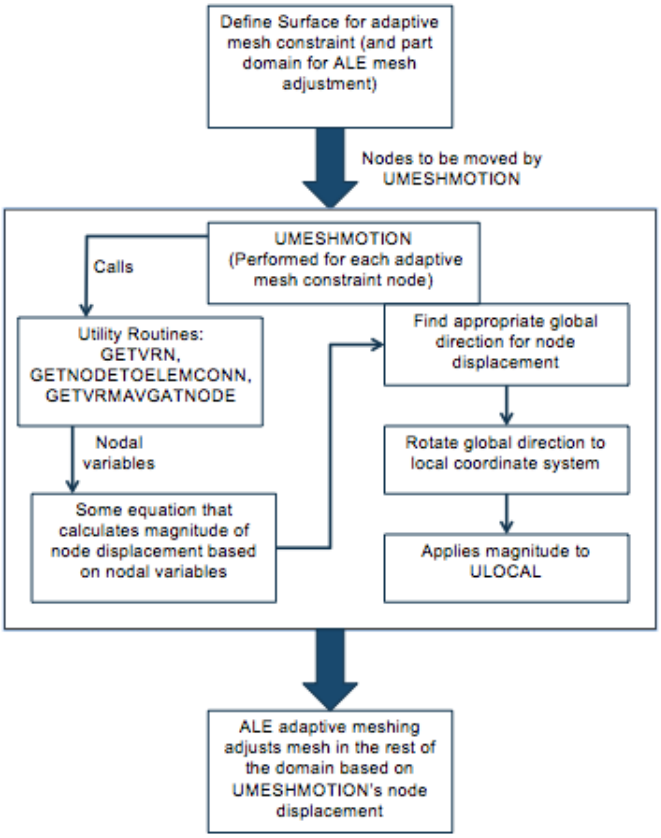


**Figure 4.1:** UMESHMOTION common structure

The situation is less obvious if the plate is instead acting on the edge of the cube, as shown in the images at the top of Fig. 4.2. On a microscopic level, the edge of the real cube will first deform slightly such that a microscopic contact surface is established, and if the plate's initial angle is maintained, this contact surface will become larger over time and always assume the same orientation as the plate. In theory, the "wear direction" should therefore be perpendicular to the *local normal* of the deformed contact surface. This concept works in reality because real parts do not have actual discontinuities at their edges and corners. There will always be some small surface that first deforms to fit with the other contact area.
In an FE model, on the other hand, the edge of the cube is be discretized with a finite number of elements that give rise to actual discontinuities. If the edges and corners of the model are not meshed with an extremely large number of elements such that rounded, continuous regions are formed, there will be no microscopic surface that conforms to the plate to provide a "natural" local normal. As a result, Abaqus will not be able to distinguish contact from different angles, as shown in Fig. 4.3.

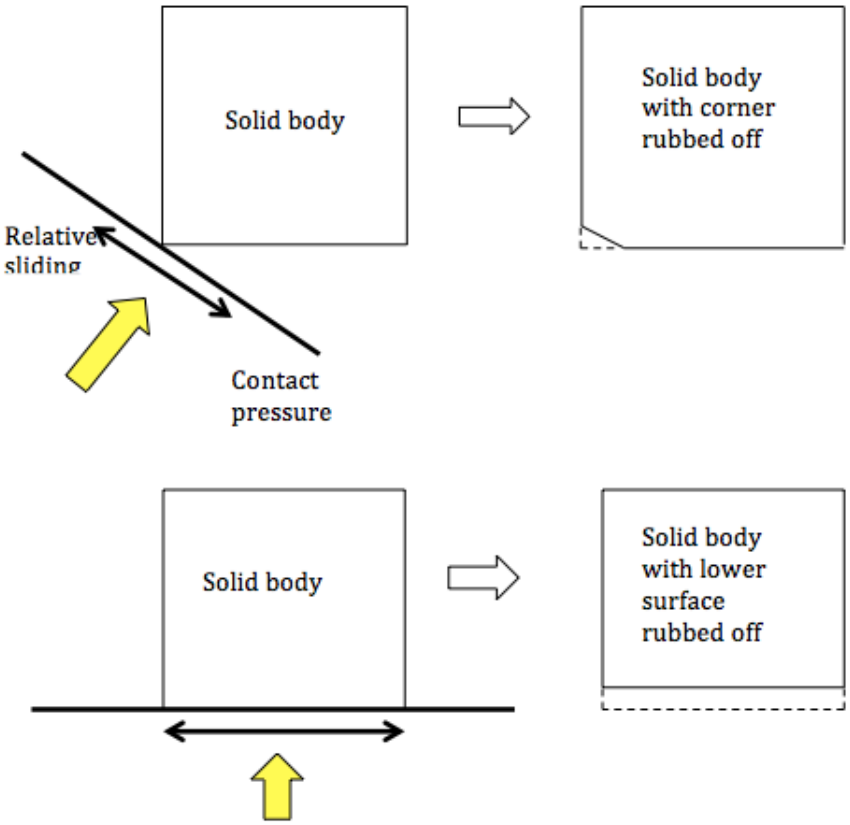


**Figure 4.2:** Change in geometry of a real solid cube (cross-section shown here) due to relative sliding against a plate at different angles

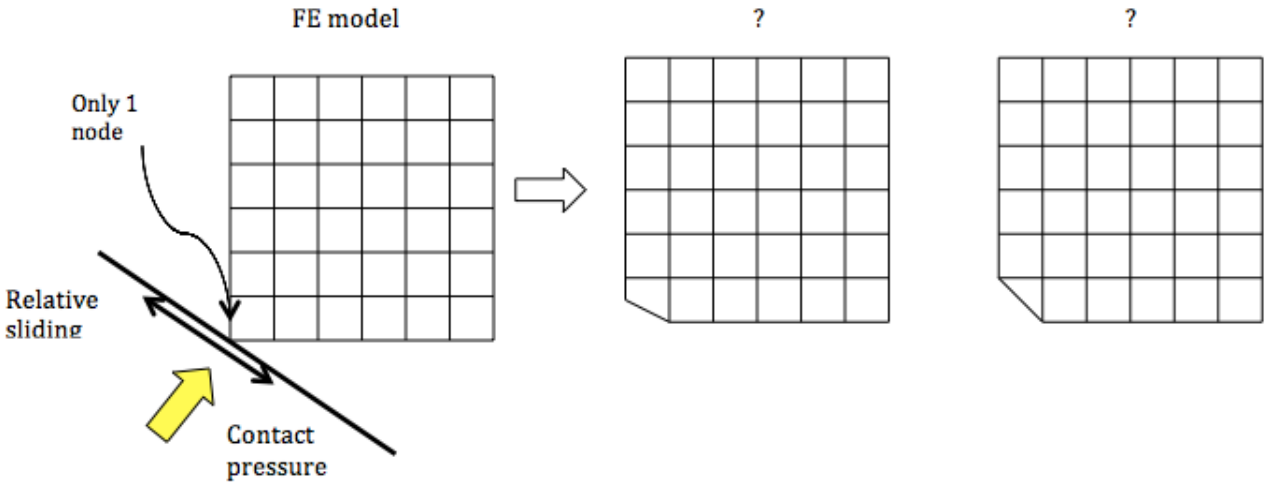


**Figure 4.3:** Wear direction is unclear in FE models if mesh is not extremely fine

One observes that if the mesh is not extremely fine, it will not be possible for Abaqus to find the proper wear direction by default. However, it is in most cases not reasonable or

indeed not possible to introduce a large number of elements at edges and corners since the dimension of the rounded discontinuities are usually extremely small. The part would have to be meshed with an extremely large number of elements in order to avoid sharp transitions that, on a big scale, are actually sharp transitions. One observes that in most cases it is much more reasonable to define the wear direction at corners and edges according to some specific prescription. The following section will therefore discuss two solutions to this problem.

### 4.3.1 Definition of node shift directions

Abaqus conveniently provides a local coordinate system called ALOCAL at each node that is calculated based on the average of the local element surface orientations. The first and second directions are parallel to the averaged element face orientation adjacent a particular node, and the third direction is normal to the first two directions and points outward from the part surface. Fig. 4.4 shows typical directions of ALOCAL defined at different nodes in the FE model of a cube.

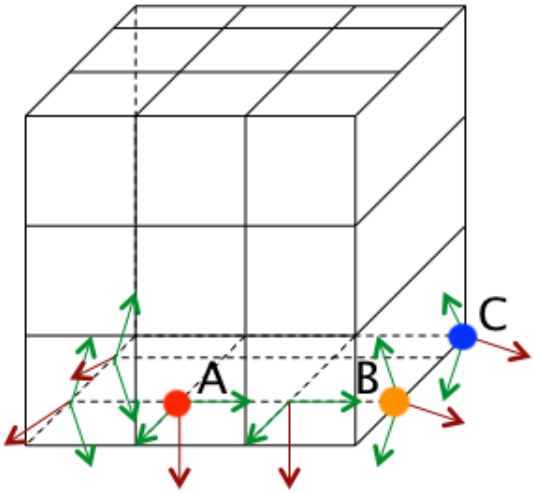


**Figure 4.4:** Local coordinate system ALOCAL as defined by Abaqus

The only way to define the displacement at each node in UMESHMOTION is to specify appropriate values for the three components of the vector ULOCAL, which is defined in the respective ALOCAL system. One observes that for nodes that are not located between sudden changes in the surface orientation (i.e. nodes that are not located at edges or corners, such as node A in Fig. 4.4), it is reasonable to simply apply the calculated node shift in the opposite direction as ULOCAL(3). For example, if one imagines the cube in Fig. 4.4 to be sliding over a plane on its bottom surface, a node shift perpendicular to the contact surface would reasonably approximate the change in geometry for *surface nodes* such as node A. In this case, the magnitude of displacement is simply equal to -ULOCAL(3).
However, for edge and corner nodes such as nodes B or C, a shift only in the ULOCAL(3) direction would not reproduce the geometry that is likely going to result from sliding the real cube on its bottom surface. Figs. 4.5 and 4.6 shows the a reasonable versus an unreasonable change in node shift at edge and corner nodes.

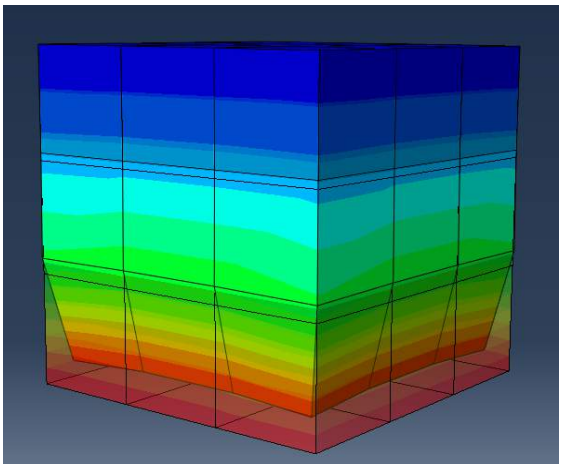

**Figure 4.5:** Incorrect node displacement for material loss from bottom surface

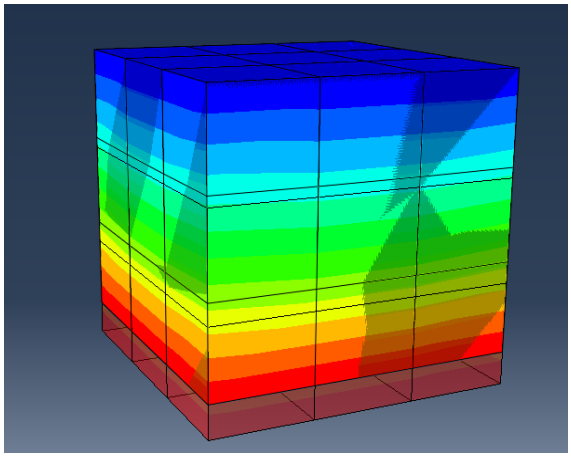

**Figure 4.6:** Correct node displacement for material loss from bottom surface

One observes that if the part had an extremely fine mesh, the sharp corners and edges of the cube would be approximated with many small elements, and in that case ULOCAL(3) would indeed still be appropriate. However, this is usually not the case since most geometries do have abrupt transitions in their outer surfaces, and in order to use ULOCAL(3) as the wear direction in all cases one would have to introduce small fillets to all edges and corners. In the actual physical part those fillets might be so small that they cannot be distinguished visually. An extremely fine mesh would also significantly increase the computational effort.
As discussed, a far better option is to redefine the wear direction at edge and corner nodes using some reasonable criteria. There are two ways to do that:

1. Define the wear direction at edges and corners to be in the *normal direction to the contact surface*
2. Define the wear direction at edges and corners to follow the outer surface of the part, i.e. the part's shape itself.

In the following both techniques shall be discussed briefly.

### Wear direction based on local normal

The idea for the first method is quite simple: by excluding surfaces that are not expected to be in contact, the local normal at edge and corner nodes are calculated only based on the orientation of a single contact surface. For example, the correct deformed geometry Fig. 4.6 was produced by pre-determining a local normal at the edge and corner nodes which is perpendicular only to the bottom surface where the adaptive mesh constraint is activated. However, it turns out that the process of finding this normal vector is less straight-forward. UMESHOMTION will still take surfaces into account where adaptive mesh constraints are not defined (see Fig. 4.5), and the only way to move any node is by defining the vector ULOCAL, which is always defined in the ALOCAL system. The most convenient way is to find the appropriate direction in the global coordinate system and then rotate this global vector into

the local system. The transformation itself is not difficult, since the basis vectors of the ALOCAL system are passed into UMESHMOTION for each node at each increment. To rotate from the global to the local system, one can simply use the following matrix multiplication:

$$\underline{n}_L = \underline{\underline{A}}^{-1}\underline{n}_G$$
$$\begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix}_L = \begin{pmatrix} b_{1,1} & b_{2,1} & b_{3,1} \\ b_{1,2} & b_{2,2} & b_{3,2} \\ b_{1,3} & b_{2,3} & b_{3,3} \end{pmatrix}^{-1} \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix}_G \tag{4.4}$$

where $\underline{n}_G$ and $\underline{n}_L$ are the normal vectors in the local and global coordinate system, respectively, and $\underline{\underline{A}}$ is the transformation matrix of ALOCAL. $bi, j$ are the components of ALOCAL's basis vectors $b_i$, and $n_i$ are the components of the normal vectors in the global (G) and local (L) systems. Since the basis vectors of ALOCAL are orthonormal, the inverse of the ALOCAL matrix is simply its transpose. The same transformation is used in several examples found in Abaqus' manual [22] [24].
To obtain the normal vector in ALOCAL by transformation, one must first find the vector in global coordinates. For a simple, stationary system such as a cube sliding on a flat plate, it is perhaps sufficient to define a constant normal vector (in the global system) as one does not expect the orientation of the contact surface to change significantly over time. After all, the cube does not rotate in space and wear is expected to be almost uniform over the contact surface. However, in general one cannot make those assumptions. The wastegate in particular is a dynamic system where the orientation of any surface can change continuously over a series of simulations. In addition, depending on where contact actually occurs, the extent of wear evolves very unevenly over a single surface. The accumulated change in geometry due to wear itself might significantly change the initial local surface orientation, thus invalidating the initial normal vector. One realizes, therefore, that a reasonable normal vector must be redefined in each time increment at each node in order to minimize inaccuracies. Conceptually, the most accurate normal vector would be defined in terms of the local element surface. In each new increment, the normal vector at an edge or a corner node should be perpendicular to the edges that are part of the contact surface and belong to immediately adjacent elements. Fig. 4.7 illustrates the concept:

Consider the edge node E in Fig. 4.7. To find the appropriate normal vectors at E, one would first need to determine the vectors $e_1$ and $e_2$ along the edges of the two connected elements, as well as the perpendicular, in-plane vector $e_p$. By cross-multiplying $e_p$ with $e_1$ and $e_p$ with $e_2$, one obtains two normal vectors that either point inward or outward from the bottom surface. To find the correct *inward* vector (since the + or - direction of a vector is determined by the order of multiplication which is difficult to determine for arbitrary elements), it is necessary to define an additional vector $s$ (purple) which points from E to a *subsurface node* S (it does not matter if S lies inside the cube or on its outer surface). One then computes the angle between both normal vectors and $s$, and if the angle exceeds 90°, it must be pointing outward and should thus be flipped to the opposite direction. Having determined the correct orientation for both normal vectors, the wear vector $n$ (normalized) is then defined to be in the average direction of the two normal vectors. For the cube in Fig. 4.7 the angle between the two element normals must be extremely small, but for a general part where there might be a larger difference between the orientations of two adjacent element faces, the angle could

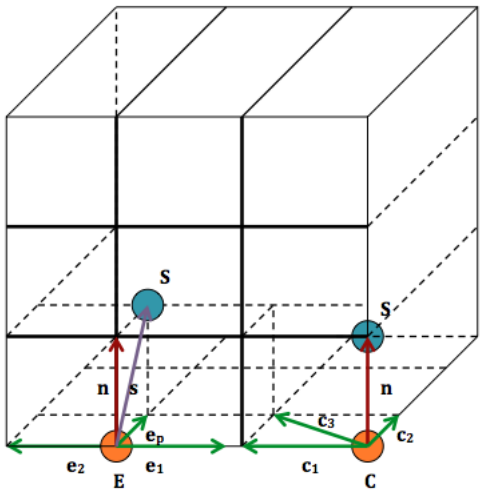


**Figure 4.7:** Local normal vectors (red) calculated based on planar vectors of connected elements

be much larger, and their average direction would therefore be a better approximation of the wear direction.

The procedure for calculating the local normal at corner node C is very similar. Since only one element is connected to such a corner node, the wear direction is appropriately defined by the cross product of any two of the three vectors $c_1$, $c_2$, or $c_3$. In general it is not obvious how to immediately distinguish between edge vectors $c_1$ or $c_2$ and diagonal vector $c_3$ (one could compare their lengths and use the shorter two vectors, which should usually be the edge vectors), but the distinction is not strictly necessary since all vectors lie in the same element face. This approach for calculating the appropriate local normal vector was introduced by Hegadekatted et al. who used it to determine the wear direction for all nodes [7], as they did not use UMESHMOTION which provides the correct node shift direction for all but edge and corner nodes.

To determine any vector in Fig. 4.7, one must first find the relevant node that defines the vector's direction. For example, there must be a general way to find node S and its coordinates, such that vector $s$ can be formed between S and E. For simple, linear brick elements defined by eight nodes, all relevant nodes can be distinguished by performing some simple exclusion operations on the model's basic *element connectivity* (A matrix that defines an element in terms of the nodes connect to it. It can be easily extracted from the model or output database). Having found the appropriate node numbers from the element connectivity information, UMESHMOTION will then call its utility routine GETVRN to obtain all three global components of a certain node. The flow chart in Fig. 4.8 summarizes the detailed steps towards finding the wear direction in ALOCAL.

As one observes, this rather long-winded procedure for finding the wear direction is usually unnecessary since most of the nodes are not located at part boundaries. Edge and corner nodes usually comprise a small proportion of the adaptive constraint region, so the calculations are not expected to have a significant impact on the overall computation time:

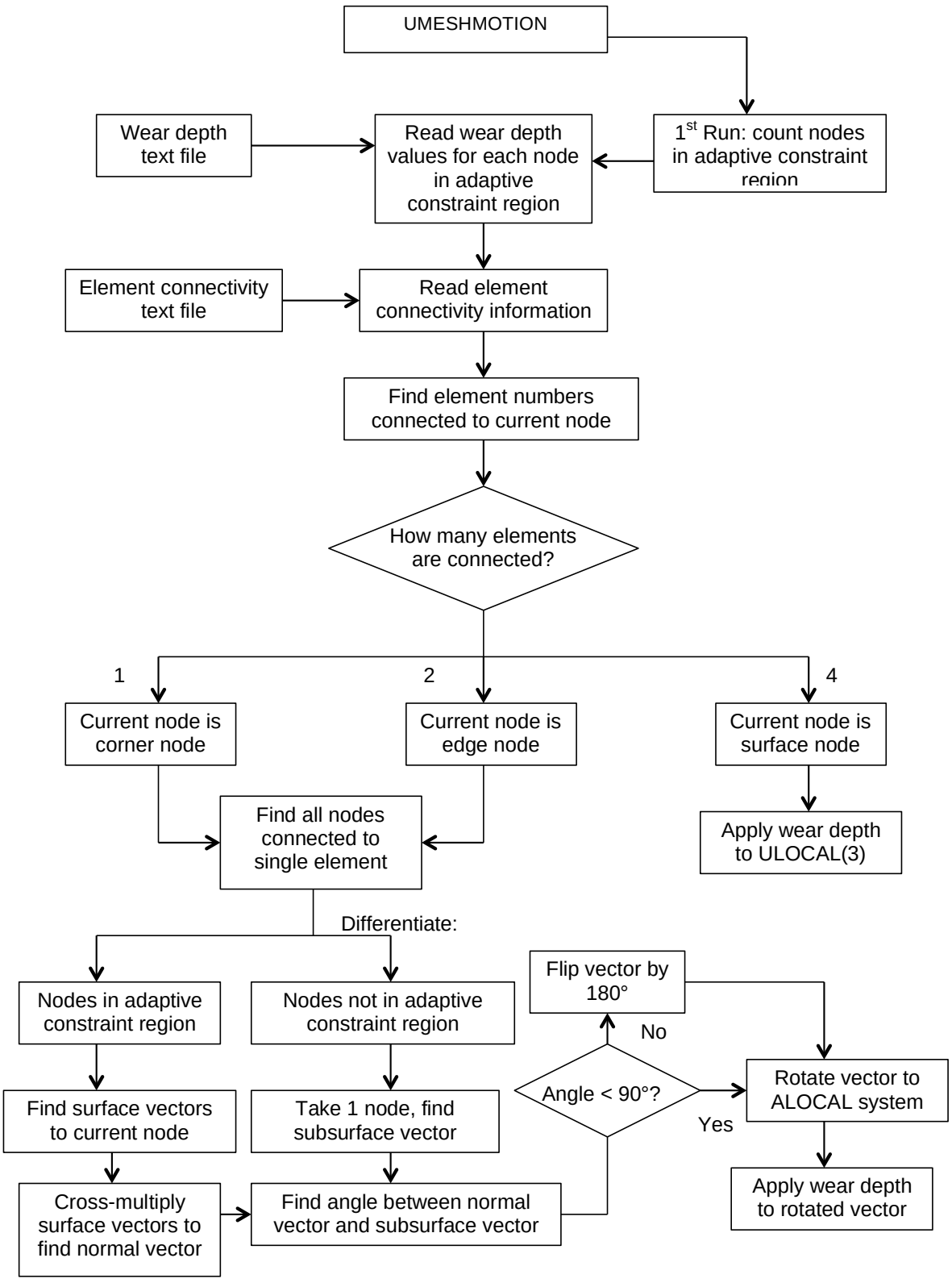


**Figure 4.8:** Detailed flowchart of tasks performed by UMESHMOTION for every node in adaptive constraint region

It must be noted, however, that the algorithm used to find all nodes and vectors as described in Fig. 4.8 only works if the model is defined such that:

1. All elements in the adaptive mesh constraint region are defined by only eight nodes. The required nodes could not be distinguished with such simple methods if e.g. nonlinear elements with more than eight nodes are used. This should not be a severe limitation since in any case ALE adaptive meshing in 3D is only supported for linear elements [19].
2. In the model definition all nodes and elements must have a unique label (i.e. tick " Do not use parts and assemblies in input files" in the model attributes settings). This is because an external element connectivity text file must be generated prior to applying the wear simulation, and the node and element labels read by UMESHMOTION from that file must agree with Abaqus' internal labelling that is accessed by utility GETVRN.
3. In the part geometry itself, the angle between the adaptive constraint surface (i.e. the contact surface) and its adjacent surfaces should not deviate too much from 90°. Otherwise the normal vector found by cross-multiplying vectors on the contact surface will not provide the most reasonable wear direction.

While constraints 1. and 2. are quite self-explanatory, item 3. might require some clarification as it relates to the second way of defining the wear direction.

### Wear direction based on part geometry

Fig. 4.9 illustrates the problem that could arise if the wear direction is always assumed to be perpendicular to the contact surface:

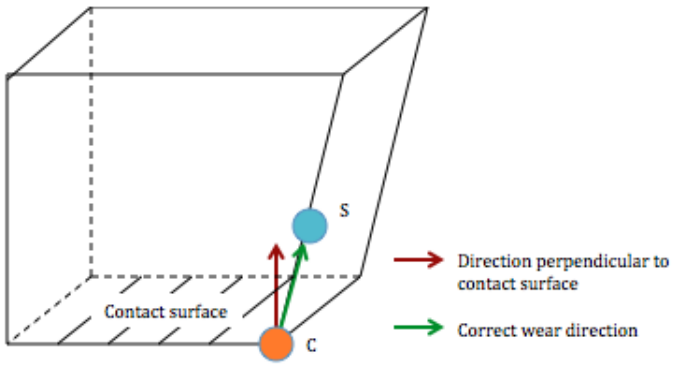


**Figure 4.9:** Proper wear direction in case of oblique angle in geometry

Fig. 4.9 shows a block with a slanted side where edge and corner nodes are connected to surfaces that do not form a right angle. Consider again uniform sliding wear at the bottom surface: For an accurate representation of the change in geometry, node C should in this case move in the direction of the outer edge instead of perpendicular to the contact surface. This means that a node on the contact surface's periphery should preferably move towards the node that is located immediately above it on an adjacent outer surface. For node C, the

appropriate “wear direction node" should thus be node S. Moving node C perpendicular to the contact surface would in this case change the angle of the edge that is connected to C and thus fail to capture the most realistic change in geometry.
This approach is actually easier to implement in practice: To define the wear direction, one only needs to find the location of the “wear direction node" S, so it is no longer necessary to define multiple vectors and to cross-multiply them. The flowchart in Fig. 4.10 shows all steps that have to be included in UMESHMOTION if this method is used (the first part is identical to Fig. 4.8 and is thus omitted):

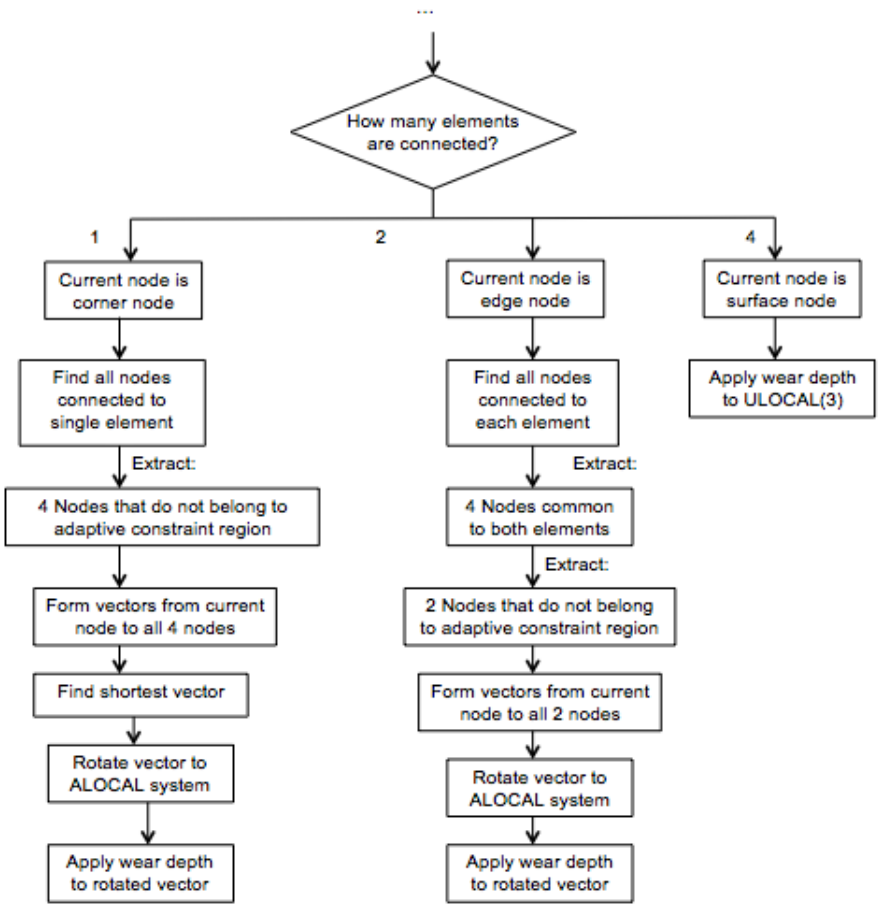


**Figure 4.10:** Proper wear direction in case of oblique angle in geometry

As shown in Fig. 4.10, this method relies on finding the correct S node by evaluating the lengths of vectors within an element. Since S is always located immediately above or below the node that is currently considered by UMESHMOTION, the vector to S should be the shortest of all vectors that are formed in each case. The concept is illustrated in Figs. 4.11 and 4.12:

The blue and green vectors in Figs. 4.11 and 4.12 are vectors that are formed with all nodes that can be narrowed down using the element connectivity information and the node numbers in the adaptive constraint region. As one observes, the relevant node S is in both cases associated with the shortest vector.
The downside to this method is that it relies on the general shape of a brick element. Should

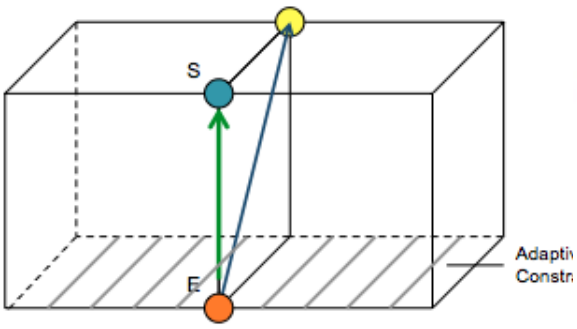


**Figure 4.11:** Wear direction at edge nodes (green)

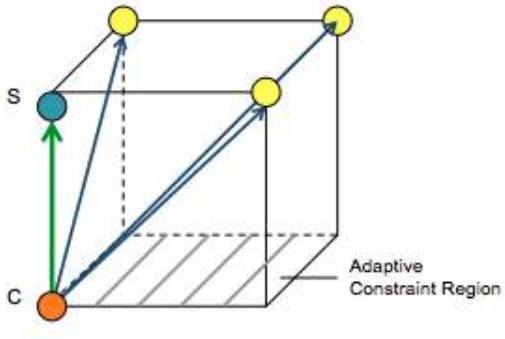


**Figure 4.12:** Wear direction at corner nodes (green)

an element become significantly distorted at some point such that the distance to node S is no longer the shortest, the computed wear direction would consequently be incorrect. However, severe mesh distortions are usually not expected to occur since the purpose of the ALE adaptive meshing technique is to prevent such distortions in the first place. Therefore, this method should be valid in most cases.
The outer surfaces of most parts in the wastegate assembly - such as the lever or the bushing - intersect at right angles, which means that it is appropriate to use the first method. The second method should only be used if a given contact surface forms oblique angles with any adjacent surfaces.
Apart from defining the proper wear directions, there are a few additional aspects that should be considered when applying UMESHMOTION to a system with multiple parts and contact surfaces. Some of the issues that have been encountered are discussed briefly in the following paragraphs.

### 4.3.2 Definition of Adaptive Mesh Constraint Regions

Both methods discussed in the previous section require that a distinction can be made between nodes that belong to the adaptive constraint region (i.e. the contact surface) and those who do not. This implies that any contact is assumed to be effective on the same continuous surface, meaning that one must not include discontinuous planes in the same adaptive mesh constraint region. For example, in Fig. 4.13 a single adaptive mesh constraint must be defined to contain only one of the six outer surfaces of any element. If a contact is expected to occur on two or more adjacent surfaces, it is necessary to define a separate adaptive mesh constraint as well as a separate contact interaction, such that the direction of node displacement can be clearly distinguished between contacts that are expected to change the geometry in different directions. An illustration of this concept is shown in Fig. 4.13.

For a part with adjacent contact surfaces, such as surfaces A and B in Fig. 4.13, one must therefore perform the node displacement separately such that nodes 1 and 2 can move in the correct directions according to the effect of the two distinct contact interactions. If a single adaptive mesh constraint contains two different contact surfaces, edge and corner nodes will move randomly either perpendicular to one or the other surface (in case the first method for finding the wear direction is used).

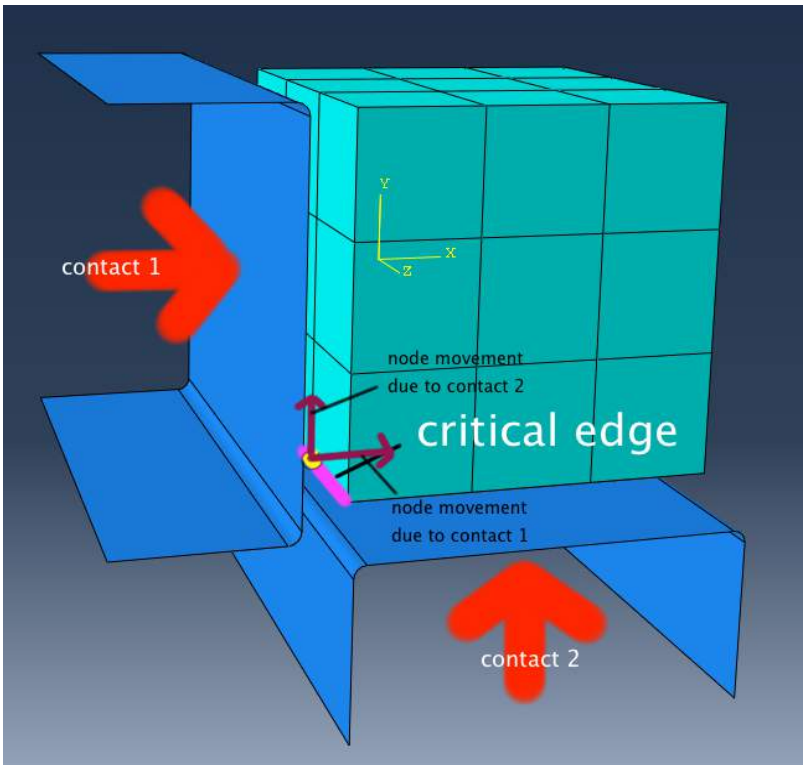


**Figure 4.13:** Contact on adjacent sides requires separate definitions of the adaptive constraint region

### 4.3.3 Wear simulation on both surfaces of a contact

For two surfaces in contact, it is sufficient in many cases to consider material loss from only one of them. As shown in 2.4, most scientific studies investigated some specific properties of the wear simulation (e.g. time incrementation, extrapolation etc.), where the results can be easily compared to experimental data. However, the objective of the company that commissioned this project is to ultimately replicate or predict wear on all parts of the system that are susceptible to damage, which means that the simulation should be set up with as few constraints as possible. At part interfaces of the wastegate system, wear is frequently seen on both parts of an interaction, which means that the UMESHMOTION routine should be able to perform node shift on both surfaces that are involved in a contact.
The main issue to be considered is that in Master/Slave interactions, the relative slip (CSLIP1 and CSLIP2) can only be obtained for nodes on the slave surface. If this contact formulation is selected, it is necessary to define a symmetric contact, meaning that the interaction must be defined twice such that master and slave surfaces are switched in the second interaction. Since solving contact problems is often the bottleneck in terms of computation speed, one can imagine that a symmetric contact will significantly increase the computational effort since each contact interaction now has to be solved twice. This approach is in most cases unproblematic for the user, but is not recommended for very large models (such as the wastegate model).
An easy solution is to use the general contact formulations, for which contact outputs are recorded on both contact surfaces. This general contact formulation was previously only

available for Abaqus/Explicit, but has since been extended to Abaqus/Standard. If one finds that the solutions obtained with the general contact formulation is robust and reliable, it is recommended to use this option if one intends to investigate wear on both contact partners.

### 4.3.4 Adaptive mesh controls and node type definitions

It is important to note that node displacement is not always influenced by adaptive mesh controls. Amongst other things, adaptive mesh controls are a means to modify the criteria by which a certain node in the adaptive mesh domain is recognized as either a surface node, an edge node, or a corner node. This classification is based on a threshold angle between two element faces (details can be found in the Abaqus User Manual [19]), and not on the number of connected elements. Most importantly, the mesh controls will affect the node displacement only if adaptive constraints are not defined by UMESHMOTION. For example, in other applications one might apply a fixed displacement or velocity constraint parameter on a particular surface that does not depend on solution variables (e.g. variables from the contact solution). In that case, if a node is identified as an edge node, it is constrained to only move along that edge. If it is identified as a corner node, it is prevented from moving in any direction.
However, if user-defined adaptive mesh constraints are applied, all nodes in the adaptive constraint region will move in the direction given by UMESHMOTION, regardless of their classification. Therefore, it makes sense to use an appropriate adaptive mesh control on the entire adaptive mesh domain such that one may influence the motion of nodes that lie in the *boundary regions* but do not belong to the *constraint region*. For example, a node at the front edge of the top surface of the cube in Fig. 4.7 can only move horizontally along that edge (if it is recognized as an edge node), but node E at the edge of the bottom surface will still move according to the direction given by UMESHMOTION as it belongs to the user-defined adaptive mesh constraint region).
Adaptive mesh controls have no effect on the definition of ALOCAL. In fact, it is most likely the other way around since the angle between local normals is used for the node type classification. (So there is no inbuilt function in Abaqus to compute the correct wear direction in all cases.)
A more detailed description of how exactly the node types are determined can be found in the Abaqus User Manual [19], and is omitted here because it does not directly affect the magnitude or direction of wear.
Having reviewed some fundamental issues that relate to the basic principles of the wear simulation, one must now derive a valid method to integrate this technique into a global simulation routine. The next section will explain in detail the different approaches that were considered.

## 4.4 Combining Implicit Dynamic Simulations with ALE Adaptive Meshing

As mentioned in the introduction to this chapter, ALE Adaptive Meshing in Abaqus/Standard is currently only available for general static analysis, steady-state transport analysis, coupled pore fluid flow and stress analysis, and coupled temperature-displacement analysis [19]. While

the possibilities of ALE adaptive meshing seem to be more extensively developed in Abaqus/-Explicit, wear simulations are exclusive to Abaqus/Standard.
The wastegate's dynamic motion, as discussed in Ch. 3, was simulated so far with an implicit dynamic analysis in Abaqus/Standard. Due to the time-dependency of the load cases and the importance of the inertia loads, the simulation cannot be reasonably converted to a general static analysis or a steady-state transport analysis. Therefore, the most sensible approach here is to document relevant contact outputs from the dynamic simulation and apply those in a subsequent general static analysis where ALE Adaptive Meshing is possible. Essentially, after completing the dynamic analysis one would extract relevant solution parameters from the output database and compute the accumulated wear depth for each node in a so-called "post-processing step". The wear depth values will then be used by UMESHMOTION in the following static analysis.
Apart from the obvious reason that adaptive meshing cannot be used in a dynamic analysis, there is an additional motivation to separate the calculation of the incurred wear depth from the actual wear simulation. As earlier discussions have implied, clearly visible wear damage that might impair the wastegate's performance develops over a very long period of time. For example, damage in the wastegate's lever/bushing interface investigated by Meyer was generated by driving a car several tens of thousands of kilometers. Similarly, wear damage in the specimens available at BMW formed in engine endurance runs that lasted up to two weeks (which converts to approximately 26000 km driving distance at a reasonable car speed of 80 km/h). Clearly, even if one assumes that the simulation is as fast as real time (meaning that, for example, simulating the system's behavior over 1 second would also take 1 second of computation time), it would still be extremely impractical to run a simulation over several weeks. (In reality though, computing the dynamic behavior of a large model requires much longer than the represented real time, as will be shown later.)
If the wear simulation is therefore performed in the conventional way where the wear depths are directly calculated in UMESHMOTION and applied immediately, it would be not be possible (or at least it would be very difficult) to process the numbers in any meaningful way. As will be shown later, the extrapolation scheme derived for enhancing the calculated wear depths relies on the presence of a post-processing step where the numbers can be evaluated. The following sections will discuss several possibilities to combine the wear simulation with a dynamic analysis. The most viable method is eventually selected for implementation.

### 4.4.1 Alternating dynamic and static steps in the same analysis

The most obvious way to insert an intermediate wear simulation is to define alternating steps in the same analysis, such that each dynamic step is always followed by a general static step. This approach is simple and well-established, and all solution outputs as well as geometric modifications from one step would be automatically propagated to the next step. A simple flowchart in Fig. 4.14 illustrates this concept.

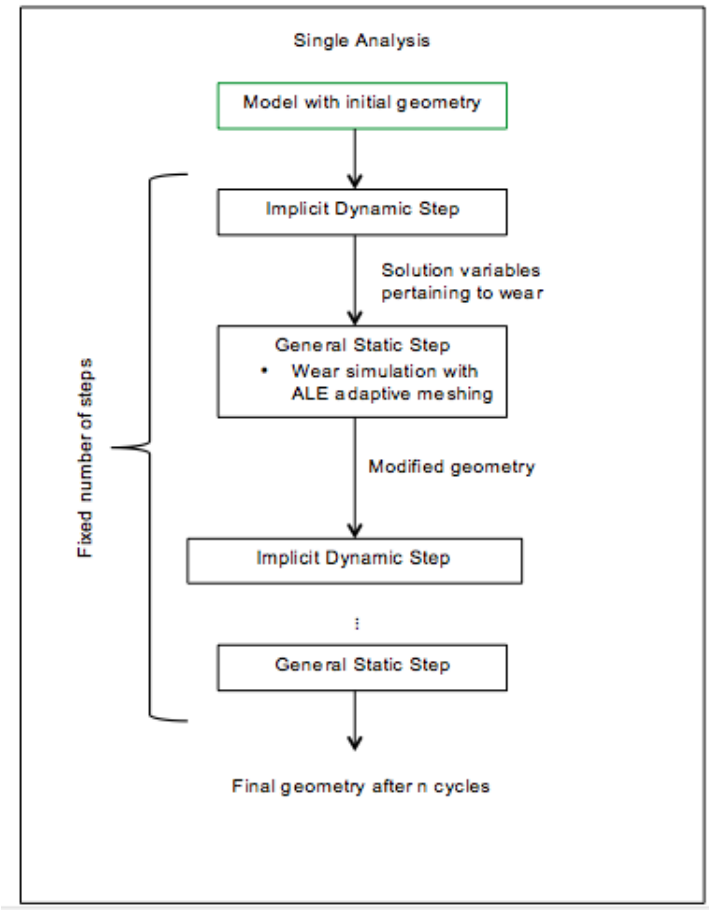


**Figure 4.14:** Single analysis flowchart with a certain number of dynamic/static cycles

Even though this method seems rather straight-forward, there are several reasons why it is inadequate for the wear simulation:

1. With this method, it is only possible to perform a fixed number of steps that are defined in advance, since the complete input file has to be created before a job is submitted. It is not possible to decide on the number of steps based on, for example, the outcome of previous calculations.

2. The transfer of solution variables from the dynamic to the static step is problematic, since it is not possible to read or post-process results from the output database before the the wear simulation is performed. It is usually very easy to request output to be written to the output database, and obtaining those variables in post-processing is equally uncomplicated. If there was a “break" before the static analysis is launched, one could access the odb, perform some operations on solution dependent outputs, and calculate the accumulated wear depth for each node. In the static analysis, UMESHMOTION would read in those values and perform the node displacement accordingly. However, while the analysis is still ongoing, conventional post-processing is not possible. Therefore, it would be necessary to simultaneously create solution outputs in files other than the odb whilst running the dynamic analysis. Two ways to accomplish this

have been considered, but upon close examination both are either not feasible or very impractical:

(a) One could define outputs using the subroutine UVARM such that the wear depth numbers are stored internally and accessed by UMESHMOTION later. UVARM is another Abaqus subroutine that allows the user to define additional solution variables based on outputs from the analysis' results file (the .fil file).
This approach could have been feasible if it was possible for UVARM and UMESHMOTION to call the same utility routines. While UMESHMOTION is able to call utilities GETVRN for variables defined at nodes or GETVRMAVGATNODE to access solution variables defined or averaged at nodes, UVARM can only call GETVR*M* to obtain material point information defined at element integration points [21]. The problem is that contact variables such as contact stress (CSTRESS) and relative slip (CDISP) are only defined as nodal outputs and thus cannot be accessed by GETVRM and UVARM. Therefore, contact outputs cannot be stored or processed during the dynamic analysis with UVARM.

(b) Another possibility would be to request contact data to be written to the .dat file during the dynamic analysis, such that UMESHMOTION can simply read the .dat file in the next step. This option is possible in theory but certainly not very practical. For one, UMESHMOTION can only obtain raw data from the .dat file, i.e. unprocessed values of CSTRESS, CDISP etc. This means that, depending on the model size and number of time increments, UMESHMOTION would have to read in an extremely large amount of data (at least three output variables per node per increment if Archard's equation is used).
In addition, wear depth calculations would also have to be performed in UMESHMOTION itself, which could significantly complicate the coding of the initially simple and straight-forward subroutine. In addition, finding the correct numbers in a complicated file like the .dat file is not trivial and could easily lead to errors: Apart from the specified solution variables, the .dat file contains a copious amount of additional information, which means that UMESHMOTION must be equipped with a sophisticated reading technique to extract the useful outputs. It would be much easier to read a simple text file that only contains one parameter for each node.

Considering the inherent downsides to this method, one realizes it is not worthwhile to pursue it any further. While the technical difficulty of transferring solutions between different steps could be solved with some effort and a more thorough investigation, the fact that only a fixed number of cycles can be performed each time means that this approach is unfit for the purposes of this study. One should thus focus on a fundamentally different procedure.

### 4.4.2 Importing and Editing of Input Files

As shown in the previous section, the single-analysis approach is not very promising and also difficult to implement. The alternative is to consider ways by which results can be transferred between two different analyses. As mentioned previously, it is imperative that several cycles of dynamic and static simulations can be performed in an automated way, such that the outcome

of the previous analysis can be transferred as an input to the next analysis. Independent of the exact method, all relevant operations should thus be integrated in a scripted loop such that they can be performed repeatedly according to some predefined criteria.
The first approach based on this principle is straight-forward and easy to implement, even though it has a few drawbacks that will be discussed later. The key method here is to simply change the initial coordinates of each node in the input file such that the nodes' spatial positions are redefined to be the the coordinates in the final configuration of the previous analysis. Since neither the dynamic nor the static analysis will add or subtract nodes from the original mesh, this process should be quite robust in most cases. Fig. 4.15 illustrates the working principle of this approach, which will be henceforth referred to as the *input file method*.

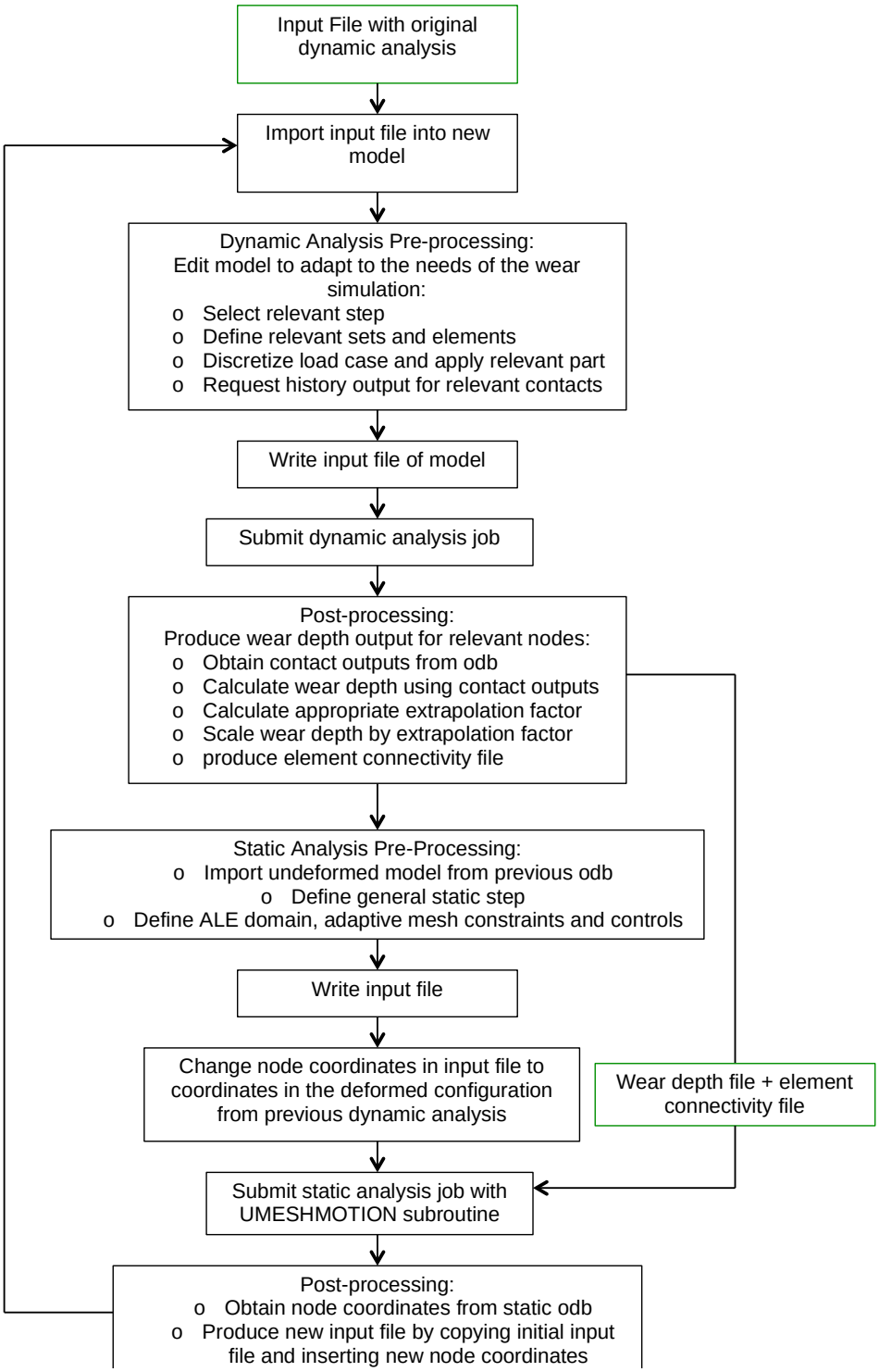


**Figure 4.15:** Flowchart for wear simulation based on editing the input file

The most important task units in described in Fig. 4.15 are now going to be discussed in detail.

1. **Dynamic Analysis Pre-Processing**
   The input file method essentially relies on Abaqus/CAE to repeatedly generate new models by importing information from an existing input file. The new model can be easily modified with simple Python scripting, which makes it very easy to add new features (such as adaptive meshing) or to suppress elements that are not essential to the current analysis. The input file method is therefore very flexible. In principle, one can import any arbitrary input file and tailor the newly created model to the needs of the wear simulation.
   In most cases, there are a few pre-processing tasks that have to be performed for all models if the relevant items are not already defined in the original input file. First, if multiple steps exist in the input file, it is necessary to first select the one for which wear-relevant outputs should be requested. For example, in most input files that define the wastegate's dynamic simulation, there are several analysis steps that are supposed to reflect some type of characteristic motion. Those might include, for instance, a step for opening the flap to the desired initial angle, stationary steps where the flap does not move, or steps for different ways of closing the flap. However, the step that is of interest is usually the one in which time-varying gaspulsation loads act on the flap while the lever is moved by the displacement of the actuator (see Ch. 3). In a wear simulation that focuses on the effects of a particular load case, all non-essential steps should therefore be suppressed.
   Having selected the relevant step, one should subsequently decide on how to process the loads and boundary conditions defined in the original input file. For example, if the objective is to achieve a high accuracy in the wear simulation, one can subdivide the load into several parts (with respect to the time) and apply them in separate simulation cycles, meaning that the next part of the load is applied after the geometry has been modified by the previous wear simulation. If this is unnecessary, the original load case can of course also be be maintained.
   Finally, the pre-processing code should define a history output request for all contact solution variables that are going to be relevant for calculating the wear depth. Of course, it is also possible to request any additional outputs that one intends to evaluate later.

2. **First Post-Processing**
   Once the first dynamic simulation is completed, the script proceeds to perform some essential post-processing tasks. The wear depths for all nodes in contact are calculated in this step by extracting the history of relevant contact solution variables from the output database and combining them according to the selected wear equation. For simple sliding wear, the post-processing script should therefore perform the following tasks:

   (a) Obtain the history output for CPRESS, CSLIP1 and CSLIP2,

   (b) Calculate the average value of CPRESS between consecutive time increments,

   (c) Calculate the incremental relative sliding distance for CSLIP1 and CLIP2 by subtracting the value at increment i-1 from the value at time i. Then calculate the vector magnitude of the incremental CSLIP1 and CSLIP2, and finally:

(d) Calculate the wear depth with Archard's equation and the predefined wear coefficient K

To speed up the wear simulation, the calculated wear depths are often amplified with a certain "extrapolation factor". The calculation of this factor and the rationale for using it will be discussed more extensively in Ch. 4.5. For now it suffices to say that in most cases, the calculated wear depths are uniformly scaled up with a certain integer. The final values for the wear depth at each node in the adaptive constraint region are finally written to a simple external text file that is read later by UMESHMOTION.
In addition to the wear depths, the post-processing will also generate a text file containing the element connectivity definitions of all elements that belong to the adaptive constraint region. As discussed in Sec. 4.3.1, this information is needed to determine the wear direction at corner and edge nodes.

3. **Static Analysis Pre-Processing**
Once the first post-processing step is completed, the model for the wear simulation can be set up in a static analysis. The only purpose of the static analysis is to move the contact nodes according to the wear depths calculated in the post processing step whilst maintaining the position of all parts at the end of the previous dynamic analysis. This means that all rigid body motions should be carried over to the new "wear simulation model" such that the next dynamic cycle is able to continue the motion of the system from where it was interrupted in the previous cycle.
The most obvious way to transfer the latest position of the assembly is to import the parts in their deformed configuration in the last increment of the last step of the dynamic analysis. However, the deformed configuration can only be imported as parts but not as models. This means that all sets and surfaces that have been conveniently defined in the original input file will be lost and have to be re-constructed in the new model. For mesh-based parts created by the import function, finding the appropriate surfaces and element domains turns out to be rather difficult and tedious. It would thus be much easier if the original definitions could be maintained.
To solve this problem, one first imports the undeformed assembly as a complete model the same way it was done at the beginning of the routine. Subsequently, the code performs all the necessary pre-processing tasks for setting up the wear simulation, such as defining adaptive mesh domains and constraint regions, requesting relevant history outputs (for example the volume loss), and removing unnecessary loads and constraints that might conflict with the adaptive meshing. Once the model is thoroughly prepared, the script then creates a job and instructs Abaqus/CAE to write the input file for this new job. This input file now contains all relevant wear simulation inputs as well as the sets and surfaces from the dynamic analysis.
In a subsequent step, the script will extract the coordinates of all nodes in the last time increment of the previous dynamic simulation and insert them into the latest wear simulation input file. In practice, the script creates a second wear simulation input file where only the node coordinates are different. This copied input file can now be submitted for the wear simulation analysis, where it is of course essential that the appropriate UMESHMOTION subroutine is specified.

4. **Second Post-Processing**
After completing the static analysis where the modified geometry is established, one

must again find a way to transfer the deformed geometry to the next simulation cycle. As discussed earlier, importing the deformed part directly is impractical because everything except what is included in the part would be lost. This issue is even more problematic here since one would also have to redefine the all the loads and boundary conditions for the next dynamic analysis. Therefore, the most efficient method is again to edit the input file. This time, the script generates a copy of the original input file where all loads, boundary conditions and interactions are conveniently defined, and only inserts the node coordinates from the last time increment of the static wear simulation. In the next cycle this modified input file is imported by Abaqus/CAE and overwrites the first dynamic model. This whole process can be performed any number of times.

One observes that this method of importing and rewriting the input file repeatedly is very flexible and versatile since the model is rebuilt after each wear simulation. One can add, delete or redefine any set and surface that was initially imported with the original input file. If necessary, one may also apply new loads or boundary conditions to any step in the new model without problems. To include anything new, it is only necessary to add a few commands to the python script, though one should of course be consistent and ensure that new sets or load case definitions do not conflict with existing ones. In most cases there are ways to adjust everything to avoid inconsistencies.
A notable feature that cannot be changed is the number of nodes and elements. When the input file is rewritten, the script assumes that the number of nodes is the same from one analysis to the next. Therefore, removing or adding new nodes or elements in any pre-processing step will most likely lead to unpredictable errors. Though it is not recommended, the mesh may be changed slightly using the *editMesh* function of Abaqus/CAE if the operation does not affect the total number and labelling of the nodes. This is possible because all parts in imported models become mesh-based parts. For more details on the mesh editing tool please refer to the Abaqus/CAE User's Guide [25] and the Abaqus Scripting Reference Guide [26].
The input file method therefore provides the user with a lot of freedom to customize the model and the output of the analysis. The python routine can also be easily transformed into a general script that is able to work with a variety of structurally similar input files such that it may serve as a quick tool to assess the effects of different parameters (e.g. a different wear coefficient, load case, extrapolation factor etc.). However, as indicated earlier, this approach has a major flaw: By simply rewriting node coordinates in the input file, nothing except the modified geometry can be transferred to the next cycle (the same applies if the "import deformed part" function is used). This means that the entire stress state of the system is lost whenever a new model is created. Also, in addition to the required rigid body displacements, the routine will also transfer all elastic deformations incurred in the previous analysis. In each subsequent cycle the load is therefore reapplied to the deformed part as if it was undeformed, which might introduce significant errors in the contact solution (as will be shown in Ch. 5.2). The error generated by this problem is demonstrated and evaluated extensively in Ch. 5.2, and a straight-forward solution will be suggested for a simple system.
To avoid any errors by default, it is necessary to preserve not only the change in geometry but also the overall stress state of the system in subsequent cycles. In Abaqus, this is typically done with a *restart analysis*. As the next section will show, a wear simulation routine based on restart is very difficult to implement and also less flexible than the input file method, but would in theory be the most legitimate approach.

### 4.4.3 Restarting the Analysis

*RESTART* is the standard method to continue an analysis from a specific step or increment. In the first analysis one would request restart data to be written with a certain frequency (e.g. every second increment, or based on a time interval), such that a second analysis may then continue the calculations from any recorded state with a new step and/or a new load case. In the context of wear simulations, the idea is to perform the wear simulation analysis - i.e. the static analysis where ALE adaptive meshing is applicable - as a restart from the final step of the dynamic analysis where contact solution outputs have been recorded. This approach would also allow for access to the output database in a post-processing step, where the wear depths for each node can be calculated and documented in an appropriate fashion. The routine for the restart method is illustrated in Fig. 4.16:

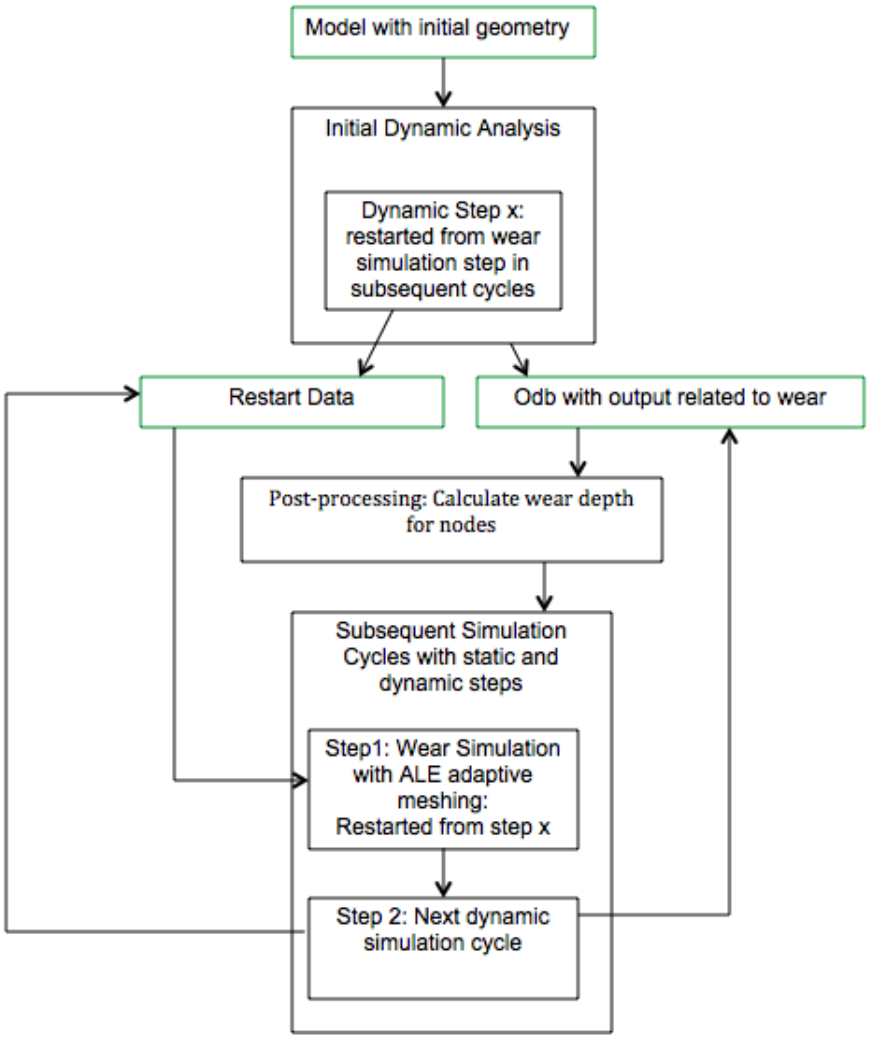


**Figure 4.16:** Flowchart for wear simulation based on restart

As one observes in Fig. 4.16, the simulation routine based on restart is decidedly shorter than

the input file method discussed in the previous section. The restart method only requires a single post-processing phase in order to calculate the wear depths since the modified geometry is carried over by default. One would start with an initial model and its load case, perform the dynamic simulation whilst producing restart data, generate wear-related results by reading the odb in the same way as described for the input file method, and finally restart the analysis from the last increment of the dynamic step. The simulation would in this case continue with an added static step where all parameters related to adaptive meshing is defined, and then simply append the next dynamic cycle as an additional step to the same analysis. In the appended dynamic step the script must request restart data to be generated again in order to continue the loop. With this method, the changes in geometry as well as the most recent stress state are automatically carried over to the next step (which is actually the point of a restart analysis). It is therefore also no longer necessary to manually change the node coordinates.

The restart-based simulation routine therefore appears to be much simpler and mathematically more accurate than the input file method, but it is in fact subject to many restrictions. Unlike the input file method, adding sets and load cases to the model with Abaqus/CAE proved to be quite problematic in the restart routine. For example, in a restart analysis new sets and surfaces can only be defined with names that include the name of the assembly in the initial analysis. If the initial model is created by e.g. importing an arbitrary input file (as most wastegate models are defined with input files), a simple python code that adds a new step to the model for the restart analysis might induce Abaqus/CAE to generate an invalid input file. The most frequently encountered input file errors involved incorrect naming of sets and surfaces or redundant definitions of previously defined amplitudes. In many cases it was necessary to manually correct for the errors generated by CAE. (Of course it is possible to write a code to do the correction automatically, but the correction is most likely specific to each model).

Apart from input file definition errors, the restart routine also encountered several unexpected problems. For example, propagation of displacement boundary conditions and adaptive mesh constraints over more than two analysis cycles proved to be problematic, since they have to be de- and reactivated many times over. For instance, if a general static step with ALE adaptive meshing is followed immediately by another general static step, the next static step with adaptive meshing failed to produce any output if the adaptive mesh constraint was deactivated in the second step. In most cases, the restart loop was not able to run automatically since the execution had to be paused to manually correct the CAE-generated input files. In subsequent dynamic/static cycles, the analysis also sometimes failed to converge due to unknown reasons.

Due to the many difficulties encountered, attempts to implement a continuous and automated simulation routine based on restart unfortunately have not been successful. The most critical problem in the end was that the dynamic analysis in the third cycle unexpectedly terminated in the first increment.

In conclusion, the restart routine turned out to be very unpredictable and difficult to automate. It is likely that a functioning routine can be established with more time and effort, but this approach is still less flexible than the input file method due to the restrictions to the definition of sets and surfaces. Unlike the input file method, it would certainly be very challenging to set up a restart routine that is able to work with any predefined input file. If a solution can be found for the stress state problem, the input file method would certainly be preferable. As will be shown in Ch. 5.2, at least for simple systems the errors generated by

the input file method can be corrected quite easily.

In conclusion, the input file method seems to be the only feasible approach that makes it possible to perform repeated wear simulations based on contact outputs from a dynamic analysis. The results obtained with this simulation routine is therefore going to be investigated in detail in Chapters 5 and 6.
As mentioned in the introduction to this chapter, another important element that should be included in a wear simulation routine is a sensible approach to the “extrapolation" of the calculated wear depths. The next section introduces a simple technique that is able to significantly speed up the progression of wear.

## 4.5 Extrapolation of Calculated Wear Depth

A simulation of the wastegate's dynamic motion covers a significantly shorter time than is required for wear damage to become apparent. With a mesh that is sufficiently refined to ensure a reasonably accurate contact solution, simulating the system's behavior in 0.08 seconds real time could take up to 19 hours. However, significant changes in the wastegate geometry due to wear usually occurs after several days or weeks in operation. For example, the distinct wear damage on the wastegate lever occurred in an engine endurance run that lasted approximately two weeks.
As mentioned before, even if one could reduce the computation time to being equal to the real time it describes, it would still take an extremely long time to finish one complete wear simulation. Clearly, the geometric changes that are produced in a single simulation cycle must be enhanced in some way.
The straight-forward solution is to simply scale up the calculated wear depth by some constant factor. But what number is reasonable and how should it be determined? Of course it is possible to choose the factor arbitrarily, but without some logical criteria it would be impossible to know if the extrapolation could significantly affect the accuracy of the results. As shown by Mukras et al., one could start by considering the impact on the contact solutions [27]. Most analytic equations for calculating wear are closely related to solutions from the contact problem. If Archard's equation is used, wear occurs only at nodes where contact pressure (CPRESS) and relative slip (CDISP) are found simultaneously. It is obvious, therefore, that apart from the coefficients in the wear equation itself the accuracy of the wear simulation will largely depend on the mesh refinement and the contact solution variables.
If the wear incurred over a certain time period is very small, the positions of the contact nodes will be approximately the same as what they were in the beginning. This means that even if the nodes are shifted, the contact solutions resulting from the same external loading also will not have changed noticeably. Therefore, it is reasonable to assume that this very slightly modified contact will again produce approximately the same wear depth. This means that the initially calculated wear depths can be increased linearly to represent the accumulated effect of many cycles without actually having to perform the calculations for all those cycles. Clearly, the extent of wear can only be amplified until there is a significant, i.e. unacceptable, error in the contact solution. One observes that if all wear depth numbers are multiplied by a single factor, the differences between those initial values will also have increased, thus making the contact surface increasingly rough, or less smooth. However, at least in uniform

sliding wear with no other significant damage mechanisms, the contact surface should in theory always be macroscopically smooth, meaning that one should always find a relatively continuous distribution of the contact stresses over the wear profile. Mukras et al. therefore decided to evaluate the contact pressure found at adjacent nodes in the contact area and selected the extrapolation factor based on a *critical pressure difference* [27]. The technique could have been adopted for the work in the current study, but it turns out that this approach would be rather inefficient since the wear depths are calculated outside of the analysis. To evaluate the contact pressure, one would have to insert an additional contact analysis after the wear simulation, which could significantly increase the overall computation time.
Since the method proposed by Mukras et al. is based on a rather arbitrarily defined difference in the contact pressure, it should also be legitimate to apply a critical difference directly to the calculated wear depth. If the difference in wear depth between adjacent nodes is found to exceed this limit, one can assume that the smoothness of the wear profile - and therefore the smoothness in the contact pressure distribution - is also severely affected. Therefore, the following tasks are performed in the post-processing step to find the appropriate extrapolation factor:

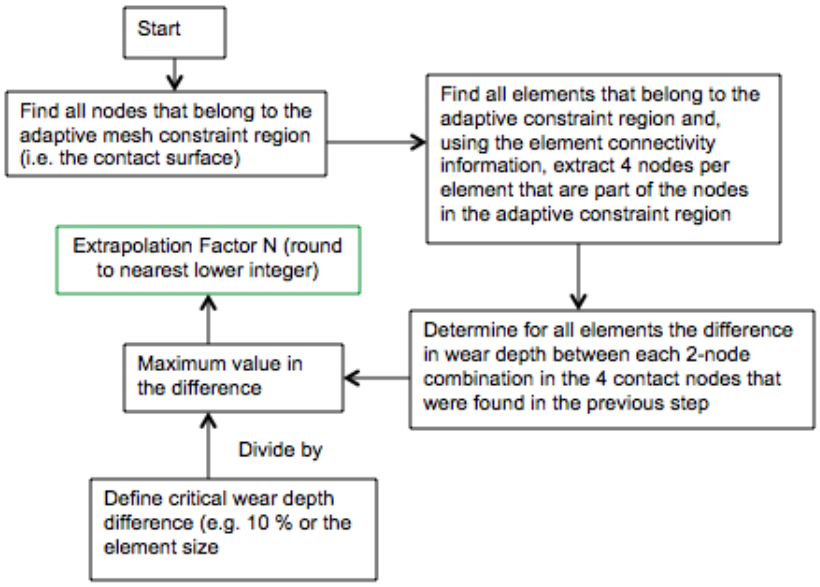


**Figure 4.17:** Flowchart for finding the extrapolation factor in each simulation cycle

As one observes in Fig. 4.17, the extrapolation factor N is always calculated such that it amplifies the maximum wear depth difference to just about the critical limit. All other wear depth numbers will be multiplied by the same factor to represent the effect of N loading cycles. The average difference in node shift across any element should therefore always be lower than the specified critical limit. The extrapolation factor is rounded down to the nearest integer because it does not make sense to include the effects of some fraction of a cycle. Including a fraction means that in theory the system configuration has not returned to the final configuration in the simulation, so the next cycle should actually continue from that position and not from the beginning. In practice, however, the extrapolation factors are usually very large (typically around 1000 - 5000) such that the effect of the decimals are negligible.

Of course, the interesting question is how to select the critical difference limit. The most reasonable thing to do is to investigate the contact stress distributions beforehand to see which maximum height difference over a single element would still generate an acceptable pressure variation. Though in the end it will mostly depend on how much accuracy one is prepared to sacrifice for computation speed.

## 4.6 Summary

This chapter elaborated on all relevant methods and techniques that have been investigated for wear simulations applied to a dynamic system. First, Archard's fundamental equation for sliding wear was adapted to the the needs of an FE simulation by making a few reasonable assumptions. Second, it was found that the basic structure of UMESHMOTION has to be improved with an additional feature in order to correct for the wear direction at surface discontinuities. Two methods have been suggested to determine a reasonable vector for node displacement at edges and corners: 1. by using the local normal vector, or 2. by following a subsurface node. The former method is more reliable but only strictly applicable to perpendicular surfaces, and the latter is less robust but in principle also valid for sides at oblique angles. The UMESHMOTION subroutine code for both methods can be found in Appx. B and C.
The discussion then proceeded to a number of secondary issues that follow from the derived operating principles of UMESHMOTION. It was found that certain rules have to be followed in the definition of interactions and adaptive constraint regions in order to be consistent with the way in which the wear directions are defined. In addition, wear on both surfaces of a contact was found to be possible by using the general contact formulation, or by defining a symmetric contact formulation if master/slave contact pairs are used.
In the second part of this chapter, several methods to integrate the wear simulation with a dynamic analysis were considered. Techniques that have been investigated include: simple stacking of dynamic and static steps within a single analysis, repeated importing and changing of input files with intermediate post-processing steps, and a routine based on Abaqus' restart capability. Though the input file method contains an inherent error, it was found to be the most feasible and flexible approach, and was therefore fully implemented and parameterized for general input files. Finally, a simple extrapolation scheme to accelerate the wear simulation was proposed. This extrapolation scheme calculates a single extrapolation factor to linearly amplify the wear depths found with Archard's equation and is limited by a critical difference in the wear depths of adjacent nodes. This limit is used to ensure a reasonably smooth contact surface and contact stress distribution. The Python script for the wear simulation routine according to the input file method can be found in Appx. A.

# Chapter 5

# Verification and Performance Evaluation of Wear Simulation Methods

As discussed in the last chapter, the techniques derived for solving different problems of the wear simulation are effective in producing the required output but are limited in their accuracy. This chapter will specifically assess the performance of the newly devised input file method and demonstrate the error that arises due to the loss of the stress state between consecutive analyses. If one chooses to use this method to predict the progression of wear in a certain model, one should be aware of its shortcomings and attempt to take those into account in the evaluation of the results.
However, before any observations are made it is necessary to first verify that all calculation procedures have been implemented correctly. This is demonstrated by comparing the program's output to a very simple case where the result can be calculated analytically.

## 5.1 Verification of UMESHMOTION and Output Processing

While the input file method itself is inherently flawed, the calculation of wear based on Archard's equation should produce the expected result in very simple applications where all parameters are known. By showing this one confirms that any discrepancies in the analysis of more complicated systems are not due to programming errors. This section will verify 1. the calculation of accumulated wear depth in the post-processing step of the input file method, and 2. the coding of subroutine UMESHMOTION where the magnitude and direction of node shift are determined.

### Verification model: cube sliding on flat surface

In order to verify the numerical data produced by Abaqus one must select an example where the result can be easily calculated analytically. This is possible with the model shown in Fig. 5.1:

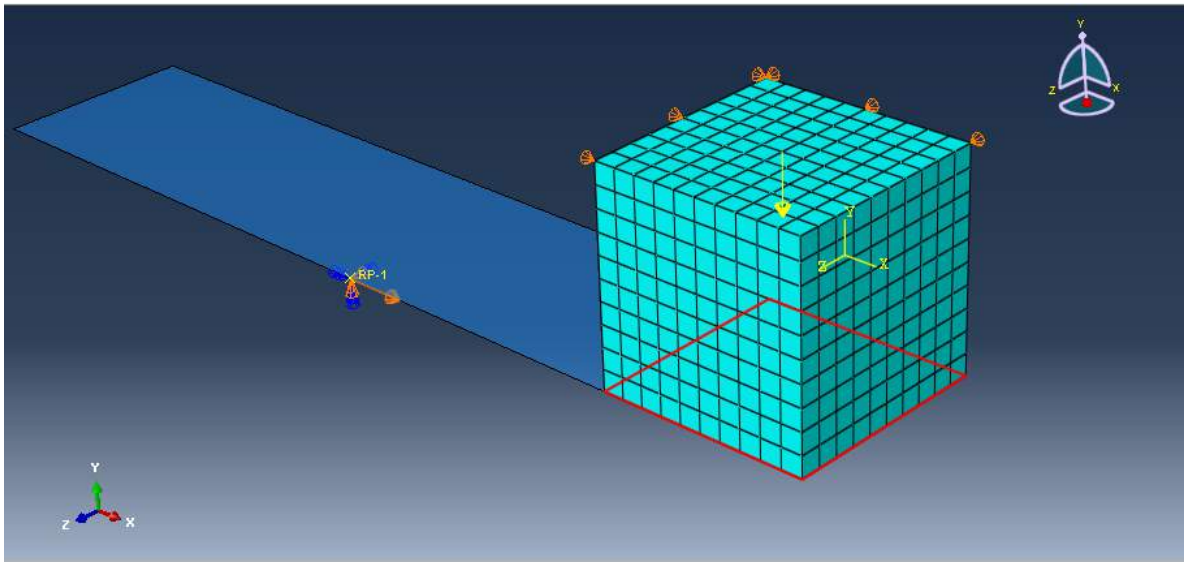


**Figure 5.1:** Simple model used for the verification of wear calculations

In Fig. 5.1, a force of 100 N is acting at the center of the top surface of a cube with side lengths of 0.3 m. All nodes on the top surface are coupled to the node where the force is applied such that they experience the same displacement. Also, the cube is fixed on two sides in x and z, respectively, such that the isotropic elastic material (assumed here to be Aluminum 6061-T6, though the numbers are not important) expands uniformly when a normal force is applied in y. A surface-to-surface contact is defined between the analytic rigid surface below the cube and the cube's bottom surface, where a "hard" contact relationship is used for the normal interaction and "frictionless" for the tangential relationship in order to prevent nonuniform sliding. Additionally, due to the applied boundary conditions in x an z at only two edges of the cube (in order to ensure uniform deformation) the contact pressure tends to become higher on the left side of the contact surface and lower on the right side when the rigid surface is moved to the left. This is prevented by specifying in the contact properties that the surfaces should not be allowed to separate after contact is established, which does indeed correspond with reality.
If the rigid surface is now moved by 0.2 m to the right, and the contact region is defined at the bottom surface of the cube (red square), the presence of a contact pressure and simultaneous relative sliding should generate a uniform loss of material from the bottom surface. Assuming a reasonable value of 5.0E-13 $[m^3/Nm]$ for the wear coefficient (approximately the average value found in Meyer's study for different wastegate material pairings [3]), the lost volume according to Archard should be:

$$\begin{aligned} W = KF_N s = 5.0E-13 \left[\frac{m^3}{Nm}\right] \times 100\,[N] \times 0.2\,[m] \\ = 1.0E-11\left[m^3\right] = 0.01\left[mm^3\right] \end{aligned} \tag{5.1}$$

where $W$ is the wear volume (volume of lost material), $K$ is Archard's wear coefficient, $F_N$ is the normal force and s is the sliding distance.
In Ch. 4.1 it was explained that for FE applications, Archard's equation has to be modified to describe the wear depth instead of the wear volume. The equation implemented in UMESHMOTION is shown again for convenience:

$$h_{node} = Kp_{avg}\Delta s = K\left(\frac{p_{i-1}+pi}{2}\right)(s_{total,i} - s_{total,i-1}) \tag{5.2}$$

where $h_{node}$ is the wear depth at a particular node, $p_{avg} = \frac{1}{2}p_{i-1} + p_i$ is the average contact pressure at that node between the current and the previous time increment, and similarly, $\Delta s = s_{total,i} - s_{total,i-1}$ is the incremental relative slip from time i-1 to time i.
Since all nodes at the bottom surface experience the same sliding distance, the lost volume produced using the adapted Archard equation for wear depth should be the same as the number calculated in Eq. 5.1. As shown in Ch. 4, the wear simulation in this study is performed after the accumulated wear depths at all nodes have been calculated using the contact solutions from the previous analysis (and Eq. 5.2). Since normal force and contact area are constant over the complete sliding distance, the wear volume in this case should be independent of whether the node shift is performed simultaneous to the sliding or in a subsequent step.
Indeed, the post-processing script calculates the same total wear depth of h = 1.1111E-10 m for all nodes at the bottom contact surface, with minor differences only in the fifth decimal place. The cube has a constant side length of 0.3 m, so the wear depth induced by a uniform upward node shift of 1.1111E-10 mm should be:

$$W = hA = (1.1111E-10[m])\,(0.3[m] \times 0.3[m]) \simeq 1.0E-11[mm^3] \tag{5.3}$$

where $A$ is the contact area of the cube. This results verifies that the post-processing calculations are correct, as the wear depth corresponds to the analytically determined wear volume.
To find out whether UMESHMOTION itself has been implemented correctly, the wear depth is now applied to the unconstrained cube in a general static analysis where the adaptive mesh constraint region is activated on the contact surface in the previous analysis (the bottom surface, shown in Fig. 5.1). The step time in this "wear simulation" is first specified to be 1.0, and the total wear depth numbers read in from the external file are applied incrementally such that in each increment i the node is shifted by a fraction $\Delta h_i$ of the total depth $h_{total}$:

$$\begin{aligned} \Delta h_i &= \Delta t_i \times h_{total} \\ h_i &= h_{i-1} + \Delta h_i \end{aligned} \tag{5.4}$$

Therefore, with a step time of 1.0 the nodes should have shifted by the value specified in the external text file. Fig. 5.2 shows the volume loss over the wear simulation step for an increasing number of step increments:

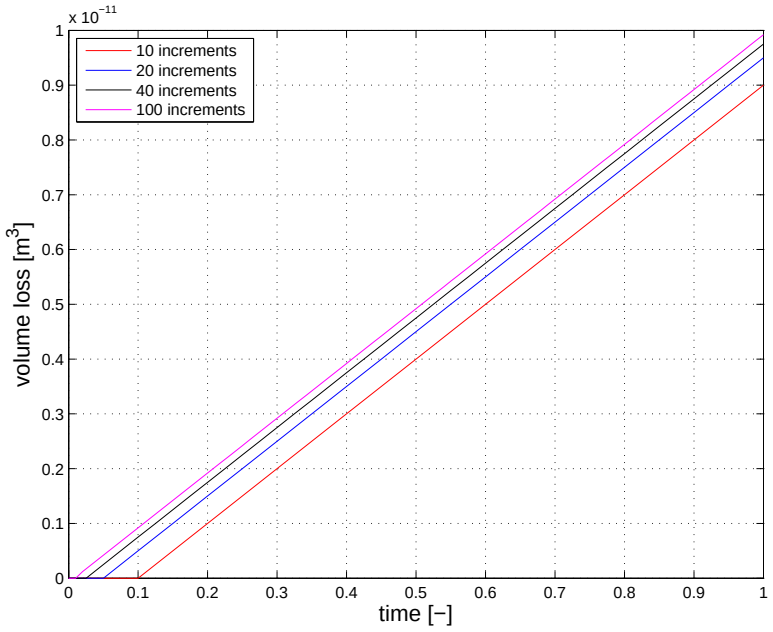


**Figure 5.2:** Volume loss with time at 10, 20, 40 and 100 step increments

As one observes in Fig. 5.2, the total volume loss is in fact not exactly equal to the calculated value of 1.0E-11 $m^3$, but always slightly lower. The total volume loss for the total increment numbers in Fig. 5.2 is given in Tab. 5.1:

| Number of step increments | 10 | 20 | 40 | 100 |
|---|---|---|---|---|
| Volume loss [$m^3$] | 9.0E-12 | 9.5E-12 | 9.75E-12 | 9.9201E-12 |

**Table 5.1:** Total volume loss with time at 10, 20, 40 and 100 step increments

Fig. 5.2 and Tab. 5.1 show that there is apparently an error in the calculated volume loss that depends on the the number of step increments. The error evidently becomes smaller as more increments are used. By observing Tab. 5.1 closely, it becomes apparent that the difference between the analytically calculated volume of 1.0E-11 $m^3$ and the simulation outputs is always equal to the volume that is incurred in a single time increment, i.e. a difference of $\Delta V = \Delta t \times V_{total}$. In addition, for all graphs in Fig. 5.2 the volume loss does not start at time = 0 but at a slightly later time. In fact, it turns out that node shift does not occur until the second increment (which is also shown by the history of displacement U at the contact nodes), even though UMESHMOTION does in fact return the correct node shift magnitude from increment 1 if the numbers are written to an external text file. This seems to be a rather curious glitch in the internal programming of adaptive meshing as no other reason was found. If no correction is made, the error in the incurred volume loss will

therefore always be proportional to the increment size in the step.
A simple solution to this problem is to simply apply an additional increment such that the total specified wear depth can be still be attained at the end of the step. For example, if an increment size of 0.1 is initially used for a total step time of 1.0, it is necessary to reset the time to 1.10 in order to achieve the wear depth that was calculated in the post-processing script. Similarly, the time should be set to $t = 1.05$ for an increment size of $\Delta t = 0.05$, $t = 1.025$ for $\Delta t = 0.025$, $t = 1.01$ for $\Delta t = 0.01$ and so on. The graph in Fig. 5.3 shows the history of volume loss for an increment size of $\Delta t = 0.1$ applied for a total step time of $t = 1.1$ such that the total number of increments is 11 instead of 10:

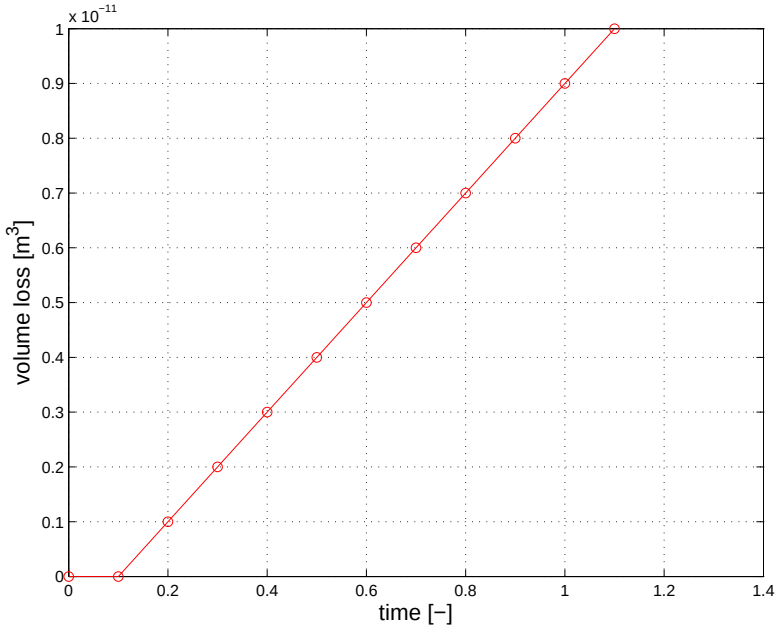


**Figure 5.3:** Volume loss with total step time of $t = 1.1$ and and increments size $\Delta t = 0.1$

As one observes in Fig. 5.3, even though there is still no volume loss at $t = 0.1$ (increment 1), the final volume loss is now equal to 1.0E-11, which is corresponds to the analytically calculated value. Since the time used in the general static analysis does not represent a real time and is only used for the application of different loadings and, in this case, geometric modifications, it is legitimate to use any step time that will result in the correct node displacement. It is thus verified that the UMESHMOTION subroutine is indeed able to produce the predefined wear depths if a correction in the wear simulation's step time is applied.
Having verified that there are no errors in the implementation of UMESHMOTION and the post-processing script, it is now time to assess the performance of the input file routine itself for contact conditions that are more complicated than the uniform cube/plate interaction.

## 5.2 Performance Assessment of Input File Method

The idea of using Abaqus/CAE to import and edit the input file repeatedly was found to be the only viable technique among several others that seemed more obvious and less trou-

blesome at first. As discussed in Ch. 4, the single-analysis method was not suitable for the needs of the current application, and the restart method unfortunately could not be successfully automated. The input file method is the only simulation routine so far that is able to process a given model an arbitrary number of times based on parameters that are determined automatically in post-processing. However, it is inhibited by a significant drawback.
As pointed out in Ch. 4, between successive analyses the procedure can only transfer a part's deformed geometry but not its latest stress state. In the most elementary system, such as the cube/plate model used for verification in Sec. 5.1, this problem does not exist as it is not necessary to rerun the analysis multiple times in the first place: Since the applied load and the initial contact area do not change, the pressure and relative slip experienced by each node is constant, and thus the outputs can be collected and multiplied to obtain the accumulated wear depth in a single step. However, in most systems the contact area changes with the progression of wear. For example, in the wastegate's lever/bushing interface the contact area is very small in the beginning (it would be single point if elastic deformations are neglected) and expands as the curved surfaces of lever and bushing are flattened due to material removal. A schematic is shown in Fig. 5.4:

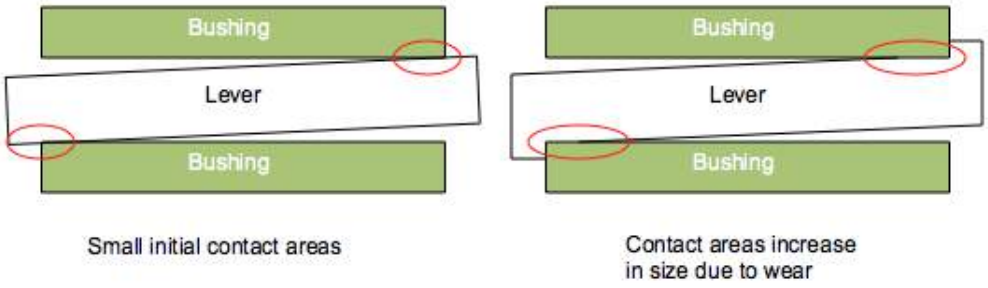


**Figure 5.4:** Cross-section of lever/bushing assembly shows increase in contact area

As wear progresses, contact pressure and relative slip will therefore spread to nodes that have not been in contact initially (unlike the cube/plate system where the contact surface will always contain the same nodes). For such systems with varying contact areas it is therefore necessary to perform the wear simulation in multiple steps, or else changes in the contact conditions cannot be taken into account.
Because the stress state is never maintained from one analysis to the next, an error is introduced with each new cycle if the loads are simply continued from where they stopped in the last cycle. In practice, one should therefore try to restore the stress state before the actual loading is continued and additional contact outputs are recorded. In systems with a simple load case (such as the shaft/bushing model investigated in the next paragraphs) the error can be eliminated with a simple trick. However, in more complex systems like the full model of the wastegate it is not always clear at what point the stress state can be considered restored. In those cases it is necessary to carefully inspect the system's behavior to find a reasonable state for continuation.
The following case study of a shaft rotating in a bushing will demonstrate the error introduced in each new cycle of the input file method. It is also shown how one could, in this particular case, attempt to correct for this error.

### 5.2.1 Input file method performance and error investigation

In order to assess the performance of a certain technique, one would conventionally compare its results to those obtained with a more reliable procedure. The input file method was developed mainly to account for simulations where ALE adaptive meshing is not possible, but if it is really applied to a dynamic analysis, similar results could not be obtained with any other method. Therefore, this investigation will consider a static analysis where the wear simulation can be performed in parallel. In this way, the results obtained from the input file method can be directly compared to the fundamentally more legitimate results of the standard method.
The investigated model is a simplified version of the wastegate's lever bushing assembly and approximately has the same dimensions. The model set-up and load case are shown in Fig. 5.5.

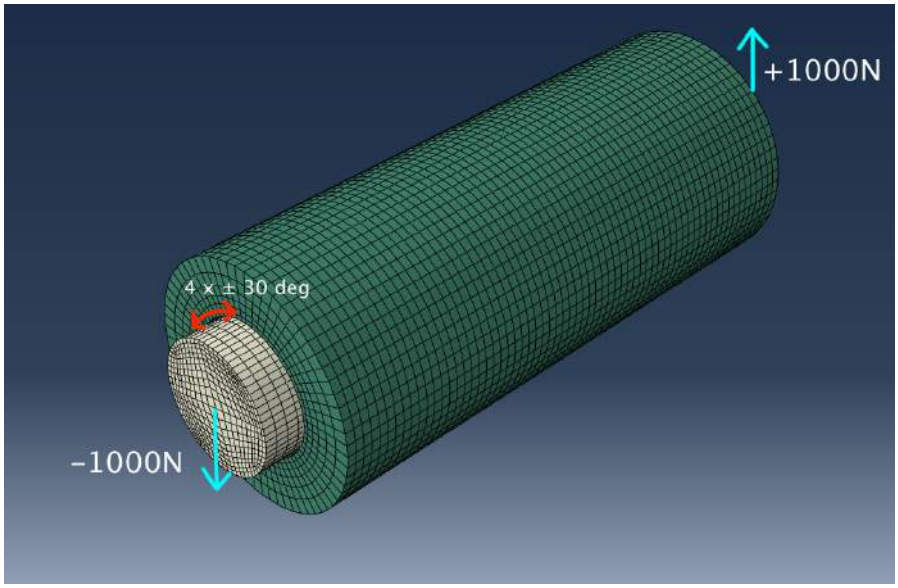


**Figure 5.5:** Simplified FE model of wastegate shaft and bushing

For the simultaneous wear simulation analysis which will serve as the reference case for comparison, two general static steps are applied:

1. Tilting the shaft in the bushing by applying a concentrated force of 1 kN in opposite directions to the center of the rod's circular ends (nodes at the ends are coupled to the center node). A very small contact area is established initially at both ends of the assembly. This configuration is similar to the wastegate's lever/bushing interaction which produces sliding wear in the real part. Fig. 5.6 shows a cross section of the model at the end of step 1:

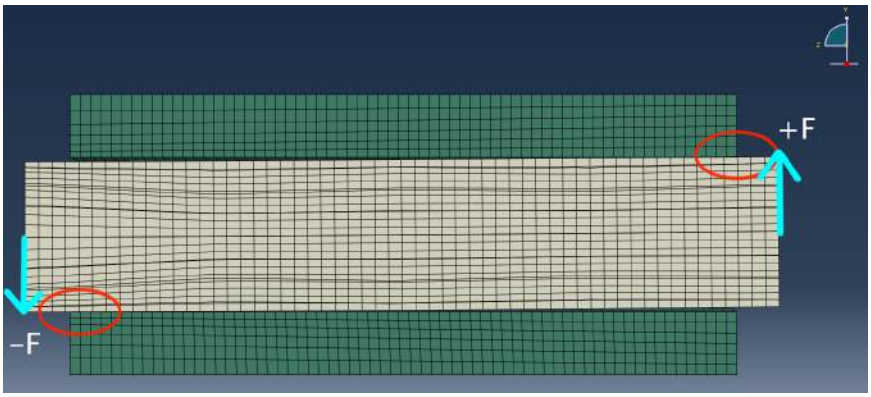


**Figure 5.6:** Tilted shaft for shaft/bushing contact at both ends

2. Rotating the shaft four times back and forth by 30° using a rotational boundary condition applied to the reference point at the center of the left side (see Fig. 5.5). This step lasts for 4 cycles, where a 30° rotation back and forth is completed in 1 cycle. Fig. 5.7 shows the complete rotational displacement in 1 cycle:

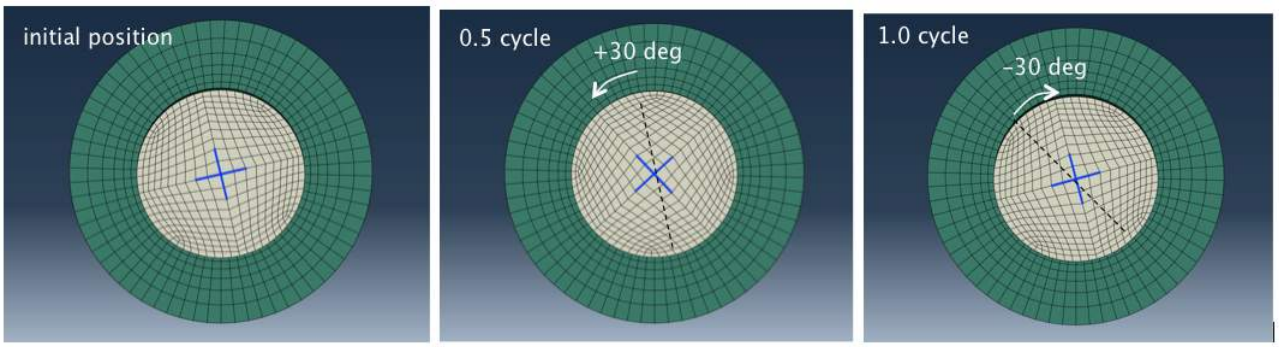


**Figure 5.7:** Complete rotational displacement in 1 cycle

For a shaft diameter of 9 mm the shaft's total sliding distance $s_{total}$ induced by the rotation is:

$$s_{total} = 4 \times 2 \times 0.542[rad] \times \frac{\pi 9[mm]}{12} = 18.45[mm]. \tag{5.5}$$

ALE adaptive meshing is applied during this rotation step. The inside surface of the bushing is selected as the adaptive mesh constraint region where nodes are shifted by UMESHMOTION according to local contact solutions in each time increment. The contact regions in the bushing are located right at the edges of the part and can be seen more clearly.

The magnitude of the loads and displacements imposed on the system are selected such that a reasonable amount of wear can be generated. Table 5.2 lists all additional input parameters related to the material:

| Parameter | E-modulus [GPa] | Poisson's ratio | friction coefficient | wear coefficient [m $^2/N$] |
|---|---|---|---|---|
| Magnitude | 90.8 | 0.3 | 0.1 | 1.0E-11 |

**Table 5.2:** Material input parameters

At the end of the wear simulation the wear profile produced by the simple static analysis is shown in Fig. 5.8. For improved visibility of the wear profile only the inside surface of the bushing is shown and the wear depth is visually magnified by a factor of 20. Figs. 5.9 and 5.10 show the detailed wear profile at the front and back ends (left and right in Fig. 5.6) of the bushing.

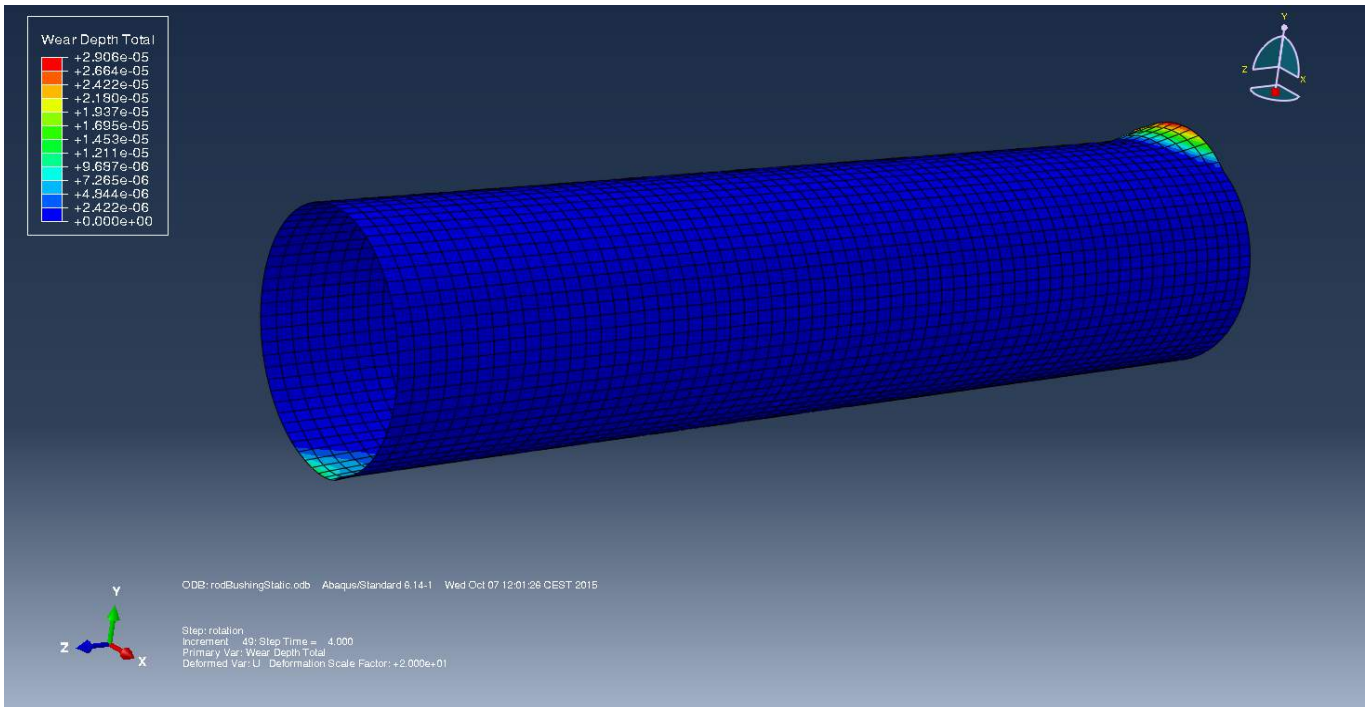


**Figure 5.8:** Wear profile at the end of static analysis (magnified 20 ×)

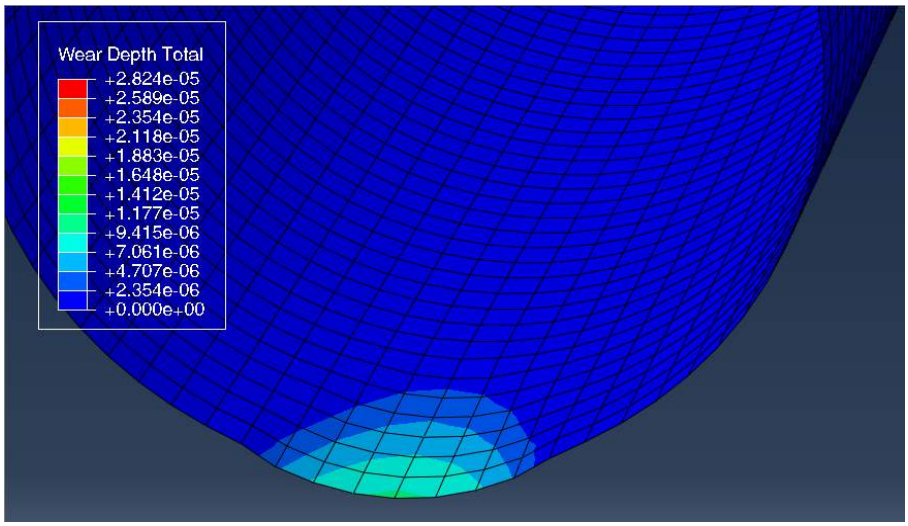


**Figure 5.9:** Wear profile left end

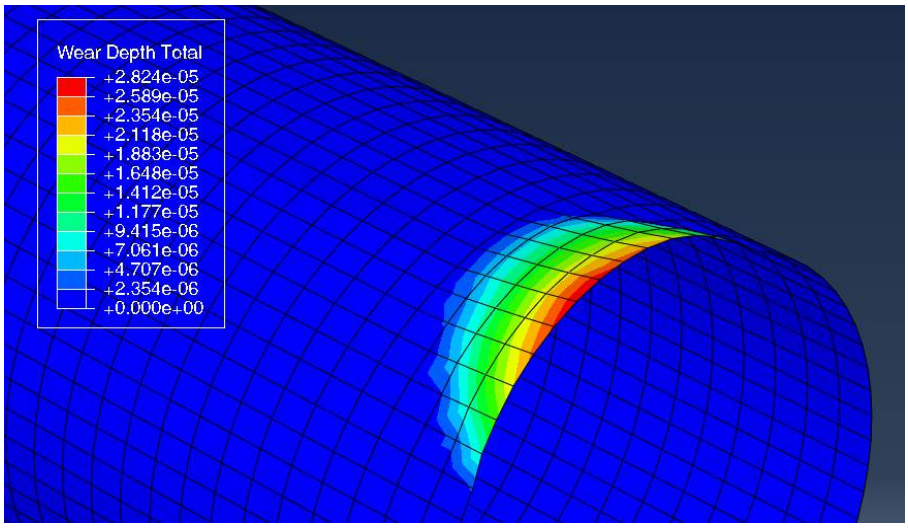


**Figure 5.10:** Wear profile right end (magnified 20 $\times$)

As one observes in Fig. 5.8, the wear profile created by applying node shift in each increment of the analysis is relatively smooth even when magnified 20 times, which agrees well with reality and confirms that the mesh is sufficiently refined. The maximum wear depth is found at the upper right contact of the bushing where the contact pressure is highest. The progression of volume loss over the duration of the rotation step is shown in Fig. 5.11:

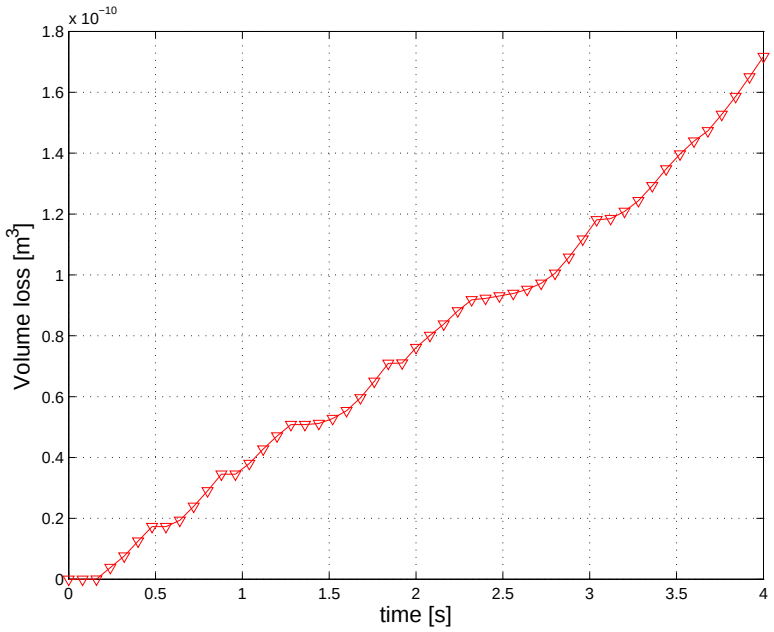


**Figure 5.11:** Bushing volume loss in rotation step

At the end of the rotation step the total volume loss is equal to 0.171734 mm$^3$. The wear profile and volume loss obtained with the simultaneous wear simulation can now be assumed as a reference to which results obtained from the input file method are going to be compared. For the input file method, the rotational displacement of the shaft is divided into an increasing number of cycles such that the result should, in theory, become increasingly similar to Figs. 5.8, 5.9 and 5.10. A simple flowchart in Fig. 5.12 clarifies the process.

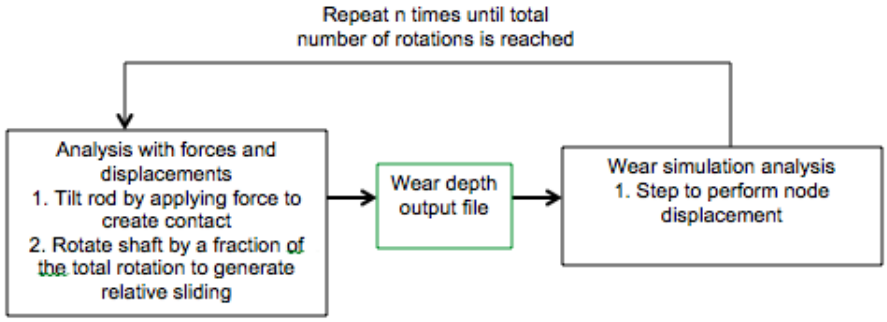


**Figure 5.12:** Flowchart for input file based analysis of rod/bushing model

Using the input file method, the wear simulation is first performed as described in Fig. 4.15 on the deformed configuration of the bushing. Fig. 5.28. The four cycles of rotation applied in the reference simulation are consecutively divided into 1, 2, 4, 8, and 16 cycles such that node shift occurs after every 18.45 mm, 9.225mm, 4.6125 mm, 2.3 mm, or 1.153 mm of sliding. The total volume loss obtained at the end of each complete run is plotted in Fig. 5.28 vs. the number of cycles used. The blue line represents the 0.1816 mm$^3$ from the reference analysis.

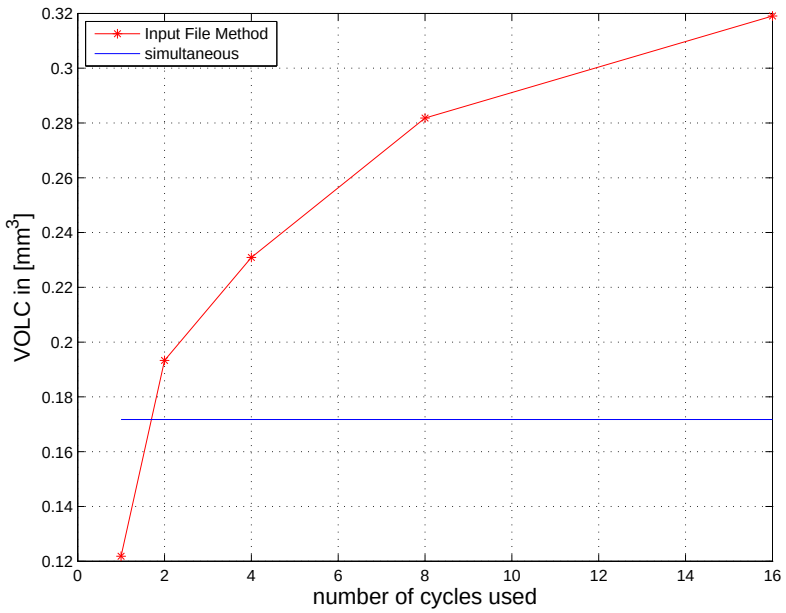


**Figure 5.13:** Material loss volume as more cycles are used

Two observations can be made from Fig. 5.28:

1. If only 1 cycle is used, the wear volume is underestimated by approximately 29.06%.

2. However, each time the number of cycles is doubled the wear volume is increasingly overestimated. The relative errors for cycles 2 - 16 are shown in Tab. 5.3:

| Number of cycles used | Reference Value | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| Total volume loss [$mm^{-3}$] | 0.17173 | 0.1933 | 0.2309 | 0.2818 | 0.3191 |
| Relative Error in % | - | 12.57 | 34.46 | 64.08 | 85.79 |

**Table 5.3:** Volume loss and relative errors with increasing numbers of cycles

The underestimation of the wear volume in case only 1 cycle is used is simply because the discretization of the rotation load case is insufficient. The reason is obvious and has been explained at the beginning of this chapter: As material is removed from the bushing, the initial contact area should become larger over time, meaning that contact pressure and relative slip should be registered at more nodes. This leads to a wider wear profile and a larger wear volume. The increase in contact area can be observed in the reference simulation where node shift happens in parallel. Figs. 5.14 - 5.17 show the contact area at the front end of the bushing at different steps times in the reference analysis:

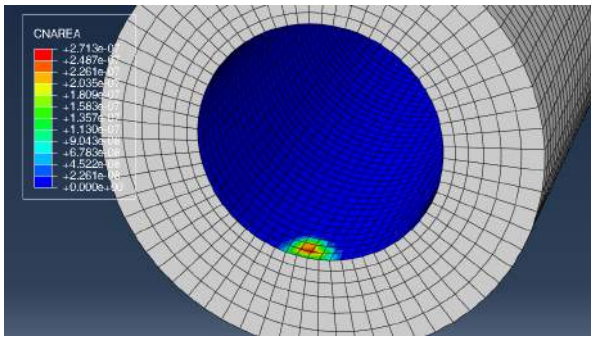


**Figure 5.14:** Bushing contact area at time=1.0

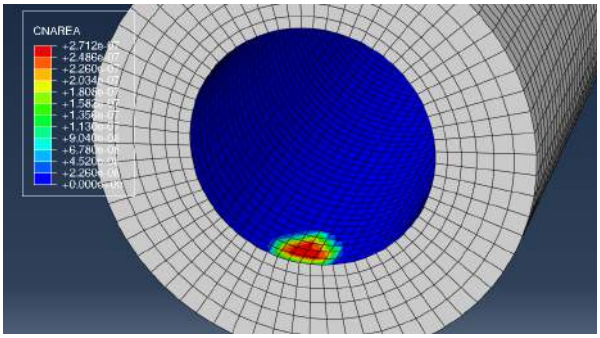


**Figure 5.15:** Bushing contact area at time=2.0

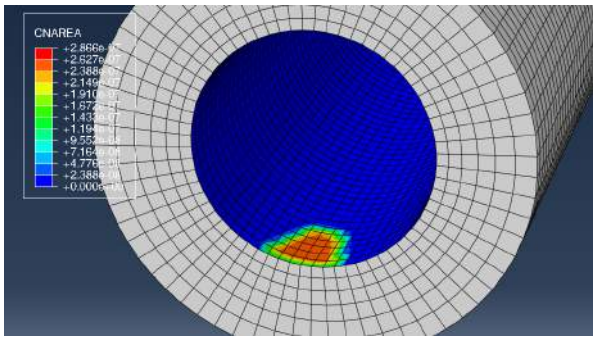


**Figure 5.16:** Bushing contact area at time=3.0

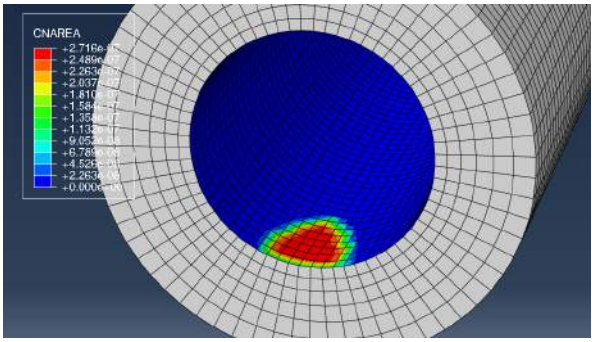


**Figure 5.17:** Bushing contact area at time=4.0

This expansion of the contact area does not happen if the geometry is not changed by material loss during the analysis. Therefore, if only 1 cycle is used the small initial contact area is maintained throughout the rotation step and only a small number of nodes will be shifted in the subsequent wear simulation. Figs. 5.18 and 5.19 show the narrow wear profiles produced using 1 simulation cycle (displacements are again magnified 20 times):

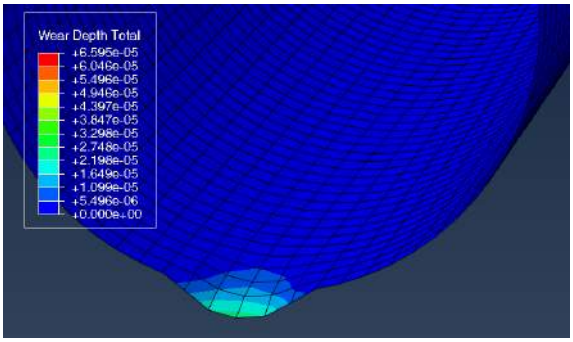


**Figure 5.18:** Wear profile left end, applied in 1 cycle

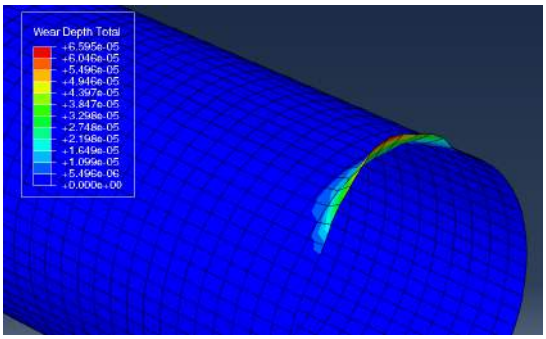


**Figure 5.19:** Wear profile right end, applied in 1 cycle

One observes that the wear profiles shown in Fig. 5.18 and 5.19 are also much steeper than the wear profiles from the reference simulation in Figs. 5.9 and 5.10. Since the contact area is

smaller, the applied force has to be supported by fewer contact nodes that in turn experience a higher contact pressure. The calculated wear depth is correspondingly also larger, leading to a steep wear profile.
The second observation is a consequence of the error that is introduced by repeatedly importing the part's deformed configuration. By importing the deformed coordinates of all nodes, one also acquires the elastic deformations incurred in the previous analysis. However, since the stress state is lost, the next analysis will assume that the imported configuration is in fact undeformed. If the initial force is re-applied in the next cycle to restore the stress state, it will induce more elastic deformation on top of what was obtained from previous cycles. For this particular system, the shaft is being pushed more into the bushing with each new cycle. This will in turn increase the contact area, and thus the wear profile becomes wider and more material is lost. Figs. 5.20 - 5.27 show the wear profiles produced by using 2, 4, 8, and 16 cycles (again magnified 20 times).

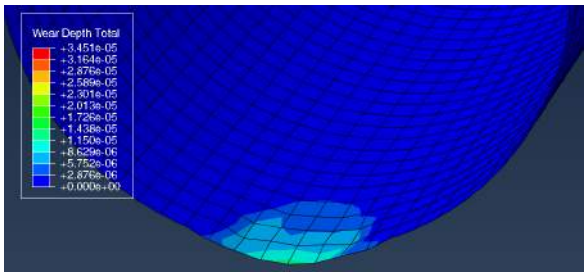


**Figure 5.20:** Wear profile left end, 2 cycles

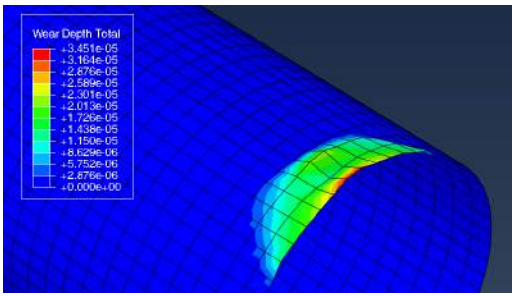


**Figure 5.21:** Wear profile right end, 2 cycles

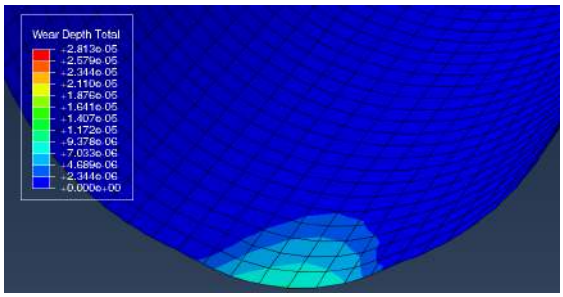


**Figure 5.22:** Wear profile left end, 4 cycles

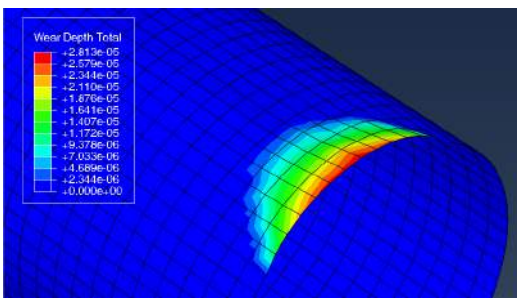


**Figure 5.23:** Wear profile right end, 4 cycles

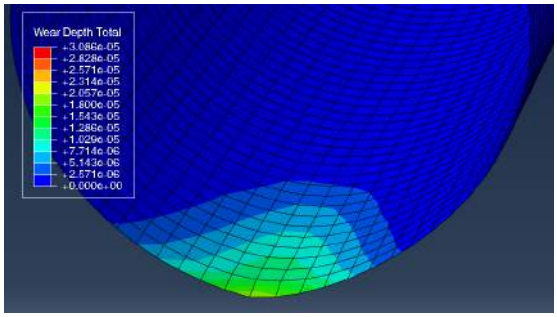


**Figure 5.24:** Wear profile left end, 8 cycles

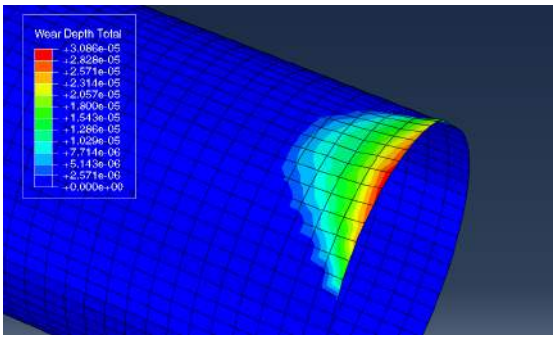


**Figure 5.25:** Wear profile right end, 8 cycles

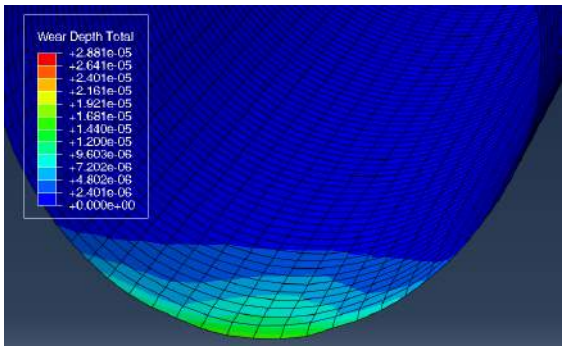


**Figure 5.26:** Wear profile left end, 16 cycles

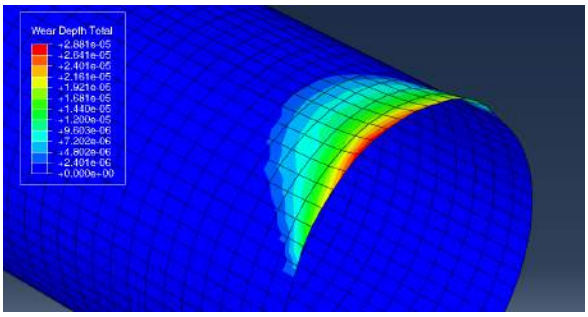


**Figure 5.27:** Wear profile right end, 16 cycles

One observes that the area affected by wear becomes much wider the more cycles are used. At 8 and 16 cycles, the profile looks significantly different from the reference profiles in Figs. 5.9 and 5.10. Conclusively, if the input file method is used for this system it is necessary to somehow reverse the elastic deformations incurred in the previous analysis. Also, one cannot simply use the undeformed configuration since the rigid body motions should be still be transferred to the following steps (specifically, in this case, the rotated position of the shaft).
The most straight-forward method is to introduce an additional step where the forces are removed and the part is allowed to relax such that the shaft is released from its contact with the bushing before node shift occurs. In each “rotation analysis" the following steps are thus performed:

1. *Tilting step*, where forces left and right act on the center of the circular ends of the rod
2. *Rotation step*, where the rod is made to rotate
3. *Release step*, where the forces applied in the tilting step are removed such that the elastic deformations “spring back"

With the additional *release step* one ensures that all elastic deformations are reversed such that the “tilting step" in the next cycle will not induce excessive deformations. Fig. 5.28 shows the volume loss obtained with the modified set-up:

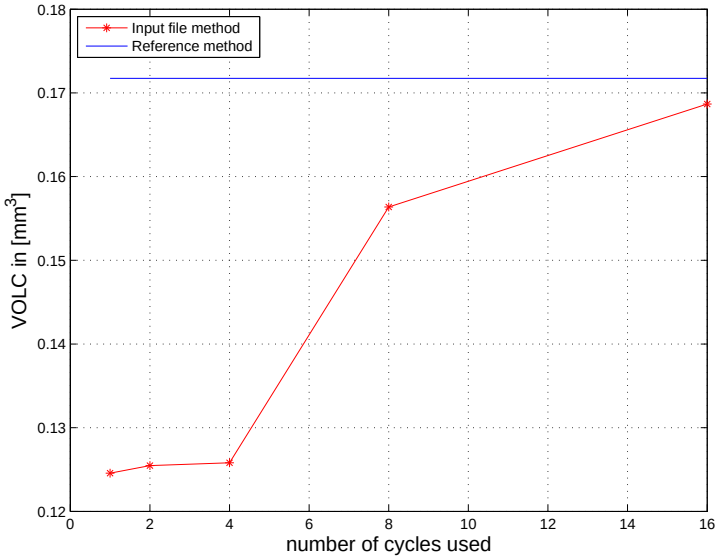


**Figure 5.28:** Material loss volume as more cycles are used (with release step)

Fig. 5.28 shows that the volume loss obtained in this case is consistently lower than the 0.171734 $mm^3$ from the reference simulation. This can be explained by the fact that the contact area is not continuously increased as it should be, but only after each wear simulation analysis. So this is again the effect of the discretization error that was discussed earlier. As more cycles are used the numbers get closer to the reference value, which indicates that it is indeed possible to achieve the correct results with the input file method. The numbers and their relative errors compared to the reference value of the volume loss are shown again in Tab. 5.4:

| Number of cycles used | Reference Value | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| Total volume loss [$mm^{-3}$] | 0.17173 | 0.1255 | 0.1258 | 0.1564 | 0.1687 |
| Relative Error in % | - | 26.94 | 26.74 | 8.94 | 1.79 |

**Table 5.4:** Volume loss and relative errors with increasing numbers of cycles

As one observes in Tab. 5.4, the relative errors compared to the reference value become smaller as more cycles are used. While the numbers do not change much from 2 to 4 cycles, the error drops significantly going from 4 to 8 cycles. At 16 cycles the difference is less than 2%. Interestingly, one observes that at 2 Cycles the error is in fact larger by 14.37% than what was obtained without the added *release step* (see Tab. 5.3). It seems that if elastic deformations are not removed prior to performing the wear simulation, at a certain point the discretization error and the error due to the artificially increased contact area tend to counteract one another.
Figs. 5.29 - 5.36 show that the wear profiles in this case also become increasingly similar to the reference wear profile shown in Figs. 5.9 and 5.10:

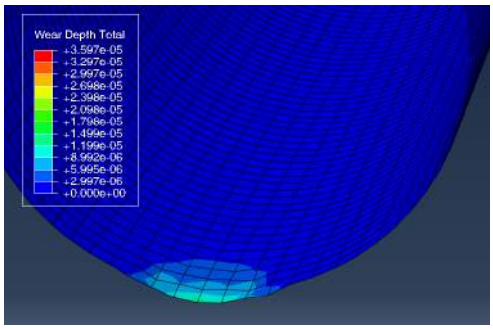


**Figure 5.29:** Wear profile left end with release step, 2 cycles

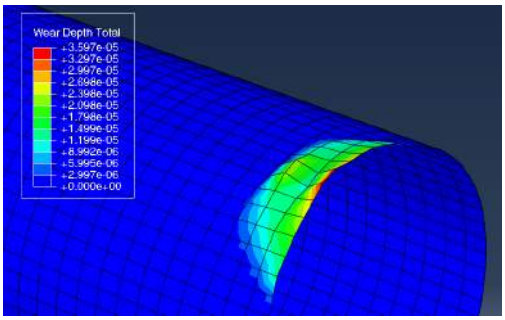


**Figure 5.30:** Wear profile right end with release step, 2 cycles

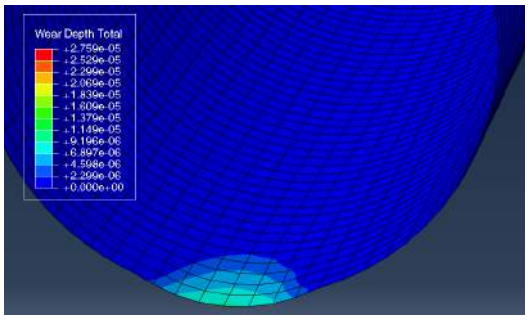


**Figure 5.31:** Wear profile left end with release step, 4 cycles

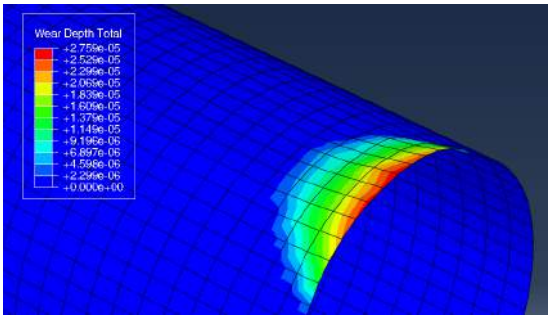


**Figure 5.32:** Wear profile right end with release step, 4 cycles

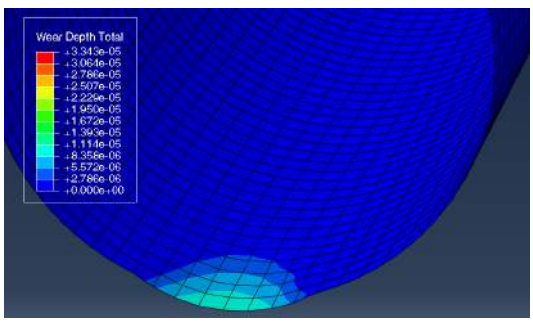


**Figure 5.33:** Wear profile left end with release step, 8 cycles

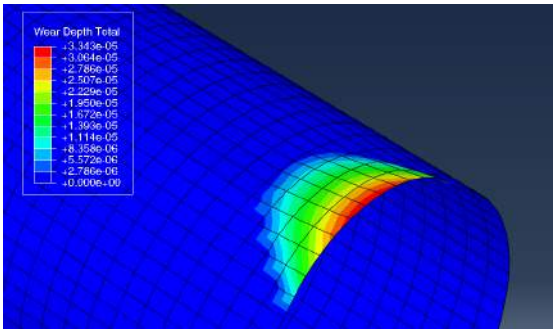


**Figure 5.34:** Wear profile right end with release step, 8 cycles

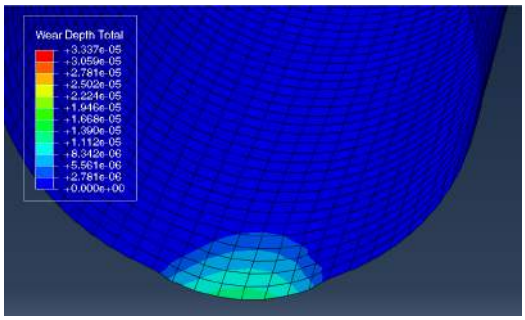


**Figure 5.35:** Wear profile left end with release step, 16 cycles

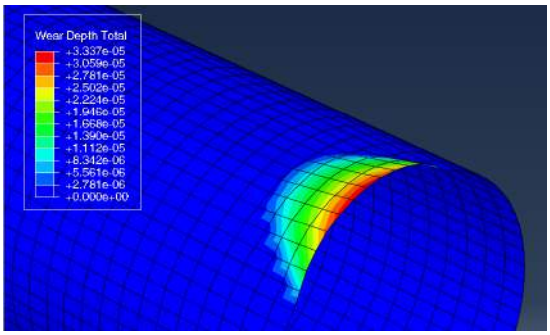


**Figure 5.36:** Wear profile right end with release step, 16 cycles

In the shaft/bushing system, the error introduced by the input file method was easily corrected by releasing the loads at the end of the first simulation. However, in more complicated systems with multiple load cases the problem might be more challenging since the release and restoration of the stress state might be extremely impractical and computationally expensive. For example, if one considers a dynamic system with a complicated load history, it is not clear how to re-establish a contact condition that is similar to the one that existed at the end of the last dynamic cycle.
In many dynamic systems (such as the wastgate device) there is also a considerable transient period where the motion of the individual parts have not yet reached a stable and periodic state. If the system is completely relaxed each time before the wear simulation is performed, in each cycle one would have to wait for that stage to complete first before the contact solutions are recorded again because the transient state would not exist had the loading not been interrupted. For a large models, the additional computation time might be unacceptable.
However, it is perhaps not always strictly necessary to restore the exact stress state. In this particular case, a systematic error was generated due to a mismatch between elastic deformations and the lack of a proper stress state, which repeatedly increased the contact area. If a different model or a different load case is considered, contact between two parts might not be continuous, and thus might not be significantly affected by the imported elastic deformations. In any case one must ensure that contact conditions are not substantially altered when the load case is re-applied in subsequent cycles.

## 5.3 Summary

The purpose of this chapter was to 1. show that all basic calculations had been implemented correctly in UMESHMOTION and the post-processing step, and 2. evaluate the capabilities and limitations of the input file method. In the first instance, it was found that the FE analysis is able to compute a volume loss that agrees with the analytic prediction of a very simple system. While there are slight deviations due to the coding in the subroutine, the error is rather small and within acceptable boundaries.
In the second instance, the input file method was applied to a more complicated model that involved a change in the contact geometry due to the progression of wear. It was shown here that two types of errors are introduced if the wear simulation is not performed in parallel with the actual load case: First, resolution errors are noticeable if the load case is not divided

into sufficiently small cycles. A change in the contact area will naturally affect the contact conditions, so if the geometry is not modified frequently the contact solutions will deviate from their true state. This error is found in any discretized process and can only be reduced by increasing the number of simulation cycles.
Second, it was demonstrated that the input file method contains a deficiency that might introduce a systematic error with each additional simulation cycle. The essential problem is the routine's inability to transfer stress states from one analysis to the next, even though deformations and node displacement are in fact carried over each time. In systems with a simple load case it was shown that this error can be corrected by removing all elastic deformations before performing the wear simulation and re-introducing them at the beginning of the next cycle. Even though the solution might not be as straight-forward in more complicated systems, in most cases the final result might be acceptable if one ensures that the contact conditions are not systematically altered from one cycle to the next. For a simple system and a simple load case, it should not be extremely difficult to adjust the analysis such that severe changes are avoided. As will be explained in the next chapter, an intricate system with an extremely complicated load history might not be suited for wear simulations in general, simply because the computational expense would be unacceptable.

# Chapter 6

# Wear Simulation Results in the Lever/Bushing Interface

Having assessed the performance and shortcomings of the derived simulation method, it is now interesting to observe what the wear simulation will predict if it is applied to a critical interface in the wastegate. As mentioned previously, the study will focus on the interface between the wastegate's lever and bushing since sliding wear is most pronounced at that location and can be suitably represented by Archard's equation. Also, the final simulation results can be qualitatively compared to an existing component with wear damage that was produced by a familiar load case.
Clearly, the results should be most accurate if the existing dynamic simulation of the entire system is used. The simulated motion has been validated with high-resolution camera recordings and should therefore generate the most authentic contact conditions in all interfaces [2]. In that case, the contact solutions obtained from that system would be applied directly to calculate the wear depth for each node. However, it turns out that the dynamic simulation of the wastegate assembly in its basic state is not entirely suitable for the wear simulation. The main difficulties are briefly explained in the next paragraphs.

## 6.1 Wastegate dynamics simulation vs. wear simulation

The most critical problem with the wastegate's dynamic simulation lies in the time scale discrepancy: In order to validate the simulated motion of the wastegate flap with camera recordings of the real part, it was necessary that variations in the high-frequency gaspulsation loads are captured precisely [2]. For that reason, all dynamic simulations are set up to depict the system's behavior within a few crankshaft rotations (i.e. multiples of two rotations, or 720°). At ordinary engine speeds (2000 to 6000 rpm) the dynamic simulation covers an extremely short period of real time. For example, the critical operating condition that causes wear in the lever/bushing interface occurs at 5980 rpm, meaning that one simulation cycle of 2 rotations would describe the system's behavior in 0.02 seconds.

On the other hand, wear is a process that evolves over a very long period of time. The wear profile shown in Fig. 4.2, for example, occurred over an engine endurance run that lasted for 2 weeks ( 330 hours). Of course, if one dynamic simulation cycle can be computed very quickly (e.g. within a few minutes), there would be no problem with using it to obtain the most realistic and accurate contact conditions. However, due to the high frequency of the loading as well as the complexity of the model (as the wastegate contains many individual parts and contact interactions), simulating 4 rotation cycles, or 0.08 seconds, requires approximately 19 hours. If no extrapolation is used, simulating all of 330 hours would require over 32000 years. The relatively simple extrapolation technique introduced in section 4.5 multiplies the calculated wear depth by a certain factor in each cycle, based on a limit that represents the smoothness of the contact surface. Wear is essentially “extrapolated" in only one dimension. However, in most interfaces wear evolves in three dimensions, meaning that one finds an expansion in the affected area as well as an increase in the wear depth (for the lever/bushing interface this effect was shown in Ch. 5.2). To produce a realistic wear profile, the system must therefore always first reach a state where the proper contact area is captured, which requires a large number of simulation cycles despite the extrapolation scheme (as will be shown in the next sections).
Apart from the time problem, the results in this case are not necessarily going to be improved by using extremely precise boundary conditions. There are several reasons for that assumption, as will be discussed in the following.

1. **Incompatibility of boundary conditions**
   Sliding wear occurs at each node where contact pressure and relative slip coincide. If one insists on being as accurate as possible, one should evidently try to match the movement of the E-actuator (which generates the relative sliding between shaft and bushing) to the variation of the gaspulsation load (which generates contact pressure) within the same time duration, i.e. 0.02 - 0.08 seconds. Even if one (reasonably) assumes that the gaspulsation cycles are perfectly repetitive, the same assumption cannot be made for the E-actuator movement. Figs. 6.1 - 6.2 show the measured E-actuator displacement signal that corresponds to the damage found in the lever/bushing interface with increasing time resolution (please zoom in to read the axis labels).

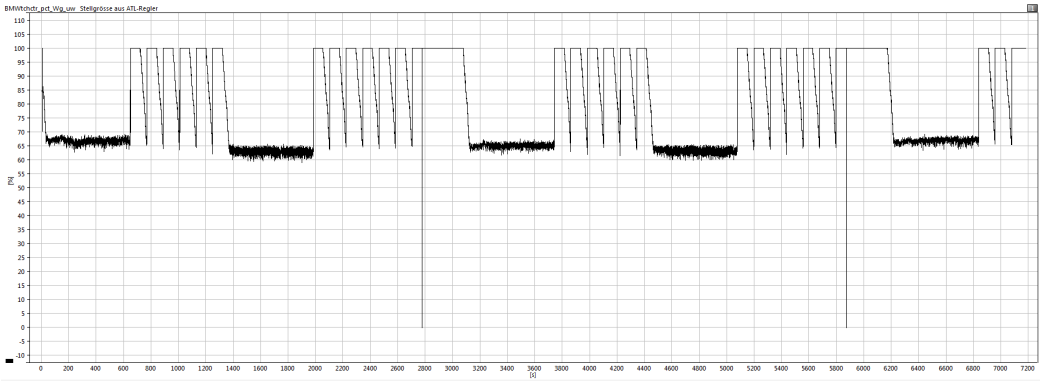


**Figure 6.1:** E-actuator signal over 2 hours

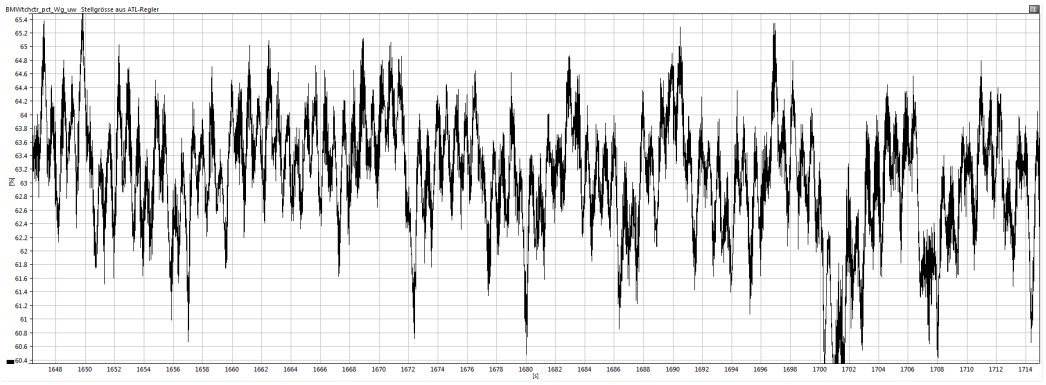

**Figure 6.2:** E-actuator signal over 60 seconds

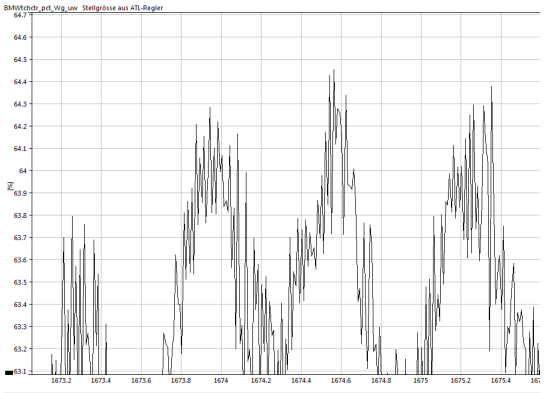

**Figure 6.3:** E-actuator signal over 2 seconds

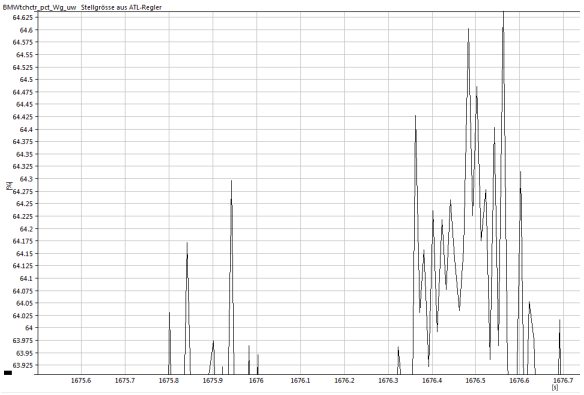

**Figure 6.4:** E-actuator signal over 1 second

Fig. 6.1 shows the complete range of available E-actuator data over 7200 seconds, where the high-frequency blocks describe the critical sliding movement between shaft and bushing. On this zoomed-out scale, it seems that the signal could be periodic within a single block. However, as one continues to zoom in on one of the blocks (in this case the second block), it becomes apparent that the variations are highly irregular even over a relatively long period of one minute, as shown in Fig. 6.2. While a minute seems quite short, it must be noted that the gaspulsation loads are only given for 0.03 % of a minute.

If one zooms in to the scale of a few seconds, as shown in Figs. 6.3 and 6.4, one observes that over a period of 20 milliseconds the signal is reduced to single peaks with very different amplitudes. This means that if one arbitrarily selects a certain peak and applies it together with the gaspulsation signal, it would only be precisely accurate for that particular time period. Of course one could repeat the same peak in the next cycle, but it would not correspond to reality since the E-actuator movement is not repetitive on that tiny scale. Conclusively, any extrapolation of such calculations, or any adaptation of the E-actuator signal, would inevitably introduce errors, thus defeating the purpose of using extremely accurate gaspulsation load variations in the first place.

2. **Insufficient mesh refinement**

For the purposes of the wear simulation, the objective of using precise boundary conditions is to generate very accurate contact solutions such that they can be used to calculate the wear depth. However, accurate contact solutions cannot be achieved if the mesh size is not small enough. The original mesh of the lever and bushing used in the model that took 19 hours to complete is shown in Figs. 6.5 and 6.6.

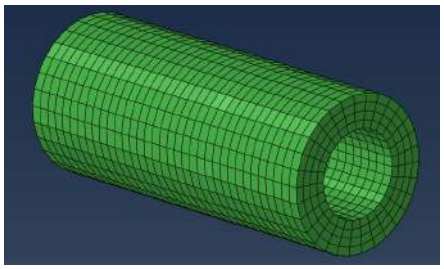

**Figure 6.5:** Mesh for bushing

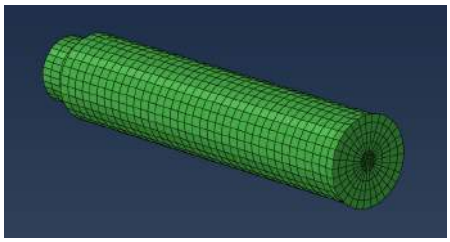

**Figure 6.6:** Mesh for lever

As one observes in Figs. 6.5 and 6.6, both bushing and lever are discretized by 30 elements over its circumference. This means that a single element, or two nodes, would cover more than 8 %, or 12°, of the parts' perimeter. For the E-actuator it was found that a displacement of 0.96 mm - which is approximately the range of the wear-inducing, high-frequency displacements in Fig. 6.1 - produces a rotation of 3.1°. This means that the lever would not be able to rotate over a single element. As mentioned previously, the initial contact area is extremely small, so according to the mesh in Fig. 6.5 and 6.6 the contact would extend over perhaps one or two nodes. This is not to say that the mesh size is not acceptable at all, but in order to achieve the degree of accuracy that one intends to produce by using the precise gaspulsation variations, it is most likely necessary to refine the mesh significantly. However, doing so would most certainly exacerbate the computation time problem.

3. **General uncertainty of wear equations and input parameters**
   The work of the current thesis project is mostly based on values that were estimated by consulting relevant literature. For example, Archard's wear coefficient is obtained from Meyer's experimental study where the wear damage between a wastegate's lever and bushing interface was investigated for different material pairings [3]. The validity of this coefficient is very questionable since it was not found for the same material pairing as the one used for the relevant model at BMW. (It is a very rough estimate based on similar materials.) Also, it is very likely that the wear coefficient will change as the contact surface continues to evolve. In future studies it will definitely be necessary to experimentally determine the variation of the wear coefficient for the specific material pairing. It is clear that the currently assumed wear coefficient might not even be close to the real value. Given this considerable uncertainty it seems unlikely that using very precise boundary conditions would make a significant difference in the accuracy of the results.

The issues discussed above are meant to explain why it is impractical to use the actual loads and boundary conditions of the existing dynamic simulation for the purposes of this project. One could certainly overcome or mitigate all those difficulties with elaborate techniques (e.g.

mesh refinement, submodelling, remeshing etc.), but investigating and implementing them would exceed the scope of this thesis project. Also, the discrepancy between the time scale of the dynamic simulation and the time scale that is required for the wear simulation is still the most critical problem for which there is no straight-forward solution.
In order to derive any meaningful statements about the progression of wear under different circumstances, it is necessary that a representative wear profile is generated within a manageable computation time. The model and the load cases must therefore be simplified considerably. Since sliding wear is most apparent in the wastegate's lever/bushing interface, those parts should be isolated and investigated with a very simple load case that reasonably approximates the effects of the gaspulsation loads and E-actuator movements. The model used in the last chapter to assess the errors of the input file method is designed to have approximately the same dimensions as the lever and bushing of the relevant wastegate model, so it is in fact appropriate to use it for further investigations. First, it is interesting to see if the input file method combined with the extrapolation technique discussed in Ch. 4.5 is indeed able to produce a wear profile that is qualitatively similar to the wear damage found in a real component.

## 6.2 Simulation of sliding wear in the lever/bushing interface

### 6.2.1 Simulation set-up and inputs

To approximate the loading in the lever/bushing interface of the wastegate, two equal but opposite forces are again applied to the ends of the lever. In this case, the magnitude of the forces are calibrated to generate a contact pressure that is similar to the average value found in the dynamic simulation of the full model, which was approximately equal to 50 MPa. It was found that a force F of 238 N on both sides will induce approximately 50 MPa at the single node of the lever that is initially in contact with the bushing. The rotation that induces relative sliding is caused by the E-actuator's oscillatory motion. According to the displacement signals shown in Figs. 6.1 - 6.3, the high-frequency, wear inducing translational movements are limited to an estimated range of 0.96 mm (which is approximately the converted amplitude of the high-frequency signal blocks). For the lever this would correspond to a rotational range of 3.1°, which is applied as a displacement boundary condition at the center node on the left side of the lever (as shown in Fig. 6.7).
Finally, it was mentioned previously that wear damage found in the real part that is available for comparison occurred in an engine endurance run of 330 hours. This means that the simplified model should be able to show the effect of the applied load case in 330 hours. Using the displacement signal of the E-actuator, one can determine the total rotational sliding distance of the lever in that time and apply it as the limit that is to be represented by the simulation.
Assuming that the signal given for 7200 seconds (or 2 hours) is continuously repeated for 330 hours, it is found that a total sliding distance of approximately 371.6 m is induced between the lever and the bushing. Clearly, this number could never be reached without any extrapolation since one cycle only generates 0.245 mm of sliding (corresponding the 3.1° of rotational displacement and circular diameter of 9 mm for the lever). Therefore, is is essential that the extrapolation scheme described in Ch. 4.5 is applied in each post-processing step. It was found that a critical value of 0.1 [mm], or approximately 30% of the element size,

is sufficient for creating a relatively smooth wear profile. The sliding distance after each simulation cycle is assumed to be equal to the extrapolation factor times the 0.245 mm.
The applied loads and boundary conditions are summarized in Fig. 6.7:

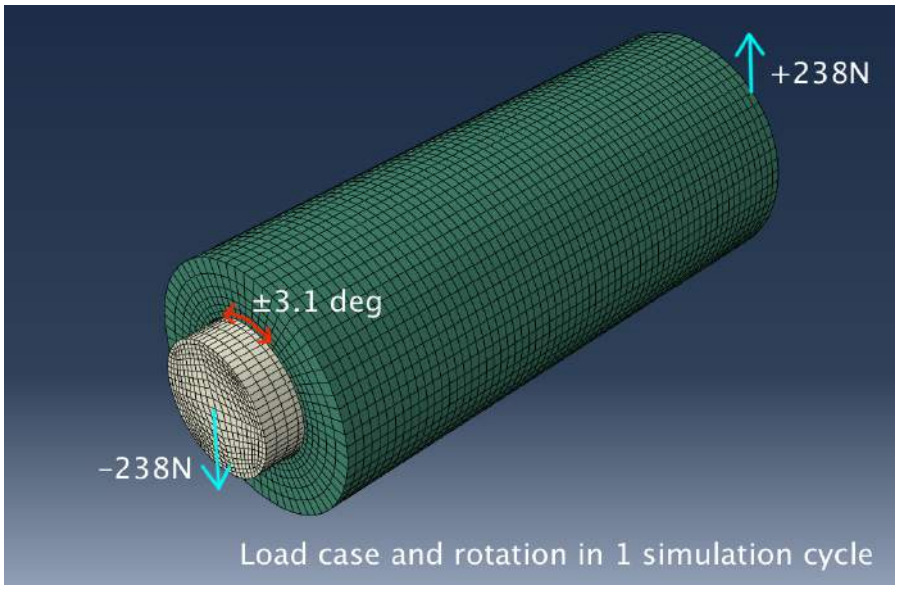


**Figure 6.7:** Applied load and rotational displacement in 1 simulation cycle

Tab. 6.1 summarizes all input parameters that are used for this wear simulation:

| **Load Case Inputs** | |
|---|---|
| External forces for contact [N] | 238.0 |
| Rotational displacement in 1 cycle [deg] | 3.1 |
| Total sliding distance [m] | 371.6 |
| **Material and contact parameters** | |
| E-modulus at 800°C [GPa] | 90.7 |
| Poisson's ratio [-] | 0.3 |
| Estimated friction coefficient $\mu$ [m] | 0.3 |
| Estimated wear coefficient K [$mm^3$/Nm] at 800°C | 2.2E-4 |
| **Critical wear depth difference for extrapolation** [mm] | 0.1 |

**Table 6.1:** Inputs to the wear simulation in the lever/bushing interface

The simulation uses material and contact properties at a temperature of 800°C since it is closest the nominal operating temperature of the wastegate. Also, it is the highest temperature for which Meyer experimentally determined wear coefficients for different material pairings. The applied value of K = 2.2E-4 [$mm^3$/Nm] is an approximate average of the wear coefficients found for eight different material pairings, since the wastegate materials used at BMW could not be matched exactly to the ones studied by Meyer at VW. As mentioned, in future studies it is essential that the appropriate value is determined experimentally for the specific materials under consideration.
The flowchart in Fig. 6.8 clarifies the steps in the simulation routine:

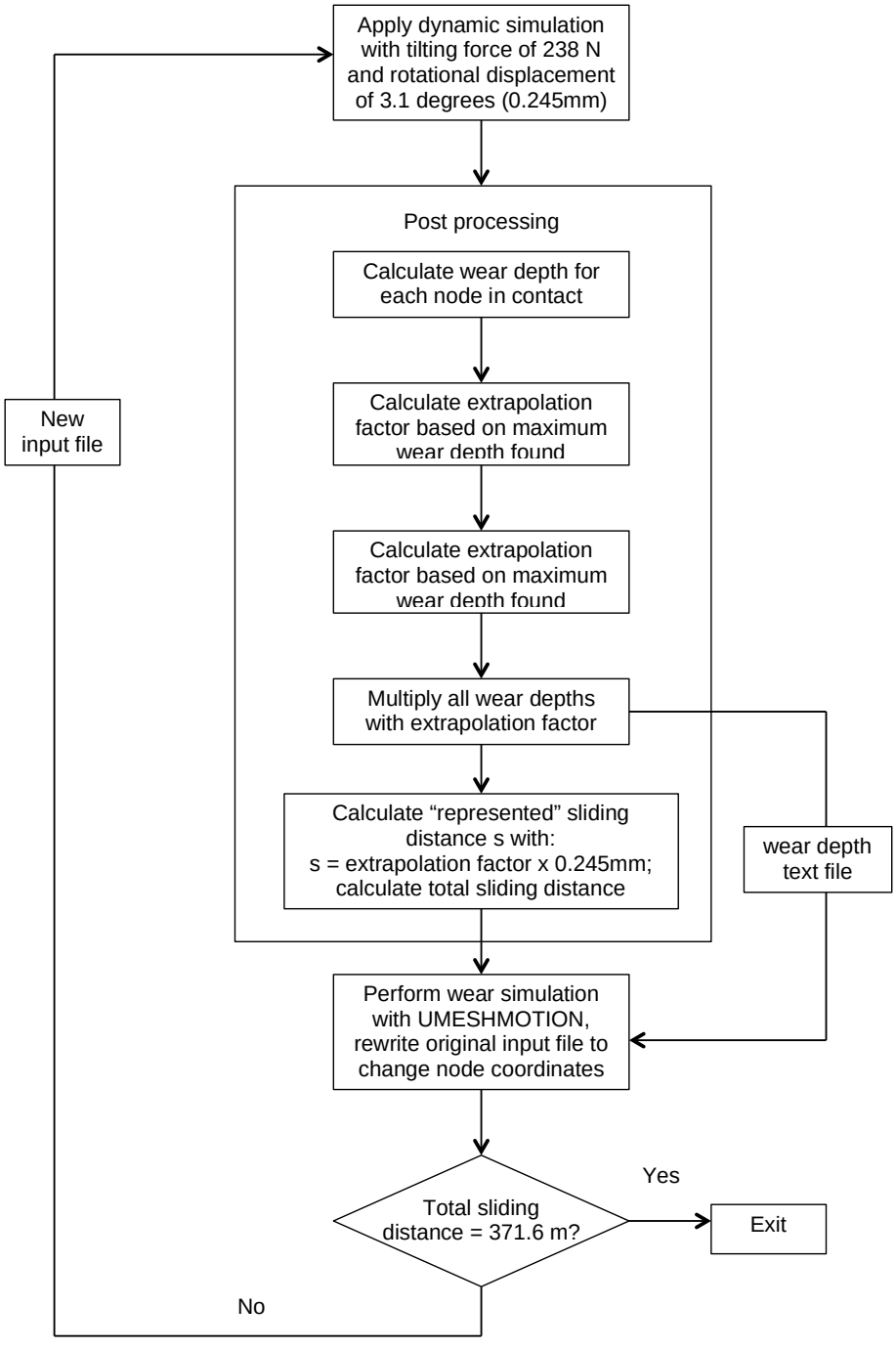


**Figure 6.8:** Flowchart for wear simulation in approximated lever/bushing system

The simulation loop in Fig. 6.8 is executed repeatedly until a total sliding distance of 371.6 m is reached.

### 6.2.2 Results and evaluation of simulated wear profiles

Figs. 6.9 - 6.14 show the lever's wear profile after 211m, 327m, and 371.6 m sliding distance.

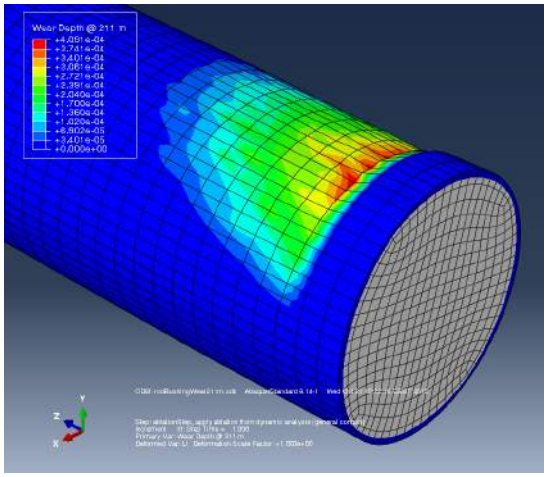

**Figure 6.9:** Wear profile at upper right contact after 211m sliding distance

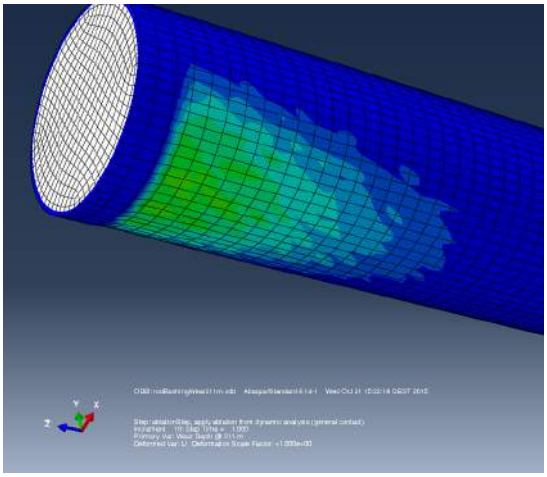

**Figure 6.10:** Wear profile at lower left contact after 211m sliding distance

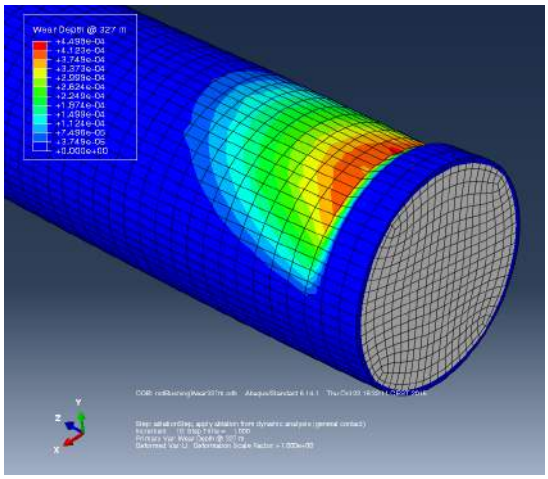

**Figure 6.11:** Wear profile at upper right contact after 327m sliding distance

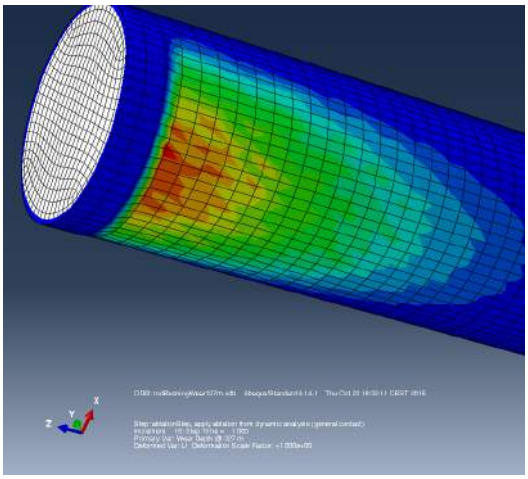

**Figure 6.12:** Wear profile at lower left contact after 327m sliding distance

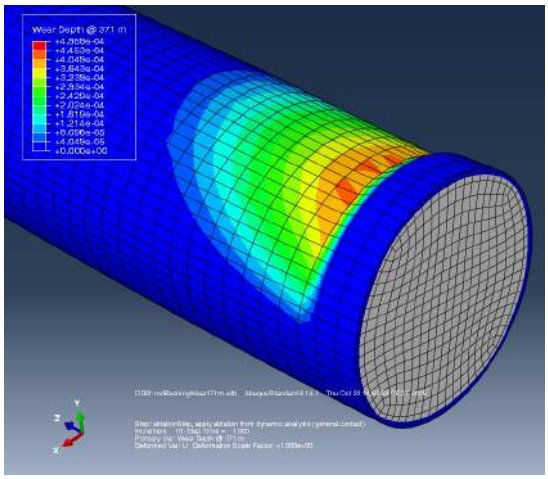

**Figure 6.13:** Wear profile at upper right contact after 371 m sliding distance

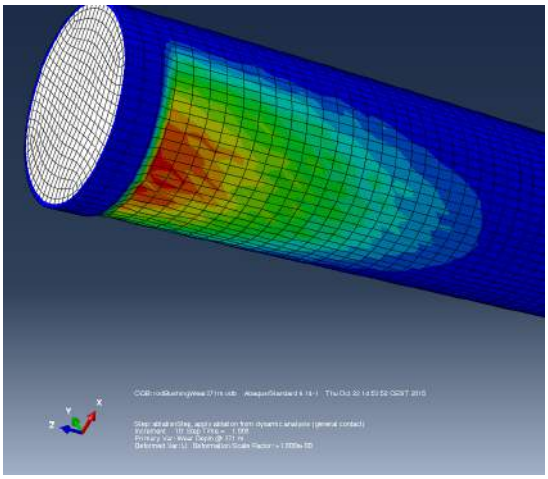

**Figure 6.14:** Wear profile at lower left contact after 371 m sliding distance

The full profile of the lever at 371 m sliding distance is shown in Fig. 6.15:

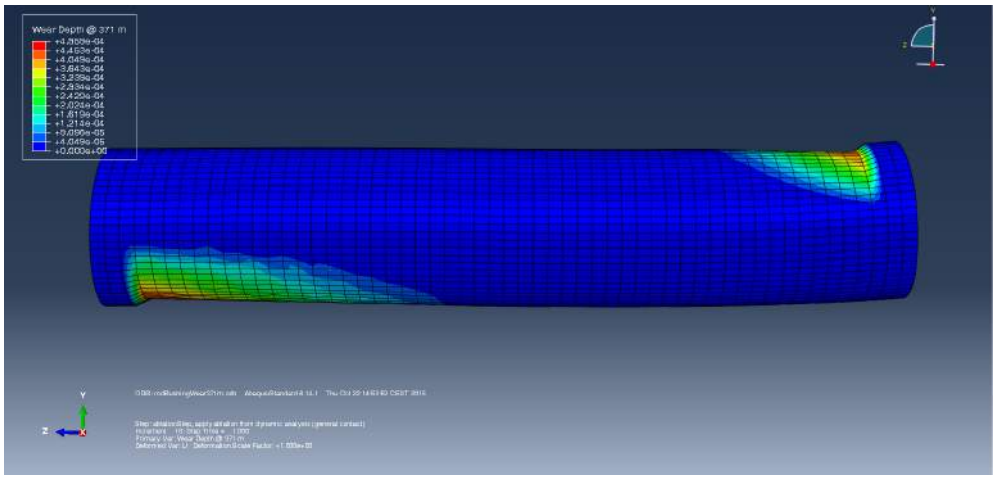

**Figure 6.15:** Side view of lever's final wear profile at 371 m sliding distance

The simulated wear profiles can now be evaluated qualitatively with respect to several properties that can be inferred about the behavior of the wear damage in the real component. The most important conclusions are discussed in the following.

### 1. Shape of simulated wear profile vs. real wear profile

The first notable observation is that the wear profile shown in Fig. 6.15 indeed exhibits a similar shape as the typical wear profile found in Meyer's experimental study. The mirrored image of Fig. 1.4 is shown in Fig. 6.16, and one observes that the slope at the end is similar to the simulated wear profile.


**Figure 6.16:** Wear damage on the surface of a wastegate lever [3]

The similarity in the wear profile confirms that the simulation procedure is indeed capable of reproducing the expected geometry . Given the correct inputs, it might one day be able to predict the wear volume and geometry quantitatively.
Compared to the real wear profile that was generated in the endurance run (from which the relevant sliding distance and loading were derived), one observes that the shape is somewhat different. At the upper right contact, the simulated profile shows a maximum wear depth of approximately 0.45 mm which levels off smoothly towards the other end of the cylinder. In the real part, there is a crater with a nearly uniform depth of 0.70 mm in the same location. The two-dimensional contour of the relevant wastegate lever is shown in Fig. 6.17.

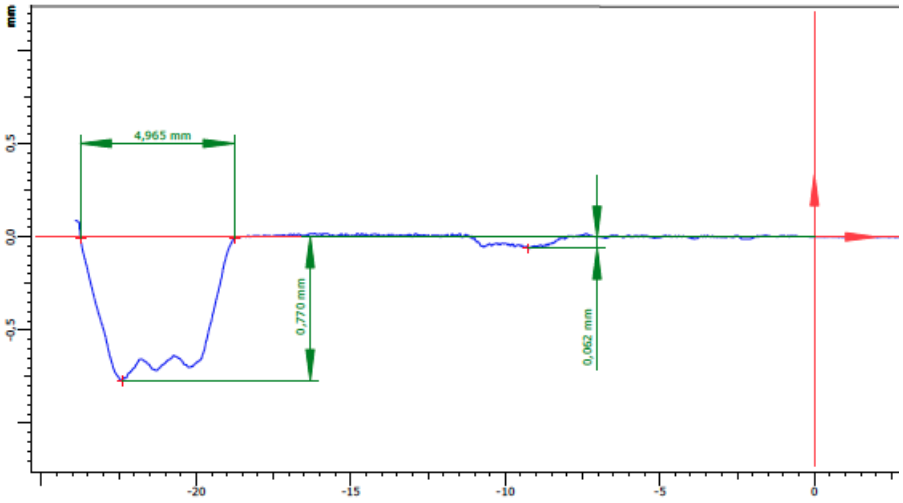


**Figure 6.17:** Measured wear profile of wastegate lever after engine endurance run of 330 hours (zoom in x and y not proportional)

If basic sliding wear is considered only for the lever, one would in fact expect a smooth and uniform slope in the wear profile as produced by the simulation or shown in Fig. 6.16. If material loss occurs gradually, a continuous contact area between lever and bushing should be maintained throughout the wastegate's time in service. However, the distinct “wear crater" in Fig. 6.17 with its clearly defined boundaries implies that more complicated interactions or wear modes could have been effective. Given the relative position of lever and bushing, the discontinuous profile might be a product of:

1. A fracture in the lever's surface, leading to the sudden detachment of a chunk of material as shown in Fig. 6.18:

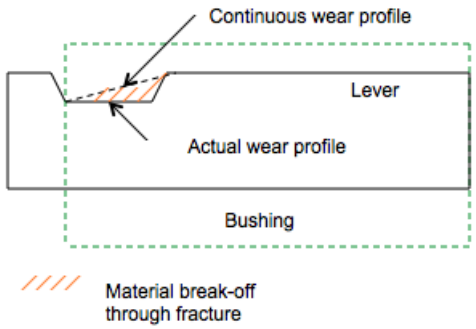


**Figure 6.18:** Sudden material loss due to fracture

While the surface in the wear crater does not look like a typical fracture surface, it is possible that bits of material broke off at some point during the operation and that the subsequent sliding wear then altered the wear surface. Another possibility would be:

2. Wedging of shaft and bushing due to bushing wear and plastic deformations. In the simulation, wear was only considered for the shaft but not for the bushing, though in reality the bushing is also affected to some degree. In addition, plastic deformations might distort the contact area such that a local indentation is created where the contact stresses are quite similar. The concept is shown schematically in Fig. 6.19:

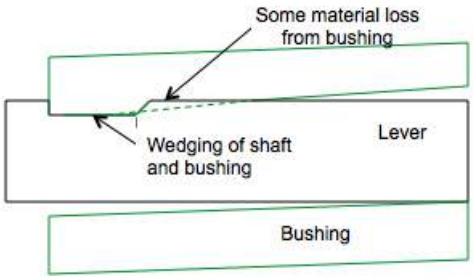


**Figure 6.19:** Wedging of shaft and bushing

At any rate, the discontinuous wear profile found in the real component likely involves other damage and deformation modes in addition to simple sliding wear, and thus would require appropriate considerations in the simulation. Identifying the exact causes in order to reproduce the effects would necessitate a close experimental investigation of the wear damage evolution in the real part, which is beyond the scope of the current thesis project. It would certainly be an appropriate subject for future studies with more time and resources.
Apart from the shape of the wear profile, it is also interesting to investigate how the material loss progresses. In Fig. 6.20 the volume of lost material due to wear is plotted against the sliding distance represented in each simulation cycle:

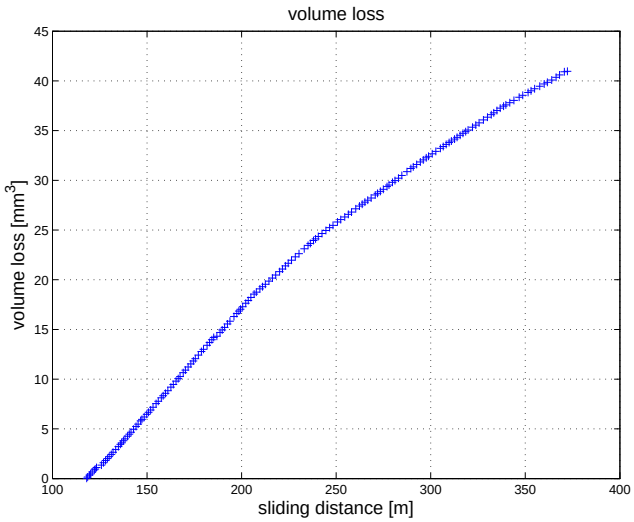


**Figure 6.20:** Volume loss vs. sliding distance

Fig. 6.20 shows that the amount of lost material increases almost linearly with the sliding distance. Since the speed of the E-actuator does not vary significantly, one concludes that the loss of material also increases linearly with time. This result agrees with the observation that the position of the E-actuator is shifted by a constant increment over the duration of the endurance run, since the actuator's control mechanism attempts to adjust for the correct opening angle of the wastegate flap. The linear shift in the E-actuator's position is shown in Fig. 6.21.

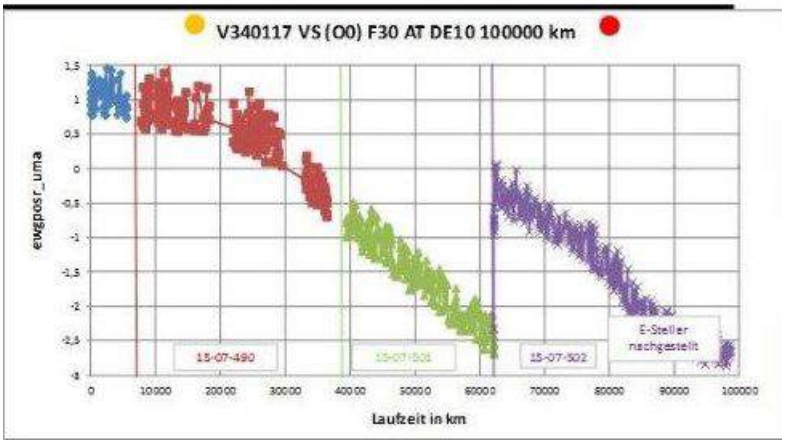


**Figure 6.21:** E-actuator position adjustment due to wear in the lever

## Development of the extrapolation factor

One observes from Figs. 6.10 and 6.9 to Figs. 6.14 and 6.13 that the wear profile becomes increasingly smooth with increasing sliding distance, especially at the upper right contact.

This result is in agreement with reality even though the mesh is not extremely fine, which shows that the selected critical extrapolation limit is reasonable. As the contact area becomes larger, the wear depths at adjacent nodes become increasingly similar, which means that a higher extrapolation factor is calculated towards the end of the simulation. The extrapolation factors are plotted in Fig. 6.22 for each simulation cycle.

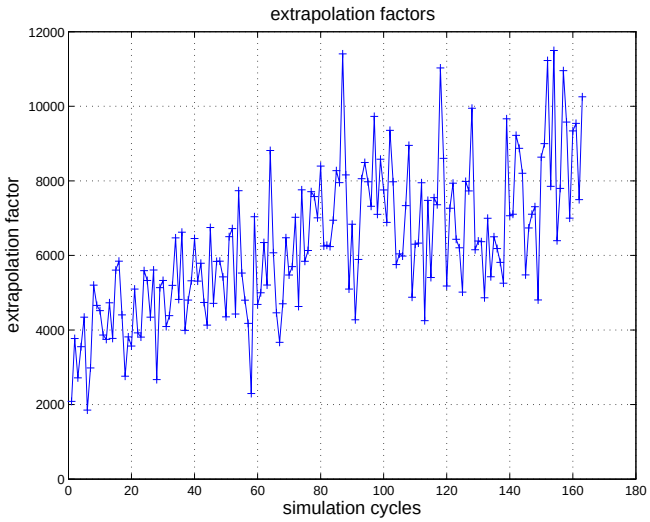


**Figure 6.22:** Extrapolation factor vs. simulation cycle

One observes in Fig. 6.22 that the calculated maximum extrapolation factor varies quite significantly from one simulation cycle to the next. However, there is also a clear upward tendency in later cycles. The simulation is therefore accelerated once a larger contact area is created. This means that with the current extrapolation technique, the bottleneck in the wear simulation lies in establishing a uniform contact area where adjacent nodes experience similar contact conditions. If the initially calculated wear depths have similar magnitudes across each element, the contact conditions are not going to be changed significantly even if a very large extrapolation factor is used. It is therefore reasonable to “pretend" that the same wear depths will be calculated for all nodes if the same simulation is run for thousands of cycles. The prescribed total sliding distance can thus be approached much faster.
Having confirmed that the results are reasonable in a qualitative sense, a parametric study is performed with respect to different wear coefficients and the material’s elastic modulus.

## 6.3 Parametric Study on wear in the lever/bushing interface

Due to a shortage in empirical data, this study is cannot be expected to produce reliable results in terms of absolute numbers. As discussed earlier, certain system specific input parameters - most notably the magnitude and evolution of the wear coefficient - first have to be determined by experiment before the wear simulation can even be validated. Accurate quantitative predictions about e.g. the progression of material loss or the dimensions of the wear profile are only meaningful on that basis.

However, it would certainly be interesting to evaluate the wear simulation in a relative sense: It might be worthwhile to observe how the simulation result will change if certain inputs are varied within reasonable boundaries. Therefore, a parametric study was performed in which the wear coefficient and the elastic modulus of the material are changed (separately) in order to observe their impact on 1. the magnitude and progression of material loss, 2. the wear profile and 3. the simulation speed as implied by the calculated extrapolation factor. Those two quantities were selected since they do in fact tend to vary as the system is subjected to different operation conditions. For example, the elastic modulus of the relevant material varies by more than 30% between 115.9 GPa at 500°$C$ to 73.9 GPa at 1000°$C$. As for the wear coefficient, it is known that this parameter is also very sensitive to small variations in the temperature or the material pairing. In fact, it might even be dependent on the geometry itself, meaning that it may not stay constant as the wear profile continues to evolve under otherwise constant environmental conditions.
The simulation routine in this case is performed until 186 m, or roughly 50% of the previous sliding distance is reached in order to limit computation time to a manageable level. After all, it is not necessary to simulate the entire duration of the endurance run if only a relative comparison between simulation results is required. Apart from the total sliding distance, the wear coefficient and the E-modulus, all other inputs to the lever/bushing system remain unchanged. The next section will first consider the effect of different wear coefficients.

### 6.3.1 Parametric Study I: Increasing wear coefficients

The simulation routine illustrated in Fig. 6.8 is applied to the lever/bushing system with the same inputs as shown in Tab. 6.1 (e.g. E = 90.7 GPa) and the following wear coefficients:

1. K = 2.2E-4 [mm$^3$/Nm]
2. K = 3.1E-4 [mm$^3$/Nm]
3. K = 5.0E-4 [mm$^3$/Nm]
4. K = 7.2E-4 [mm$^3$/Nm]

The values listed above were found in Meyer's experimental study for various contact parings that involve material 1.4848 [3], which is the material used in the relevant BMW B48 turbocharger model. They also represent a reasonable range of magnitudes that induce a noticeable difference in the results.
The progression of material loss during the simulation is considered first. Fig. 6.23 shows the lever's volume loss due to wear as more sliding distance is covered:

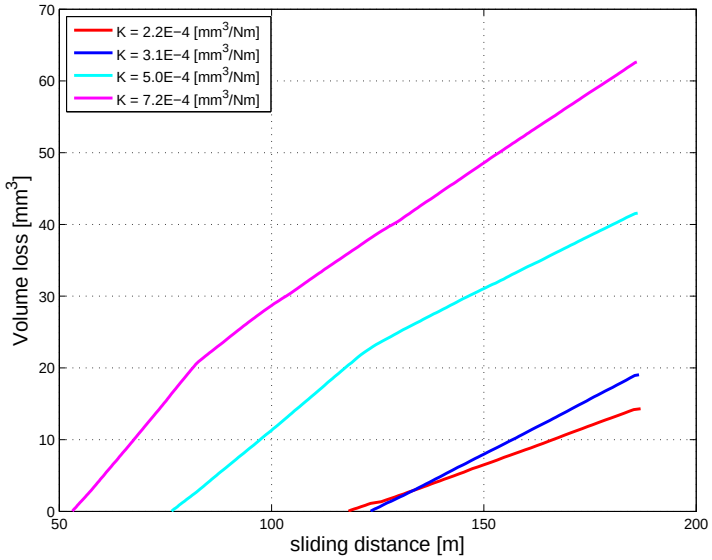


**Figure 6.23:** Volume loss vs. sliding distance for different wear coefficients

One observes in Fig. 6.23 that for all wear coefficients, volume loss is in general proportional to the sliding distance covered by the lever's rotation, despite the fact that the contact area is known to become larger as more material is removed. This also implies that the average wear depth over the wear profile must decrease continuously as more cycles are simulated. The linear relation between the volume loss and sliding distance retrospectively confirm that Archard's simple equation for sliding wear is in fact consistent with the results it produces, even though it is applied to an interaction where the contact area is not constant. The approximately linear relation is also shown in the total wear volume produced with the different wear coefficients. Fig. 6.24 shows the total volume loss at 186 m sliding distance for the values of K that have been considered:

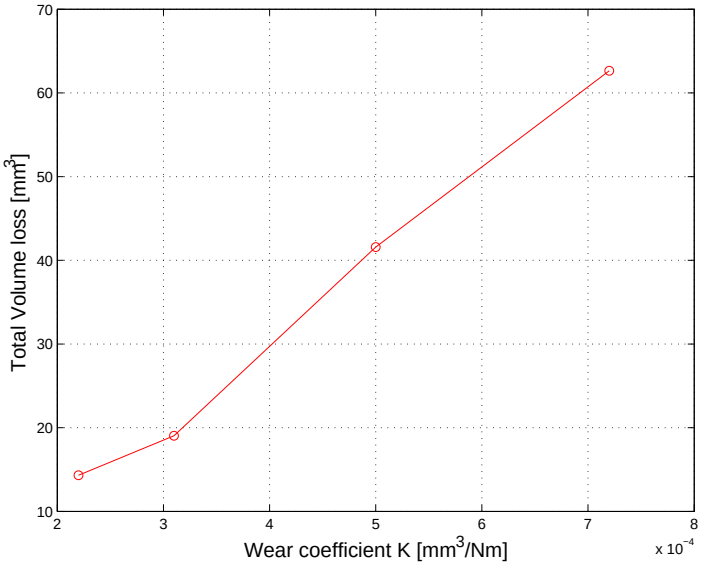


**Figure 6.24:** Total volume loss vs. wear coefficient at 186 m sliding distance

For higher values of the wear coefficient, at K = 5.0E-4 [$mm^3$/Nm] and K = 7.2E-4 [$mm^3$/Nm], one also observes a noticeable “kink" in the two longer graphs in Fig. 6.23, where the rate of volume loss apparently transitions from a higher value to one that is slightly lower. This anomaly might be the effect of an excessively large extrapolation factor: Since the externally applied forces at the ends of the lever are maintained at a constant magnitude of 100 N, the normal force in the contact area should in theory also remain unchanged. According to Archard’s equation, the wear volume per unit sliding distance should in this case also be constant. However, if at some point the extrapolation factor changed the contact geometry significantly, the normal force experienced by individual nodes might be different from that they were in the first simulation cycles. Therefore, a kink the the graphs for volume loss likely indicates that the critical wear depth difference for calculating the extrapolation factor should be decreased.
Another interesting characteristic that can be observed in Fig. 6.23 is that the graphs start at different sliding distance values: For example, at K = 2.2E-4 [$mm^3$/Nm] the first registered number of volume loss corresponds to a sliding distance of 118.11 m, while at K = 7.2E-4 [$mm^3$/Nm] the first value for sliding distance is only 53.1 m. This discrepancy is related to the applied extrapolation method: After the first simulation cycle, a non-zero wear depth is typically only registered at a single node. The calculated wear depth at this node is also very small, such that a large extrapolation factor is required to magnify this value to the critical threshold of 0.1 mm. This means that the first simulation cycle is able to represent a very long sliding distance if the initially calculated wear depth is extremely small. Therefore, it is obvious that a higher wear coefficient would in this case decrease the extrapolation factor, thus increasing the time required to accomplish the prescribed sliding distance. The number of performed simulation cycles and the approximate total simulation time are shown for the respective wear coefficients in Tab. 6.2:

| Wear coefficient K [mm$^3$/Nm] | Number of simulation cycles | Total simulation time [hrs] |
|---|---|---|
| 2.2E-4 | 59 | 14.8 |
| 3.1E-4 | 76 | 19.0 |
| 5.0E-4 | 169 | 42.3 |
| 7.2E-4 | 258 | 64.5 |

**Table 6.2:** Number of simulations and time required for different wear coefficients

Tab. 6.2 shows that the number of simulation cycles and therefore the total required simulation time increases for higher wear coefficients. This outcome makes sense because a large wear coefficient leads to higher values for the immediately calculated wear depths at individual nodes. Since the mesh size is the same in all simulations, a proportionally larger wear depth also leads to more significant differences across a single element. Consequently, the extrapolation factors calculated in each simulation cycle are going to be smaller, meaning that the total sliding distance is approached at a slower rate. The extrapolation factors determined in each simulation cycle are shown in Figs. 6.25 - 6.28. The first entries are again omitted because they are significantly larger than the rest of the numbers and would make the graphs unreadable. Also, the last extrapolation factors are in all cases very small because the total represented sliding distance is programmed to not exceed 186 m.
The graphs are plotted in separate pictures to improve visibility.

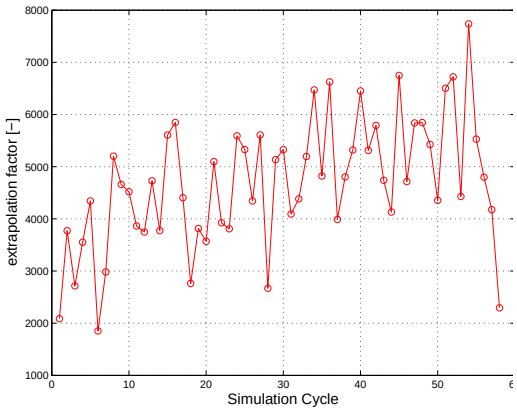


**Figure 6.25:** Extrapolation factors for K = 2.2E-4 [mm$^3$/Nm] calculated in each simulation cycle

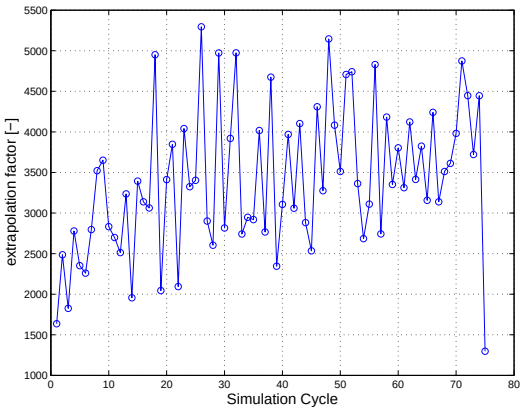


**Figure 6.26:** Extrapolation factors for K = 3.1E-4 [mm$^3$/Nm] calculated in each simulation cycle

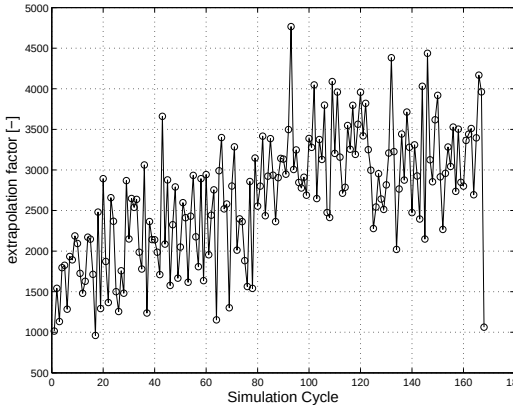


**Figure 6.27:** Extrapolation factors for K = 5.0E-4 [mm$^3$/Nm] calculated in each simulation cycle

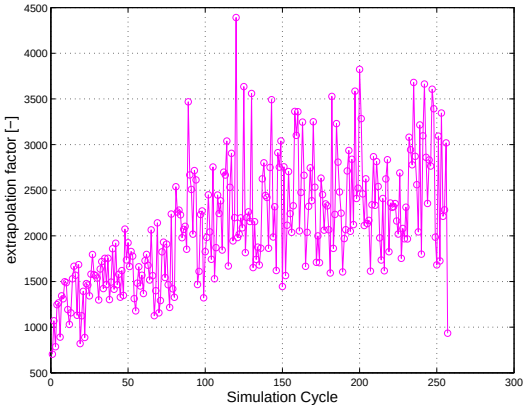


**Figure 6.28:** Extrapolation factors for K = 7.2E-4 [mm$^3$/Nm] calculated in each simulation cycle

One observes in Figs. 6.25 - 6.28 that the extrapolation factors are indeed in general smaller for higher wear coefficients, even though the values vary over a significant range. The numbers again tend to increase in later cycles, which indicates an increasingly uniform wear profile.

### 6.3.2 Parametric Study II: Decreasing elastic modulus

It would also be interesting to see how wear behaves if only the elastic modulus is changed while the wear coefficient stays constant. It is known that the wastegate system usually operates at temperatures ranging from approximately 400 - 1000°C, depending on the component's relative position to the exhaust outlets and the current engine operating condition. For example, Meyer's investigations have shown that at high engine loads, the bushing material right next to the turbine inlet experiences a maximum temperature of 965°C. The lever's temperature at approximately the same position was measured to be 825°C [3] (the temperature is understandably lower since the lever is for the most part covered by the bushing). On the other hand, at the same engine loads temperatures on the other end were found to be 675°C and 640°C for lever and bushing, respectively. The temperature difference therefore varies quite substantially over a single component, which means that the wear damage found at the opposite ends of lever and bushing might also turn out to be dissimilar.
As mentioned in the previous section, Meyer was able to determine the wear coefficients for different material pairings at various temperatures, but none of the investigated material pairings was found to be the same as the ones used for the relevant BMW wastegate system. The impact of the wear coefficient has also been assessed in the last subchapter, so the parametric study is now going to focus on another material property that is known to be influenced by temperature.
The elastic modulus of most metal alloys show a significant decrease from lower to higher temperatures. Tab. 6.3 shows the E-modulus of the 1.4848 material (an austenitic stainless cast steel with high contents of silicon and carbon).

| Temperature [°C] | E-modulus [GPa] |
|---|---|
| 500 | 115.9 |
| 800 | 90.7 |
| 900 | 82.3 |
| 1000 | 73.9 |

**Table 6.3:** E-modulus of material 1.4848 at different temperatures

As one observes in Tab. 6.3, the material's E-modulus at 500°C will decrease up to 36.2% at twice the temperature. A lower E-modulus means that elastic deformations in the part are larger for the same loading. One can imagine that this might affect the contact conditions, in that the contact areas might increase for higher elastic strains. The following parametric study will therefore investigate the wear behavior for the E-modulus numbers listed in Tab. 6.3 at a constant wear coefficient of 2.2E-4 [$mm^3$/Nm]. All loadings and boundary conditions are again given in Tab. 6.1, and the simulation is again extrapolated to a total sliding distance of 186 m.
The graph in Figs. 6.29 and 6.30 show the progression of the wear volume with sliding distance.

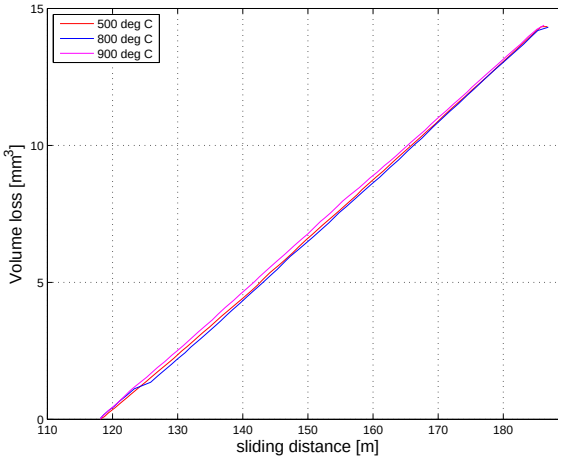


**Figure 6.29:** Volume loss vs. sliding distance for various E-modulus

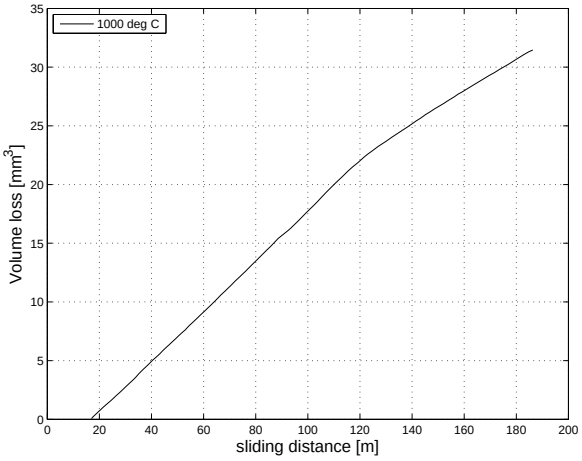


**Figure 6.30:** Volume loss vs. sliding distance for E = 73.9 GPa (at 1000°)

As one observes in Figs. 6.29 and 6.30, the wear volume history for an E-modulus of 82.3 GPa, 90.7 GPa and 115.9 GPa are very similar, but at E = 73.9 GPa the numbers are suddenly much larger. The graph for E = 73.9 GPa had to be plotted separately or else the other three graphs cannot be distinguished from one another. This outcome is rather bizarre if one considers that 73.9 GPa is not much lower than the next highest value (82.3 GPa). In fact, the difference between the 73.9 GPa and 82.3 GPa is lower than the difference between 90.7 GPa and 115.9 GPa. It is therefore quite peculiar that the results should suddenly be so different when E is lowered by another 10 GPa.
This phenomenon is most likely a consequence of the extrapolation scheme. The graphs in Fig. 6.29 all start at approximately the same sliding distance of 118 m, meaning that the

first extrapolation factor was able to advance the simulation by more than 50% of the total distance (this is again because a non-zero wear depth was registered at a single node in the first cycle, and that value was so small that it could be extrapolated to a huge extent before the contact geometry is changed significantly). However, this is not the case at E = 73.9 GPa since the the first extrapolation represents a sliding distance of only 17 m. Tab. 6.4 shows the magnitude of the extrapolation factor in the first cycle in each case:

| E-modulus [GPa] | 115.9 | 90.7 | 82.3 | 73.9 |
|---|---|---|---|---|
| First extrapolation factor [-] | 483069 | 483050 | 483053 | 69420 |

**Table 6.4:** Extrapolation factor in first simulation cycle for different E-moduli

As one observes in Tab. 6.4, the initial extrapolation factors at E = 115.9 GPa - E = 82.3 GPa are all very similar, but the value at E = 73.9 GPa is lower by a factor of 10. This indicates that there is a certain threshold in the E-modulus at which the contact condition in the first simulation cycle becomes significantly different. For example, it is likely that at E = 73.9 GPa the elastic deformations are large enough such that contact is established at more than one node in the first cycle. In that case, the relative slip at any of the contact nodes might be much greater than what is possible if only one node had been in contact. This idea is illustrated in Figs. 6.31 and 6.32 in an exaggerated fashion:

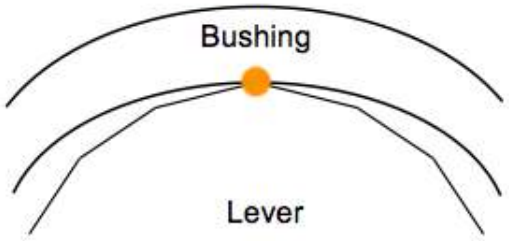


**Figure 6.31:** Small elastic deformation: contact only registered at 1 node

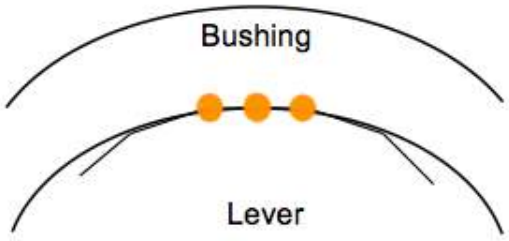


**Figure 6.32:** Large elastic deformation: contact at multiple nodes

If only one node is in contact, as shown in Fig. 6.31, this single contact cannot be maintained

for very long if the rotation does not exceed one element length. This means that the relative slip at the contact node (which is only registered while the node is in contact) is very small, and so the calculated wear depth is also very small. On the other hand, if the elastic deformation is sufficiently large such that a finite area of contact containing multiple nodes is established, relative slip can be maintained for much longer before contact is lost. In that case, the calculated wear depths at one node (such as the center node in Fig. 6.32) might be much larger than at adjacent nodes where contact is lost much sooner, leading to a big difference in the calculated wear depth and a small extrapolation factor.
This outcome shows that one should perhaps re-evaluate the mesh refinement in relation to the prescribed rotation range. In theory, even if there is only a point contact, this point contact should be transferred to different nodes as the lever rotates in the bushing. On the other hand, a finer mesh might not necessarily have an impact on the large difference in the extrapolation factors, since the wear depth differences are still expected to be very small with this adjustment (because all nodes that successively come into contact during the rotation should experience a similar relative slip distance). It is therefore possible that the outcome might indeed correspond to reality in some way. There might really be a lower limit in the E-modulus where the total incurred volume loss at the end of the complete simulation is suddenly much greater, due to the slightly larger contact area in the beginning. It is possible that at higher values of the E-modulus, a very large number of rotation cycles is in fact required to increase the tiny contact area. Once a larger contact area is established, the material loss occurs much faster (meaning that much more material is lost per cycle than at the beginning of the process).
The extrapolation factors calculated after the first simulation cycle are shown in Figs. 6.33 - 6.36 for the different E-moduli:

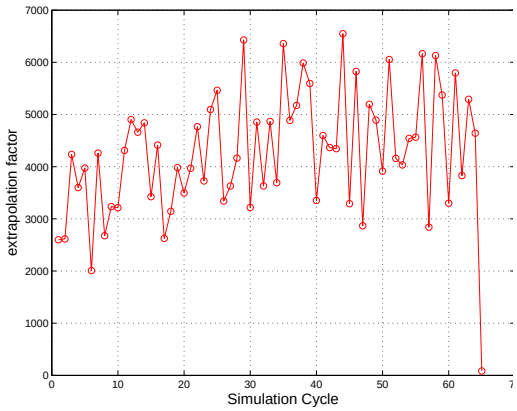


**Figure 6.33:** Extrapolation factors for E = 115.9 GPa calculated in each simulation cycle

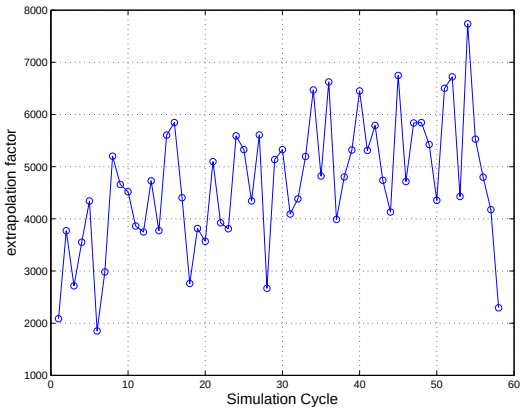


**Figure 6.34:** Extrapolation factors for E = 90.7 GPa calculated in each simulation cycle

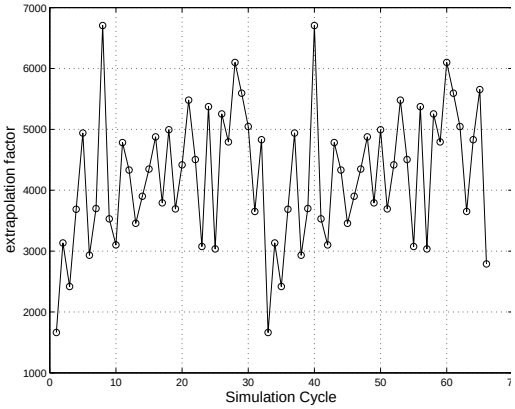


**Figure 6.35:** Extrapolation factors for E = 82.3 GPa calculated in each simulation cycle

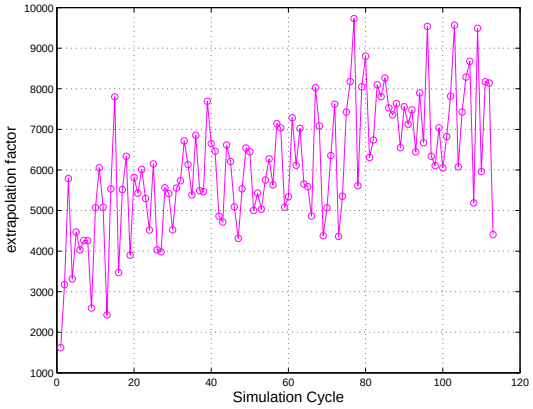


**Figure 6.36:** Extrapolation factors for E = 73.9 GPa calculated in each simulation cycle

As one observes in Figs. 6.33 - 6.36, the extrapolation factors calculated after the first cycle are not significantly different in each case (at least not as different as the initial values shown in Tab. 6.4). Tab. 6.5 shows the average extrapolation factors in each case from the 2nd cycle:

| E-modulus [GPa] | 115.9 | 90.7 | 82.3 | 73.9 |
|---|---|---|---|---|
| Average extrapolation factor [-] | 4198 | 4608 | 4160 | 6059 |

**Table 6.5:** Average extrapolation factor from 2nd to last cycle

As one observes in Tab. 6.5, the average extrapolation factor at E = 73.9 GPa is in fact larger than at higher E-moduli, but it is not in an entirely different order of magnitude. This results shows again that the total volume loss at the end of the simulation depends very much on the first significant cycle, i.e. the first cycle in which a significant difference in the wear depth is registered. If this cycle occurs early, for example in case E = 73.9 GPa, then the process is able to produce appreciable volume loss over a very long time period. If the first significant cycle occurs much later, wear damage will have a much shorter time to develop (which is also shown in Figs. 6.29 and 6.30).
In conclusion, this parametric study has shown that the extent of wear in the lever bushing interface is highly dependent on the size of the initial contact area, and therefore on the magnitude of the initial elastic deformations.

## 6.4 Summary

In this chapter, the derived wear simulation routine was applied to a simplified shaft/bushing assembly that was supposed to represent the lever/bushing interaction in the wastegate system. First, it was shown that it is indeed possible to generate a similar wear profile as the one found in Meyer's empirical study. The input file simulation routine can thus be used in

future studies to predict wear quantitatively if material specific wear coefficients are available. Compared to the wear profile of the lever that was subjected to a similar load case at BMW, the simulated profile was found to be more idealized, i.e. more smooth and continuous. This discrepancy was attributed to the presence of more unconventional damage modes, such as wedging or fracture, that were not taken into consideration in the wear simulation. It was also found that the linear progress of material loss corresponded with the linear repositioning of the electric actuator that was necessary to adjust for an optimum wastegate opening angle as wear damage continues to change the lever's geometry.
The second part of this chapter involved a parametric study with respect to the wear coefficient and the material's elastic modulus at various operating temperatures. In the first instance it was found that a higher wear coefficient indeed leads to increased total volume loss at the end of the simulation. In addition, it was observed that the simulation time is also proportional to the wear coefficient. This is mainly due to the applied extrapolation scheme, as the extrapolation factors are smaller if a greater wear depth is calculated in each simulation cycle.
The second parametric study assessed the impact of the material's E-modulus. It was found that for E-moduli at temperatures of 500°C - 900°C the magnitude and progression of material loss are only very slightly different, while at 1000°C (with E = 73.9 GPa) a sudden and significant increase was observed. This phenomenon was caused by an earlier onset of appreciable volume loss: Since the initial contact area between lever and bushing is larger for greater elastic deformations, it takes fewer cycles to change the initial geometry significantly. Wear thus had a longer time to develop, which led to more material loss at the end of the process. This observation showed that the extent of wear damage might be closely related to the initial contact area.
This chapter thus concludes the most important work done in this thesis project. In the following, Ch. 7 will present the conclusions and discuss the most essential topics that need to be addressed in future studies.

# Chapter 7

# Conclusions and Recommendations

This chapter presents the main conclusions of this thesis project and identifies the most critical problems in need of further study and improvement. The outcome of this assignment, its main scientific contribution and its significance to the client company are first going to be clarified in the following section.

## 7.1 Conclusions and Relevance of Simulation Methodology

The main conclusions of the current study are described in the following:

1. Over the course of this thesis project a comprehensive methodology was developed for the simulation of wear in a general dynamic system. The selected approach combines Abaqus' integrated adaptive meshing technique with an automated routine that uses Abaqus/CAE to repeatedly import new model definitions, modify their features, and submit them for analysis. Changes in the geometry created within a certain load cycle are determined in post-processing and transferred to the subsequent wear simulation analysis. This method makes it possible to investigate the effect of an arbitrary number of load cycles, and can also be adjusted to execute until certain criteria are met. The use of a post-processing step also facilitates the implementation of an extrapolation scheme to expedite the simulation of a damage process that, in reality, develops over a duration that otherwise would be very impractical for FE calculations. The wear simulation further includes measures to account for the proper definition of wear directions at geometric discontinuities such as edges and corners. Those procedures make it possible to achieve a valid representation of the wear-induced geometric changes in generic three-dimensional parts with arbitrary surface orientations.

2. The simulation so far only considers sliding wear as defined by Archard's model, since sliding wear between the wastegate's lever/bushing interface was found to produce the most severe wear damage in the system. Other types of wear can be included in the

routine without changing the overall structure of the script as it is only necessary to modify the wear-depth calculation procedures in the post-processing step.

3. The derived wear simulation routine is accessible and easy to use, but was found to contain an error that arises due to the loss of the stress state from one analysis to the next. This error could be corrected for simple systems by reversing the imported elastic deformations in order to limit the changes in the contact area, but it was acknowledged that the correction might be less straight-forward for more complicated interactions or load cases. It is therefore concluded that one must at least ensure that changes in the contact conditions due to import are eliminated.

4. By applying the simulation routine to a simplified system that is representative of the lever/bushing assembly of the wastegate, it was demonstrated that a qualitatively reasonable wear profile can be produced. Therefore, one can reasonably assume that accurate quantitative predictions could be achieved if appropriate wear coefficients are found. A parametric study using the same model definition and load case showed that the simulation time depends on the magnitude of the wear coefficient, and that the total volume loss is related to the material's elastic modulus.

5. The current simulation routine provides a quick an easy way to assess the impact of different input parameters on the development of sliding wear for simple systems. For example, users at the client company may wish to compare the results produced with different wear coefficients and determine the inputs that generate the outcome that is most similar to reality. (This is of course only reasonable if reliable wear coefficients are found in advance.) Also, the routine can be easily expanded to include other wear models, additional contact interactions, or more advanced extrapolation schemes. The significance of the work in the current study is therefore that it produced a generic, flexible wear simulation procedure that can be augmented in several ways. The most critical augmentations that should be considered in future studies are proposed in the next paragraphs.

## 7.2 Recommendations for future studies

The current study evidently was not able to address all relevant issues related to the simulation of wear in turbocharger wastegates. In addition, this report has shown that there are in fact several problems that could not be fully solved and therefore require more extensive investigation. The first part of this chapter will therefore discuss improvements that directly relate to the work done in this study, and the second part will suggest more advanced topics for consideration such that one can eventually achieve a comprehensive wear simulation that is more specific to the wastegate system.

### 7.2.1 Improvements on work done in the current study

There are three notable issues that have been addressed but not fully explored over the course of this assignment:

1. As Ch. 5.2 has shown, the input file method will introduce errors to the contact solution if no adjustments are made to remove the imported elastic deformations. If the load case is not periodic but follows a more complicated pattern, it is likely that there is no easy solution to correct for that error without affecting the continued motion of the system. To improve the validity of the input file method, methods to transfer the stress state should be investigated in more detail. For example, it might be possible to import the latest stress state from a previous analysis. It is not immediately clear whether this is possible with a mesh that has been modified by adaptive meshing, but it is certainly a technique that is worth exploring.

2. Secondly, Ch. 4 discussed the many difficulties that eventually prevented the implementation of a successful restart routine. A simulation method based on restart would in fact not introduce any errors as it is the most “natural" way to continue the same analysis after some kind of post-processing has been performed. The stress state would be carried over from one step to the next by default. Though it is certainly difficult to automate the restart routine, the simulation could be accomplished with more in-depth research.

3. And finally, one could try to find different ways to apply the wear simulation to the entire wastegate system. The small time scale of the existing dynamic simulation that nevertheless requires an extremely long computation time certainly poses a great challenge. It must be acknowledged, however, that the most accurate contact behavior in a certain interface can only be obtained if the motion of the entire system is taken into account. One could possibly explore different methods such as submodelling or remeshing in certain intervals. It might also be interesting to see how the changed lever geometry might retroactively affect the dynamic motion of the complete wastegate system. For that purpose, individual parts that have been modified with the wear simulation routine must be inserted back into the assembly.

### 7.2.2 General topics to be considered in more advanced studies

To continue the research on wastegate wear simulations productively, it is recommended that prospective investigators consider the following suggestions:

1. The wear simulation must be connected more directly to specific experimental data and observations. In order to achieve a more targeted simulation, material and system-specific information must be obtained. For example, it is likely that the wear coefficient will vary at different stages of the wear process, or that certain unusual damage mechanisms are activated somewhere along the way. Only by closely observing the behavior of a specific interaction is it possible to determine what kind of wear phenomena the simulation is supposed to reproduce. For instance, it was shown in Ch. 6 that the wear profile of the lever subjected to sliding wear in the engine endurance testing is noticeably different from the ideal wear profile that should in theory result from simple sliding wear. Therefore, it is important to find out what exactly led to the irregular wear profile. If e.g. wedging or plastic deformations are found to significantly affect the wear behavior, the simulation routine should include those effects. A validation of the numerical results is only possible if all relevant processes have been accounted for.

2. If one intends to include the more precise, high-frequency gas pulsation loads as well as a more accurate description of the actuator movement, it is necessary to accelerate the wear simulation with more advanced extrapolation schemes. So far, the extrapolation is performed in one dimension, meaning that it is only possible to enhance the wear depth at nodes where it already has a non-zero value. This limits the speed of the extrapolation as the technique is not intended to change the contact area itself. If a method can be found that is able to extrapolate the wear depth incurred at individual nodes to adjacent nodes, the wear simulation could become considerably faster. It will certainly be challenging to find a reasonable way to determine the magnitude of the "virtual" wear depths and to validate the procedure with a real wear profile.

3. Finally, the the simulation routine should be expanded to include additional wear models and equations. For example, it is known that impact wear is very important in the flap/lever interface as it leads to material loss as well as plastic deformations. An appropriate impact wear equation should exhibit the same simplicity and dependency on FE solution variables as Archard's equation.

# References


[1] J.K. Miller. *Turbo - Real World High-Performance Turbocharger Systems*, volume 1. CarTech Inc., 2008.

[2] M. Wibmer, T. Schmidt, O. Grabherr, and B. Durst. Simulation of turbocharger wastegate dynamics. *Motortechnische Zeitschrift (MTZ)*, 76(2):28–31, February 2015.

[3] K. Meyer. *Hochtemperatur-VerschleiÃ§verhalten der Wastegate-Lagerung von Abgasturboladern fÃijr Otto-Motoren.* Ph. d. dissertation, Otto-von-Guericke-UniversitÃďt Magdeburg, October 2011.

[4] G.W. Stachoviak and A.W. Batchelor. *Engineering Tribology.* Butterworth-Heinemann, 4 edition, 2013.

[5] P. Podra and S. Andersson. Simulating sliding wear with finite element method. *Tribology International*, 32:71–81, 1999.

[6] A. SÃűderberg and S. Andersson. Simulation of wear and contact pressure distribution at the pad-to-rotor interface in a disk brake using general purpose finite element analysis software. *Wear*, 267:2243–2251, 2009.

[7] V. Hegadekatte, N. Huber, and O. Kraft. Finite element based simulation of dry sliding wear. *Modelling and Simulation in Materials Science and Engineering*, 13:57–75, 2005.

[8] E.M. Bortoleto, A.C. Rovani, V. Seriacopi, F.J. Profito, D.C. Zachariadis, I.F. Machado, A. Sinatora, and R.M. Souza. Experimental and numerical analysis of dry contact in the pin on disk test. *Wear*, 301:19–26, 2013.

[9] F.J. Martinez, M. Canales, S. Izquierdo, M.A. Jimenez, and M.A. Martinez. Finite element implementation and validation of wear modelling in sliding polymer-metal contacts. *Wear*, 284:52–64, 2012.

[10] H Czichos and K. Habig. *Tribologie-Handbuch: Tribometrie, Tribomaterialien, Tribotechnik*, volume 3. Vieweg + Teubner Verlag, 2010.

[11] G. Banish. *Engine Management*, volume 1. CarTech Inc., 2007.

[12] J.A. Williams. *Engineering Tribology*, volume 1. Oxford University Press, 1994.

[13] R. Holm. *Electric Contacts*, volume 4. Springer, 1967.

[14] J.F. Archard and W. Hirst. The wear of metals under unlubricated conditions. *Proceedings of the Royal Society of London*, 236(1206):397–410, 1956.

[15] M. Oquist. Numerical simulations of mild wear using updated geometry with different step size approaches. *Wear*, 249:6–11, 2001.

[16] A. Rezaei, W. Van Paepegem, P. De Baets, W. Ost, and J. Degriek. Adaptive finite element simulation of wear evolution in radial sliding bearings. *Wear*, 296:660–671, 2012.

[17] N.H. Kim, D. Won, D. Burris, B. Holtkamp, G.R. Gessel, P. Swanson, and W.G. Sawyer. Finite element analysis and experimens of metal/metal wear in oscillatory contacts. *Wear*, 258:1787–1793, 2005.

[18] C. Mattheck. *Design in Nature*. Springer, 1998.

[19] Dassault Systemes. *Abaqus User's Manual*, 6-14 edition, 2014.

[20] P.D. Lax and B. Wendroff. Difference schemes for hyperbolic equations with high order of accuracy. *Communications on Pure and Applied Mathematics*, 17:381–398, 1964.

[21] Dassault Systemes. *Abaqus User Subroutines Reference Guide*, 6-14 edition, 2014.

[22] Dassault Systemes. *Abaqus Example Problems Manual*, 6-14 edition, 2014.

[23] T. Schmidt. *Mischreibung und VerschleiÃ§ in Hydraulikdichtsystemen - Modellbildung, Simulation und experimentelle Analyse*. Ph. d. dissertation, Gottfried Wilhelm Leibniz UniversitÃďt Hannover, November 2011.

[24] Dassault Systemes. *Abaqus Verification Guide*, 6-14 edition, 2014.

[25] Dassault Systemes. *Abaqus/CAE User's Guide*, 6-14 edition, 2014.

[26] Dassault Systemes. *Abaqus Scripting Reference Guide*, 6-14 edition, 2014.

[27] S. Mukras, N.H. Kim, W.G. Sawyer, D.B. Jackson, and W.B. Lawrence. Numerical integration schemes and parallel computation for wear prediction using finite element method. *Wear*, 266:822–831, 2009.

# Appendix A

# General Wear Simulation Script

## A.1 Python Code for Input File Wear Simulation Method

```
"""
Gen.py

general wear simulation script

- parameterized for input files with unique node and element labels
- must be run in Abaqus/CAE
- runs umeshmotion_simplified2

"""
import time
from abaqus import*
from abaqusConstants import*
backwardCompatibility.setValues(includeDeprecated=True, reportDeprecated=
    False)


startTime = time.time()
print 'Start Time: ', startTime

inputFileInput = getInput(prompt='Input File (no file extension): ')
inputFile = inputFileInput

firstRun = 1
numCycle = 0
relativeSliding = 0.0
numCycles = [1]
VOLC = 0.0
totalSliding = 100 #some big number

while relativeSliding < totalSliding:
```

```
31
32          #SECTION I: initial analysis (dynamic analysis)
33    #create model from input file
34
35          numCycle = numCycle + 1
36
37          if firstRun == 1:
38
39              myModel = mdb.ModelFromInputFile(name=inputFile+str('
                    DynamicWearModel'), inputFileName= inputFile+str('.inp'))
40
41              #get surfaces
42
43              surfaces = myModel.rootAssembly.surfaces.keys()
44              surfaceNumbers = [0]*len(surfaces)
45              surfCount1 = 0
46              for i in range(len(surfaces)):
47                  surfCount1 = surfCount1 + 1
48                  surfaceNumbers[i] = surfCount1, surfaces[i]
49
50              surfCount2 = 0
51              for surf in surfaces:
52                  surfCount2 = surfCount2 + 1
53                  print str(surfCount2)+'. '+ surf + '\n'
54
55              #get element sets
56
57              instanceKey = myModel.rootAssembly.instances.keys()
58
59              elementSets = myModel.rootAssembly.instances[instanceKey[0]].
                    sets.keys() + myModel.rootAssembly.sets.keys()
60              allExistingNodes = myModel.rootAssembly.instances[instanceKey
                    [0]].nodes
61              numberAllInitialNodes = len(allExistingNodes)
62
63
64              elsetNumbers = [0]*len(elementSets)
65              elsetCount1 = 0
66              for i in range(len(elementSets)):
67                  elsetCount1 = elsetCount1 + 1
68                  elsetNumbers[i] = elsetCount1, elementSets[i]
69
70              elsetCount2 = 0
71              for elset in elementSets:
72                  elsetCount2 = elsetCount2 + 1
73                  print str(elsetCount2)+'. '+ elset + '\n'
74
75
76              surfaceString = 'Ablation Surface(s): ' + '\n' + '(Select
                    number from message area)'
77              aleDomainString = 'ALE adpative domain(s): '  + '\n' + '(
                    Select number from message area)'
78
```

```
79              fieldStrings = [['Total real sliding distance',''],['wearStep
                    ',''], [surfaceString,''], ['Wear Coefficient K: ',''],
80                              ['Critical Wear Depth Difference: ',''], [
                                    aleDomainString,''], ['Simulation Cycles','
                                    ''], ['extrapolation Factor',''],
81                              ['Relative Sliding Distance in 1 Cycle', ''],
                                    ['Tie Node Label',''']]
82
83              inputData = getInputs(fields=fieldStrings, dialogTitle ='
                    Input Data')
84
85              ErrorMessage = 'Error: Enter <"Real sliding distance" and "
                    Critical Wear Depth Difference"> OR <"Simulation cycles"
                    and "Extrapolation Factor">'
86
87        if inputData[0] != '' and inputData[6] != '':
88              warning = getWarningReply(message=ErrorMessage, buttons=(YES,
                    NO,CANCEL))
89              break
90        elif inputData[0] != '' and inputData[7] != '':
91              warning = getWarningReply(message=ErrorMessage, buttons=(YES,
                    NO,CANCEL))
92              break
93        elif inputData[4] != '' and inputData[6] != '':
94              warning = getWarningReply(message=ErrorMessage, buttons=(YES,
                    NO,CANCEL))
95              break
96        elif inputData[4] != '' and inputData[7] != '':
97              warning = getWarningReply(message=ErrorMessage, buttons=(YES,
                    NO,CANCEL))
98              break
99
100       if inputData[8] != '':
101             relativeSliding1Cycle = float(inputData[8])
102
103       if inputData[0] != '' and inputData[7]=='' :
104             totalSliding = float(inputData[0])
105       elif inputData[7] != '' and inputData[0]=='' :
106             totalSliding = relativeSliding1Cycle*int(inputData[6])*int(
                    inputData[7])
107
108       surfaceKeys = inputData[2].split(',')
109       for i in range(len(surfaceKeys)):
110           surfaceKeys[i] = int(surfaceKeys[i])
111
112       ablationSurfaces = []
113       for number in surfaceKeys:
114           ablationSurfaces.append(surfaceNumbers[number-1][1])
115
116       # ALE adaptive Domains for later
117
118       elsetKeys = inputData[5].split(',')
119       for i in range(len(elsetKeys)):
```

```
120             elsetKeys[i] = int(elsetKeys[i])
121
122         aleElementDomains = []
123         for number in elsetKeys:
124             aleElementDomains.append(elsetNumbers[number-1][1])
125
126         # assume that only relevant steps are included in input file
127
128         #get history output for specified surfaces
129
130         surfaceNodes = [0]*len(surfaceKeys)
131         surfaceNodesSet = [0]*len(surfaceKeys)
132         for i in range(len(ablationSurfaces)):
133         surfaceNodes[i] = myModel.rootAssembly.surfaces[ablationSurfaces[i
                ]].nodes
134         surfaceNodesSet[i] = myModel.rootAssembly.Set(name=ablationSurfaces
                [i]+'Set', nodes=surfaceNodes[i])
135
136         historyItemCount = 0
137         for i in range(len(surfaceNodesSet)):
138         historyItemCount = historyItemCount + 1
139         myModel.HistoryOutputRequest(createStepName=inputData[1],
140         frequency=1, name='ContactHistory'+str(historyItemCount), rebar=
                EXCLUDE, region=
141         surfaceNodesSet[i-1], sectionPoints=DEFAULT, variables=('CSTRESS',
                'CDISP', 'CAREA'))
142
143         dynamicJob = mdb.Job(name=inputFile+str('_dynamic1'), model=myModel
                , numDomains = 8, numCpus=8)
144         dynamicJob.writeInput()
145         myModel.setValues(noPartsInputFile=ON)
146         dynamicJob.submit()
147         dynamicJob.waitForCompletion()
148
149
150     # subsequent cycles
151           elif firstRun != 1:
152
153               myModel = mdb.ModelFromInputFile(name=inputFile+str('
                      DynamicWearModel'), inputFileName= inputFile+str('_mod.inp
                      '))
154
155               #get history output for specified surfaces
156
157               surfaceNodes = [0]*len(surfaceKeys)
158               surfaceNodesSet = [0]*len(surfaceKeys)
159               for i in range(len(ablationSurfaces)):
160         surfaceNodes[i] = myModel.rootAssembly.surfaces[ablationSurfaces[i
                ]].nodes
161         surfaceNodesSet[i] = myModel.rootAssembly.Set(name=ablationSurfaces
                [i]+'Set', nodes=surfaceNodes[i])
162
163               historyItemCount = 0
```

```
164             for i in range(len(surfaceNodesSet)):
165         historyItemCount = historyItemCount + 1
166         myModel.HistoryOutputRequest(createStepName=inputData[1],
167             frequency=1, name='ContactHistory'+str(historyItemCount),
                rebar=EXCLUDE, region=
168             surfaceNodesSet[i], sectionPoints=DEFAULT, variables=('
                CSTRESS', 'CDISP', 'CAREA'))
169
170             dynamicJob = mdb.Job(name=inputFile+str('_dynamic'), model=
                myModel, numDomains = 8, numCpus=8)
171             myModel.setValues(noPartsInputFile=ON)
172         dynamicJob.writeInput()
173         dynamicJob.submit()
174         dynamicJob.waitForCompletion()
175
176
177     #
        ##############################################################################
178
179
180     ##SECTION II: Post-processing
181
182     #Calculate Ablation Depths
183
184     from odbAccess import *
185     from odbMaterial import*
186     from odbSection import*
187     from itertools import chain
188     import numpy as np
189
190     if firstRun == 1:
191           odbPath = inputFile+str('_dynamic1.odb')
192     else:
193           odbPath = inputFile+str('_dynamic.odb')
194
195     odb = openOdb(path=odbPath)
196
197     allNodesDynamic = odb.rootAssembly.instances['PART-1-1'].nodes
198
199     # !! if there is a coupling node:
200
201     if inputData[9] != str(''):
202           missingNodeDefinition = []
203           for node in allNodesDynamic:
204                if node.label == int(inputData[9]):
205                      missingNodeDefinition.append(' ' + ' ' + ' ' + str(
                          node.label) +',' +  ' ' + str(node.coordinates
                          [0]) +
206                   ',' + ' ' + str(node.coordinates[1]) +',' + ' ' + str(
                       node.coordinates[2]))
207     # !!!!
208
```

```
209
210   ablationNodeSets = [0]*len(ablationSurfaces)
211   for i in range(len(ablationSurfaces)):
212       ablationNodeSets[i]= ablationSurfaces[i]+'SET'
213
214   contactNodes = [0]*len(ablationNodeSets)
215   for i in range(len(ablationNodeSets)):
216       contactNodes[i] = odb.rootAssembly.instances['PART-1-1'].nodeSets[
              ablationNodeSets[i]].nodes
217
218   allContactNodes = contactNodes[0]
219   for i in range(len(contactNodes)-1):
220       allContactNodes = allContactNodes + contactNodes[i+1]
221
222   allContactNodeLabels = [node.label for node in allContactNodes]
223
224
225   # assign variables to all frames in odb:
226
227   histPointContact = []
228   for i in range(0, len(allContactNodes)):
229     histPointContact.append(HistoryPoint(node=allContactNodes[i]))
230
231
232   ContactRegionAll = []
233   for i in range(0, len(histPointContact)):
234     ContactRegionAll.append(odb.steps[inputData[1]].getHistoryRegion(
          point=histPointContact[i]))
235
236
237   #
          #------------------------------------------------------------------------------
238
239
240   cpressData= []
241   contactNodeData = []
242   cslip1Data = []
243   cslip2Data = []
244
245   outputKeys = ContactRegionAll[0].historyOutputs.keys()
246
247   for i in range(0, len(ContactRegionAll)):
248     cpressData.append(ContactRegionAll[i].historyOutputs[outputKeys[1]].
          data)
249     cslip1Data.append(ContactRegionAll[i].historyOutputs[outputKeys[4]].
          data)
250     cslip2Data.append(ContactRegionAll[i].historyOutputs[outputKeys[5]].
          data)
251
252   def f(t):
253     if type(t) == list or type(t) == tuple:
254       return [f(i) for i in t]
```

```
255     return t
256
257
258
259   cpressData_list = f([list(i) for i in cpressData])
260   cslip1Data_list = f([list(i) for i in cslip1Data])
261   cslip2Data_list = f([list(i) for i in cslip2Data])
262
263
264   #Compute incremental cslip and avg. cpress
265
266   cslip1_delta = [[[cslip1Data_list[k][i+1][j] - cslip1Data_list[k][i][j]
          for j in range(len(cslip1Data_list[k][i]))]
267     for i in range(len(cslip1Data_list[k])-1)] for k in range(len(
          cslip1Data_list))]
268
269   cslip2_delta = [[[cslip2Data_list[k][i+1][j] - cslip2Data_list[k][i][j]
          for j in range(len(cslip2Data_list[k][i]))]
270     for i in range(len(cslip2Data_list[k])-1)] for k in range(len(
          cslip2Data_list))]
271
272   cslip_delta_mgnt = [[[cslip1_delta[k][i][0], np.sqrt((cslip1_delta[k][i
          ][1])**2 + (cslip2_delta[k][i][1])**2)]
273     for i in range(len(cslip1_delta[k]))] for k in range(len(cslip1_delta
          ))]
274
275   cpress_avg = [[[cpressData_list[k][i+1][0] - cpressData_list[k][i][0],
          0.5*(cpressData_list[k][i+1][1] + cpressData_list[k][i][1])]
276     for i in range(len(cpressData_list[k])-1)] for k in range(len(
          cpressData_list))]
277
278   #Compute ablation depths
279   K = float(inputData[3])
280
281   ablation_delta = [[[cslip1_delta[k][i][0], K*cpress_avg[k][i][1]*
          cslip_delta_mgnt[k][i][1]]
282     for i in range(len(cslip1_delta[k]))] for k in range(len(cslip1_delta
          ))]
283
284   ablation_delta_noTime = [[[ablation_delta[k][i][1]] for i in range(len(
          ablation_delta[0]))] for k in range(len(ablation_delta))]
285
286
287   ablation_total = []
288   for k in range(len(ablation_delta_noTime)):
289     ablation_total.append(sum(ablation_delta_noTime[k]))
290
291
292   #
          #-------------------------------------------------------------------------------
293
294   # find extrapolation factor
```

```
295
296   if inputData[4] != '':
297         nodesAndAblation_numbers = [0]*2
298         nodesAndAblation_numbers[0] = [allContactNodeLabels[i] for i in
                range(len(allContactNodeLabels))]
299         nodesAndAblation_numbers[1] = [ablation_total[j] for j in range(
                len(ablation_total))]
300
301         # get element connectivity
302
303         contactElements = [0]*len(ablationSurfaces)
304
305         for i in range(len(ablationSurfaces)):
306         contactElements[i] = odb.rootAssembly.surfaces[ablationSurfaces[i
                ]].elements[0]
307
308         allContactElements = contactElements[0]
309         for i in range(len(contactElements)-1):
310       allContactElements = allContactElements + contactElements[i+1]
311
312
313         connectedContactNodes = [[0]*8] * len(allContactElements)
314         connectedContactNodes_corrected = [0] * len(allContactElements)
315         for i in range(len(connectedContactNodes)):
316     connectedContactNodes[i] = allContactElements[i].connectivity
317     connectedContactNodes_corrected[i] =  [connectedContactNodes[i][k]
            for k in range(len(connectedContactNodes[i])) if
            connectedContactNodes[i][k] in allContactNodeLabels]
318
319         ablationIndex = [0]*len(connectedContactNodes_corrected)
320         for i in range(len(connectedContactNodes_corrected)):
321     ablationIndex[i] = [nodesAndAblation_numbers[0].index(
            connectedContactNodes_corrected[i][j]) for j in range(len(
            connectedContactNodes_corrected[0]))]
322
323
324         ablationElements = [0]*len(ablationIndex)
325         for i in range(len(ablationIndex)):
326     ablationElements[i] = [nodesAndAblation_numbers[1][j] for j in
            ablationIndex[i]]
327
328
329         ## obtain ablation depth difference
330
331         ablationElementDelta = [0]*len(ablationElements)
332         for i in range(len(ablationElements)):
333       ablationElementDelta[i] = [abs(ablationElements[i][0] -
              ablationElements[i][-1]), abs(ablationElements[i][1] -
              ablationElements[i][0]),
334       abs(ablationElements[i][2] - ablationElements[i][1]), abs(
              ablationElements[i][3] - ablationElements[i][2]),
335       abs(ablationElements[i][2] - ablationElements[i][0]), abs(
              ablationElements[i][3] - ablationElements[i][1])]
```

```
336
337
338         ## check if ablation difference exceeds critical value
339
340         ablationDeltaCritical = float(inputData[4])
341
342         maxAblationDifferenceElements = []
343         for i in range(len(ablationElementDelta)):
344       maxAblationDifferenceElements.append(max(ablationElementDelta[i]))
345
346         maxAblationDifference = max(maxAblationDifferenceElements)
347
348         # Difference between max. ablation difference and critical value:
349
350         diffToCritical = ablationDeltaCritical - maxAblationDifference
351
352         if maxAblationDifference != 0:
353       maxScalingFactor = floor(ablationDeltaCritical/
              maxAblationDifference)
354       print 'maxScalingFactor '+str(numCycle)+ ':', maxScalingFactor
355         else:
356       maxScalingFactor = 1
357       print 'maxScalingFactor '+str(numCycle)+ ':', maxScalingFactor
358
359         if relativeSliding > totalSliding:
360                   maxScalingFactor = ceil((totalSliding-slidingTemp)/
                        relativeSliding1Cycle)
361                   print 'maxScalingFactor_corrected: ', maxScalingFactor
362
363         # calculate scaled wear depths
364
365   elif inputData[4] == '':
366         maxScalingFactor = int(inputData[7])
367
368
369
370         relativeSliding = relativeSliding + (maxScalingFactor*
              relativeSliding1Cycle)
371
372   print 'relativeSliding '+str(numCycle)+ ':', relativeSliding
373
374
375   # general contact
376
377   if maxScalingFactor > 1.0:
378       for i in range(len(ablation_total)):
379     ablation_total[i] = (ablation_total[i] * maxScalingFactor)
380
381   # combine node labels with ablation data
382
383   nodesAndAblation = []
384   for l in range(len(allContactNodeLabels)):
```

```
385     nodesAndAblation.append(str(allContactNodeLabels[l]) + ' ' + ' ' + '
            ' + str(ablation_total[l]))
386
387   writeFile = open('ablation.txt','w')
388   writeFile2 = open('ablation'+str(numCycle)+'.txt','w')
389
390   for nodeAblation in nodesAndAblation:
391     writeFile.write(nodeAblation + '\n')
392     writeFile2.write(nodeAblation + '\n')
393
394   writeFile.close()
395   writeFile2.close()
396
397
398   #
          ------------------------------------------------------------------------------
399
400   # Generate element connectivity file and adaptive constraint nodes file
          in 1st cycle
401   if firstRun == 1:
402         elementConnFile = open('elementConnectivity.txt','w')
403         aleConstraintNodesFile = open('aleConstraintNodes.txt','w')
404
405         ablationElements = [0]*len(ablationSurfaces)
406         ablationNodes = [0]*len(ablationSurfaces)
407         for i in range(len(ablationSurfaces)):
408           ablationElements[i] = odb.rootAssembly.surfaces[
                ablationSurfaces[i]].elements[0]
409           ablationNodes[i] = odb.rootAssembly.surfaces[ablationSurfaces[i
                ]].nodes[0]
410
411         allAblationElements= ablationElements[0]
412         for i in range(len(ablationElements)-1):
413       allAblationElements = allAblationElements + ablationElements[i+1]
414
415         allAblationNodes= ablationNodes[0]
416         for i in range(len(ablationNodes)-1):
417       allAblationNodes = allAblationNodes + ablationNodes[i+1]
418
419         ablationElementLabels = []
420         ablationNodeLabels = []
421         ablationElementConnectivity = []
422
423         for element in allAblationElements:
424       ablationElementLabels.append(str(element.label))
425       ablationElementConnectivity.append(str(element.connectivity))
426
427         for node in allAblationNodes:
428             ablationNodeLabels.append(str(node.label))
429
430         ablationElementConnectivityStr = [0] * len(
              ablationElementConnectivity)
```

```
431         for i in range(len(ablationElementConnectivity)):
432       ablationElementConnectivityStr[i] = ablationElementConnectivity[i].
              replace('(', ', ').replace(')', '')
433
434         ablationElementDefinitions = [0] * len(ablationElementLabels)
435         for i in range(len(ablationElementLabels)):
436       ablationElementDefinitions[i] = ablationElementLabels[i] +
              ablationElementConnectivityStr[i]
437
438         # write to text file
439
440         for line in ablationElementDefinitions:
441       elementConnFile.write(line + '\n')
442
443         for line in ablationNodeLabels:
444             aleConstraintNodesFile.write(line + '\n')
445
446         elementConnFile.close()
447         aleConstraintNodesFile.close()
448
449   odb.close()
450
451   #
          ######################################################################
452
453   #SECTION III
454
455
456   from abaqus import*
457   from abaqusConstants import*
458   backwardCompatibility.setValues(includeDeprecated=True,
          reportDeprecated=False)
459
460   if firstRun == 1:
461         myModel2 = mdb.ModelFromOdbFile(name=inputFile+str('_static'),
              odbFileName=inputFile+str('_dynamic1.odb'))
462   else:
463         myModel2 = mdb.ModelFromOdbFile(name=inputFile+str('_static'),
              odbFileName=inputFile+str('_dynamic.odb'))
464
465   #static step(s)
466
467   import step
468   import regionToolset
469
470   myModel2.StaticStep(name='ablationStep', previous='Initial',
471     timePeriod=1.1, initialInc=0.1, minInc=1.0E-5, maxInc=0.1,
472     maxNumInc=10000, nlgeom=ON, description='apply ablation from dynamic
          analysis') # set timePeriod to 1.0+timeIncrement
473
474
475   #adaptive mesh controls
```

```
476
477   myAdaptiveMeshControl = myModel2.AdaptiveMeshControl(name='
          adaptiveMeshControls', originalConfigurationProjectionWeight = 0.5,
478           standardVolumetricSmoothingWeight = 0.5)
479
480   # adaptive mesh domains
481
482   ##get regions for adaptive mesh domain
483
484   aleMeshDomains = [0]*len(aleElementDomains)
485
486   for i in range(len(aleElementDomains)):
487           aleMeshDomains[i] = myModel2.rootAssembly.instances['PART-1-1'].
                sets[aleElementDomains[i]].elements
488
489   completeAleMeshDomains = aleMeshDomains[0]
490   for i in range(len(aleMeshDomains)-1):
491         completeAleMeshDomains = completeAleMeshDomains + aleMeshDomains[i
              +1]
492
493
494   completeAleDomainRegion = regionToolset.Region(elements=
          completeAleMeshDomains)
495   #Define adaptive mesh domains for steps
496
497
498   aleMeshDomain = myModel2.steps['ablationStep'].AdaptiveMeshDomain(
          region= completeAleDomainRegion, frequency=1, initialMeshSweeps=3,
499         meshSweeps=3, controls='adaptiveMeshControls')
500
501
502   #make some sets for history output
503
504   aleDomainSet = [0]*len(aleMeshDomains)
505
506   for i in range(len(aleMeshDomains)):
507           aleDomainSet[i] = myModel2.rootAssembly.Set(name=str(
                aleElementDomains[i])+'_ALEDOMAIN', elements =
                completeAleMeshDomains)
508
509   completeAleMeshDomainSet = myModel2.rootAssembly.Set(name=str('
          completeAleMeshDomainSet'), elements = completeAleMeshDomains)
510
511   #get history output for volume loss
512
513   volHistOutput = [0]*len(aleDomainSet)
514
515   for i in range(len(aleDomainSet)):
516           volHistOutput[i] = myModel2.HistoryOutputRequest(createStepName='
                ablationStep',
517                               frequency=1, name=str(aleElementDomains[i]+'
                                    _VOL/VOLC'), rebar=EXCLUDE,
```

```
518                             region=aleDomainSet[i], sectionPoints=DEFAULT,
                                    variables=('VOL', 'VOLC'))
519
520
521   #
          ##------------------------------------------------------------------------
522
523   #adpative mesh constraint
524
525   #get regions for adaptive mesh constraint domain
526
527   aleConstraintNodes = [0]*len(ablationSurfaces)
528
529   for i in range(len(ablationSurfaces)):
530         aleConstraintNodes[i] = myModel2.rootAssembly.surfaces[str(
                ablationSurfaces[i])].nodes
531
532   completeAleConstraintNodes = aleConstraintNodes[0]
533   for i in range(len(aleConstraintNodes)-1):
534        completeAleConstraintNodes = completeAleConstraintNodes +
              aleConstraintNodes[i+1]
535
536   completeAleConstraintSet = myModel2.rootAssembly.Set(name='
          completeAleConstraintSet', nodes = completeAleConstraintNodes)
537
538   #define ale constraint regions
539
540   aleConstraintRegions = [0]*len(aleConstraintNodes)
541   for i in range(len(aleConstraintNodes)):
542         aleConstraintRegions[i] = regionToolset.Region(nodes=
              aleConstraintNodes[i])
543
544   #define adaptive mesh constraints
545
546         aleConstraints = [0]*len(aleConstraintRegions)
547         for i in range(len(aleConstraintRegions)):
548               aleConstraints[i] = myModel2.
                     DisplacementAdaptiveMeshConstraint(name=ablationSurfaces
                     [i], createStepName = 'ablationStep',
549         region=aleConstraintRegions[i], motionType = USER_DEFINED)
550
551
552   instanceKeys = myModel2.rootAssembly.instances.keys()
553
554   allElements = [0]*len(instanceKeys)
555   for i in range(len(instanceKeys)):
556        allElements[i] = myModel2.rootAssembly.instances[instanceKeys[i]].
              elements
557
558
559         partKeys = myModel2.parts.keys()
560         c1 = 0
```

```
561         wrongElement = [0]*len(instanceKeys)
562         for i in range(len(instanceKeys)):
563            for element in allElements[i]:
564               c1 = c1 + 1
565               if len(element.connectivity) == 1:
566                     wrongElement[i] = myModel2.parts[partKeys[i]].
                         elements[c1-1]
567                     myModel2.parts[partKeys[i]].deleteElement(elements=
                         wrongElement[i])
568
569
570   #
         ----------------------------------------------------------------------------
571
572
573   ##suppress constraints
574   #constraints might be a problem in adaptive meshing
575
576         constraintNames = myModel2.constraints.keys()
577
578         for constr in constraintNames:
579               myModel2.constraints[constr].suppress()
580
581   #static analysis
582
583   import job
584
585   staticJob = mdb.Job(name=inputFile+str('_static'), model=myModel2,
         userSubroutine='umeshmotion_simplified2.f')
586         myModel2.setValues(noPartsInputFile=ON)
587   staticJob.writeInput()
588   time.sleep(5)
589
590   #
         ----------------------------------------------------------------------------
591
592   #"""
593   #rewrite node coordinates in input file
594
595   #"""
596
597   from abaqus import*
598   from abaqusConstants import*
599   backwardCompatibility.setValues(includeDeprecated=True,
         reportDeprecated=False)
600   from odbAccess import *
601   from odbMaterial import*
602   from odbSection import*
603
604
605   myModel3 = mdb.Model(name=inputFile+'Temp')
```

```
606
607   odbPath = inputFile+str('_dynamic')+'.odb'
608
609   #Open output database
610
611   odbDynamic = openOdb(path=odbPath)
612   tempPart = myModel3.PartFromOdb(name=inputFile+'TempPart', odb=
          odbDynamic, shape=DEFORMED, step=-1)
613
614
615   #get node coordinates of all nodes in odb
616
617   allNodes = tempPart.nodes
618   numberAllNodes = len(allNodes)
619   diffNodes = numberAllInitialNodes - numberAllNodes
620
621   # Get all nodes after ablation
622
623
624   lengthOdbNodes = len(allNodes)
625
626   nodeLabels = []
627   nodeCoordinates = []
628   for node in allNodes:
629     nodeLabels.append(node.label)
630     nodeCoordinates.append(node.coordinates)
631
632
633   # create list of strings
634
635
636   nodeDefinitionString = []
637   for i in range(len(nodeLabels)):
638     nodeDefinitionString.append(' ' + ' ' + ' ' + str(nodeLabels[i]) +','
           +  ' ' + str(nodeCoordinates[i][0]) +
639     ',' + ' ' + str(nodeCoordinates[i][1]) +',' + ' ' + str(
          nodeCoordinates[i][2]))
640
641
642
643   modifiedInputFile = inputFile+str('_staticInput.inp')
644   initialInputFile = inputFile+str('_static.inp')
645   readFile = open(initialInputFile, 'r')
646   writeFile = open(modifiedInputFile, 'w')
647
648
649   lineCount = 0
650   nodeDefinitionStarted = 0
651   nodeDefinitionEnded = 0
652   for i, line in enumerate(readFile):
653     lineCount = lineCount + 1
654     if str('*Node') in line and nodeDefinitionStarted != 1:
655       nodeDefinitionStarted = 1
```

```
656         startLine = lineCount
657       elif str('*') in line and nodeDefinitionStarted == 1 and
              nodeDefinitionEnded !=1:
658         endLine = lineCount
659         nodeDefinitionEnded = 1
660
661
662   readFile.seek(0,0)
663   lineCount2 = 0
664
665   for i, line in enumerate(readFile):
666     lineCount2 = lineCount2 + 1
667     if lineCount2 > (startLine) and lineCount2 < endLine:
668       writeFile.write(str(nodeDefinitionString[lineCount2-(startLine+1)])
              + '\n')
669     else:
670       writeFile.write(str(line))
671
672
673   writeFile.close()
674   readFile.close()
675   odbDynamic.close()
676
677   modifiedInputJob = mdb.JobFromInputFile(name=inputFile+str('
          _staticInput'), inputFileName=inputFile+str('_staticInput.inp'),
          userSubroutine='umeshmotion_simplified2.f')
678   modifiedInputJob.submit()
679         modifiedInputJob.waitForCompletion()
680
681   #
          #-------------------------------------------------------------------------------
682
683
684   #"""
685   #rewrite node coordinates in input file
686
687   #"""
688
689   from abaqus import*
690   from abaqusConstants import*
691   backwardCompatibility.setValues(includeDeprecated=True,
          reportDeprecated=False)
692   from odbAccess import *
693   from odbMaterial import*
694   from odbSection import*
695
696   myModel3 = mdb.Model(name=inputFile+'Temp')
697
698   odbPath = inputFile+str('_staticInput')+'.odb'
699
700   #Open output database
701
```

```
702   odbStatic = openOdb(path=odbPath)
703   tempPart = myModel3.PartFromOdb(name=inputFile+'TempPart', odb=
          odbStatic, shape=DEFORMED, step=-1)
704
705   allNodesStatic = odbStatic.rootAssembly.instances['PART-1-1'].nodes
706
707   diffNodes = len(allNodesDynamic) - len(allNodesStatic)
708   if diffNodes != 0 and inputData[9] == '':
709           warning = getWarningReply(message='Node missing. Check if there
                are coupling nodes and enter coupling node label', buttons=(
                YES,NO,CANCEL))
710           break
711
712   #get node coordinates of all nodes in odb
713
714   allNodes = tempPart.nodes
715   numberAllNodes = len(allNodes)
716
717   # Get all nodes after ablation
718
719
720   lengthOdbNodes = len(allNodes)
721
722   nodeLabels = []
723   nodeCoordinates = []
724   for node in allNodes:
725     nodeLabels.append(node.label)
726     nodeCoordinates.append(node.coordinates)
727
728
729   # create list of strings
730
731
732   nodeDefinitionString = []
733   for i in range(len(nodeLabels)):
734     nodeDefinitionString.append(' ' + ' ' + ' ' + str(nodeLabels[i]) +','
          +  ' ' + str(nodeCoordinates[i][0]) +
735     ',' + ' ' + str(nodeCoordinates[i][1]) +',' + ' ' + str(
          nodeCoordinates[i][2]))
736
737   if inputData[9] != '':
738               nodeDefinitionString.append(missingNodeDefinition[0])
739
740   modifiedInputFile = inputFile+str('_mod.inp')
741   initialInputFile = inputFile+str('.inp')
742   readFile = open(initialInputFile, 'r')
743   writeFile = open(modifiedInputFile, 'w')
744
745
746   # find node definition lines
747
748   countParts = 0
749   for line in readFile:
```

```
750             if str('** PART INSTANCE') in line:
751                 countParts = countParts + 1
752
753
754         readFile.seek(0,0)
755
756     lineCount = 0
757     startLine = [0]*countParts
758     endLine = [0]*countParts
759     nodeDefinitionStarted = [0]*countParts
760     nodeDefinitionEnded = [0]*countParts
761
762     j = 0
763         for i, line in enumerate(readFile):
764         lineCount = lineCount + 1
765         if nodeDefinitionEnded[-1] != 1:
766             if str('*Node') in line and nodeDefinitionStarted[j] != 1:
767              nodeDefinitionStarted[j] = 1
768              startLine[j] = lineCount
769              j = j + 1
770             elif str('*') in line and nodeDefinitionStarted[j-1] == 1 and
                    nodeDefinitionEnded[j-1] != 1:
771              nodeDefinitionEnded[j-1] = 1
772              endLine[j-1] = lineCount
773
774
775     readFile.seek(0,0)
776
777         j = 0
778         lineCount2 = 0
779         itemCount = 0
780         allPartsDone = 0
781         for i, line in enumerate(readFile):
782           lineCount2 = lineCount2 + 1
783           if j != countParts and (lineCount2) > (startLine[j]) and (
                lineCount2) < (endLine[j]) and (itemCount + 1):
784         itemCount = itemCount + 1
785         writeFile.write(str(nodeDefinitionString[itemCount-1]) + '\n')
786         if lineCount2 == endLine[j] - 1:
787               j = j + 1
788           else:
789         writeFile.write(str(line))
790
791
792         # volume loss, sliding distance, extrapolation factor
793     odbStatic = openOdb(path = inputFile+str('_staticInput.odb'))
794     historyRegionsKeys = odbStatic.steps['ablationStep'].historyRegions.
        keys()
795
796     VOLCCurrent = odbStatic.steps['ablationStep'].historyRegions[
        historyRegionsKeys[1]].historyOutputs['VOLC'].data[-1][-1]
797     VOLC = VOLC - VOLCCurrent
798
```

```
799
800     VOLCFile = open('VOLC.txt','a')
801     slidingDistanceFile = open('slidingDistance.txt','a')
802     extrapolationFactorsFile = open('extrapolationFactors.txt','a')
803
804
805     VOLCFile.write(str(VOLC) + '\n')
806     slidingDistanceFile.write(str(relativeSliding) + '\n')
807     extrapolationFactorsFile.write(str(maxScalingFactor) + '\n')
808
809           VOLCFile.close()
810     slidingDistanceFile.close()
811     extrapolationFactorsFile.close()
812
813
814
815     writeFile.close()
816     readFile.close()
817     odbStatic.close()
818
819     firstRun = 2
820
821           #
                ------------------------------------------------------------------------
822
823
824 # post-processing/visualization
825 if relativeSliding >= totalSliding:
826
827     """
828     add wear depth
829
830     """
831     numberOfAblationFiles = numCycle
832     wearFile = [0]*numberOfAblationFiles
833     for i in range(0,numberOfAblationFiles):
834           wearFile[i] = open('ablation'+str(i+1)+'.txt','r')
835
836
837
838     data = [0]*numberOfAblationFiles
839     for i in range(0,numberOfAblationFiles):
840         data[i] = []
841         for line in wearFile[i]:
842       data[i].append(line.replace(',','').replace('\n','').split(' '))
843
844
845     for i in range(0,numberOfAblationFiles):
846         wearFile[i].close()
847
848
849     nodeLabels = []
```

```
850   for item in data[0]:
851       nodeLabels.append(int(item[0]))
852
853
854   wearDepth = [0]*numberOfAblationFiles
855   for i in range(0,numberOfAblationFiles):
856       wearDepth[i] = []
857
858   for i in range(0,numberOfAblationFiles):
859       for item in data[i]:
860     wearDepth[i].append(float(item[3]))
861
862
863   wearDepthTotal = [0]*len(wearDepth[0])
864   for j in range(len(wearDepth[0])):
865       for i in range(0,numberOfAblationFiles):
866     wearDepthTotal[j] = wearDepthTotal[j] + wearDepth[i][j]
867
868   wearTotalFile = open('wearDepthTotal.txt', 'w')
869
870
871   for i in range(len(wearDepthTotal)):
872         wearTotalFile.write(str(nodeLabels[i]) + '\t' + str(
              wearDepthTotal[i]) + '\n')
873
874
875   wearTotalFile.close()
876
877         #
              ----------------------------------------------------------------------------
878
879         #static step(s)
880
881   myModel4 = mdb.ModelFromOdbFile(name=inputFile+str('_visualizationModel
          '), odbFileName=inputFile+str('_dynamic1.odb'))
882
883   import step
884   import regionToolset
885
886   myModel4.StaticStep(name='ablationStep', previous='Initial',
887     timePeriod=1.1, initialInc=0.1, minInc=1.0E-5, maxInc=0.1,
888     maxNumInc=10000, nlgeom=ON, description='apply ablation from dynamic
          analysis') # set timePeriod to 1.0+timeIncrement
889
890
891   #adaptive mesh controls
892
893   myAdaptiveMeshControl = myModel4.AdaptiveMeshControl(name='
          adaptiveMeshControls', originalConfigurationProjectionWeight = 0.5,
894         standardVolumetricSmoothingWeight = 0.5)
895
896   # adaptive mesh domains
```

```
897
898   ##get regions for adaptive mesh domain
899
900   aleMeshDomains = [0]*len(aleElementDomains)
901
902   for i in range(len(aleElementDomains)):
903         aleMeshDomains[i] = myModel4.rootAssembly.instances['PART-1-1'].
              sets[aleElementDomains[i]].elements
904
905   completeAleMeshDomains = aleMeshDomains[0]
906   for i in range(len(aleMeshDomains)-1):
907        completeAleMeshDomains = completeAleMeshDomains + aleMeshDomains[i
             +1]
908
909
910   completeAleDomainRegion = regionToolset.Region(elements=
         completeAleMeshDomains)
911   #Define adaptive mesh domains for steps
912
913
914   aleMeshDomain = myModel4.steps['ablationStep'].AdaptiveMeshDomain(
         region= completeAleDomainRegion, frequency=1, initialMeshSweeps=3,
915       meshSweeps=3, controls='adaptiveMeshControls')
916
917
918   #make some sets for history output
919
920   aleDomainSet = [0]*len(aleMeshDomains)
921
922   for i in range(len(aleMeshDomains)):
923         aleDomainSet[i] = myModel4.rootAssembly.Set(name=str(
              aleElementDomains[i])+'_ALEDOMAIN', elements =
              completeAleMeshDomains)
924
925   completeAleMeshDomainSet = myModel4.rootAssembly.Set(name=str('
         completeAleMeshDomainSet'), elements = completeAleMeshDomains)
926
927   #get history output for volume loss
928
929   volHistOutput = [0]*len(aleDomainSet)
930
931   for i in range(len(aleDomainSet)):
932         volHistOutput[i] = myModel4.HistoryOutputRequest(createStepName='
              ablationStep',
933                               frequency=1, name=str(aleElementDomains[i]+'
                                  _VOL/VOLC'), rebar=EXCLUDE,
934                               region=aleDomainSet[i], sectionPoints=DEFAULT,
                                  variables=('VOL', 'VOLC'))
935
936
937   #
         ##------------------------------------------------------------------------
```

```
938
939   #adpative mesh constraint
940
941   #get regions for adaptive mesh constraint domain
942
943   aleConstraintNodes = [0]*len(ablationSurfaces)
944
945   for i in range(len(ablationSurfaces)):
946         aleConstraintNodes[i] = myModel4.rootAssembly.surfaces[str(
                ablationSurfaces[i])].nodes
947
948   completeAleConstraintNodes = aleConstraintNodes[0]
949   for i in range(len(aleConstraintNodes)-1):
950        completeAleConstraintNodes = completeAleConstraintNodes +
              aleConstraintNodes[i+1]
951
952   completeAleConstraintSet = myModel4.rootAssembly.Set(name='
         completeAleConstraintSet', nodes = completeAleConstraintNodes)
953
954   #define ale constraint regions
955
956   aleConstraintRegions = [0]*len(aleConstraintNodes)
957   for i in range(len(aleConstraintNodes)):
958         aleConstraintRegions[i] = regionToolset.Region(nodes=
                aleConstraintNodes[i])
959
960   #define adaptive mesh constraints
961
962         aleConstraints = [0]*len(aleConstraintRegions)
963         for i in range(len(aleConstraintRegions)):
964               aleConstraints[i] = myModel4.
                     DisplacementAdaptiveMeshConstraint(name=ablationSurfaces
                     [i], createStepName = 'ablationStep',
965         region=aleConstraintRegions[i], motionType = USER_DEFINED)
966
967
968   instanceKeys = myModel4.rootAssembly.instances.keys()
969
970   allElements = [0]*len(instanceKeys)
971   for i in range(len(instanceKeys)):
972        allElements[i] = myModel4.rootAssembly.instances[instanceKeys[i]].
              elements
973
974
975         partKeys = myModel4.parts.keys()
976         c1 = 0
977         wrongElement = [0]*len(instanceKeys)
978         for i in range(len(instanceKeys)):
979            for element in allElements[i]:
980               c1 = c1 + 1
981               if len(element.connectivity) == 1:
982                     wrongElement[i] = myModel4.parts[partKeys[i]].
                           elements[c1-1]
```

```
983                     myModel4.parts[partKeys[i]].deleteElement(elements=
                            wrongElement[i])
984
985
986     #
            ------------------------------------------------------------------------------
987
988
989     ##suppress constraints
990     #constraints might be a problem in adaptive meshing
991
992           constraintNames = myModel4.constraints.keys()
993
994           for constr in constraintNames:
995                 myModel4.constraints[constr].suppress()
996
997     #static analysis
998
999     import job
1000
1001    visualizationJob = mdb.Job(name=inputFile+str('_visualization'), model=
            myModel4, userSubroutine='umeshmotion_simplified2_visualization.f')
1002          myModel4.setValues(noPartsInputFile=ON)
1003          visualizationJob.writeInput()
1004          visualizationJob.submit()
1005          visualizationJob.waitForCompletion()
1006
1007          #
                 ------------------------------------------------------------------------------
1008
1009          # create ablation field output
1010
1011    from odbAccess import*
1012
1013    staticOdbName = inputFile+'_visualization.odb'
1014    staticOdb = openOdb(path=staticOdbName)
1015    staticOdbInstance = staticOdb.rootAssembly.instances['PART-1-1']
1016
1017    ablationFrames = [0]*len(staticOdb.steps['ablationStep'].frames)
1018
1019    for i in range(len(ablationFrames)):
1020          ablationFrames[i] = staticOdb.steps['ablationStep'].frames[i]
1021
1022
1023    ablationFile = open('wearDepthTotal.txt', 'r')
1024
1025    ablation = []
1026    for i, line in enumerate(ablationFile):
1027          ablation.append(line.split())
1028
1029
```

```
1030    ablationFile.close()
1031
1032    nodeLabels = [int(ablation[i][0]) for i in range(len(ablation))]
1033    m = [0]*len(ablationFrames)
1034    for i in range(len(m)):
1035        m[i] = i*(1.0/len(ablationFrames))
1036
1037    ablationData = [0]*len(ablationFrames)
1038    for j in range(len(ablationFrames)):
1039          ablationData[j] = [[m[j]*float(ablation[i][1])] for i in range(
                len(ablation))]
1040
1041    # create field output object
1042
1043    ablationFieldOutput = [0] * len(ablationFrames)
1044    for i in range(len(ablationFieldOutput)):
1045          ablationFieldOutput[i] = ablationFrames[i].FieldOutput(name='Wear
                Depth total @' + str(totalSliding), description='Wear depths
                read from text file', type=SCALAR)
1046
1047
1048    # add relevant data to field output object
1049
1050    for i in range(len(ablationFieldOutput)):
1051          ablationFieldOutput[i].addData(position=NODAL, instance=
                staticOdbInstance, labels=nodeLabels, data=ablationData[i])
1052
1053
1054    staticOdb.save()
1055    staticOdb.close()
```

# Appendix B

# UMESHMOTION Subroutine 1

## B.1 UMESHMOTION Subroutine 1: Node shift direction normal to contact surface

```
1         INCLUDE 'umeshmotion_module.f'
2         INCLUDE 'umeshmotion_functions.f'
3   C
4         SUBROUTINE UMESHMOTION(UREF,ULOCAL,NODE,NNDOF,LNODETYPE,ALOCAL,
5        1 NDIM,TIME,DTIME,PNEWDT,KSTEP,KINC,KMESHSWEEP,JMATYP,JGVBLOCK,
6        2 LSMOOTH)
7   C
8         USE MODULE_UMESHMOTION ! LOAD MODULE
9         IMPLICIT NONE
10  C
11  C     DECLARATION OF VARIABLES
12  C
13        INTEGER,PARAMETER::NPRECD=2 ! PARAMETER FOR DOUBLE PRECISSION
14  C
15        INTEGER NODE,NNDOF,LNODETYPE,NDIM,KSTEP,KINC,KMESHSWEEP,LSMOOTH,
16       1 FINDINDEX,FIRSTRUN,KINC_MEM, ZERO, K1, K2
17  C
18        REAL*8 UREF,DTIME,PNEWDT
19  C
20        REAL*8 ULOCAL(NDIM),ALOCAL(NDIM,*),TIME(2),JMATYP(*),JGVBLOCK(*),
21       1 ARRAY(15)
22  C
23  C     INITIAL VALUES FOR FIRST SUBROUTINE RUN
24        DATA FIRSTRUN /-1/
25        DATA KINC_MEM /0/
26  C
27  C     ****************************************************************
28  C     * MAIN SECTION                                                 *
29  C     ****************************************************************
```

```
30  C
31  C      ONLY FOR FIRST SUBROUTINE RUN
32         IF (FIRSTRUN.EQ.-1) THEN
33            WRITE(6,*) 'FIRSTRUN'
34            WRITE(6,*) 'KINC:', KINC
35            ZERO = 0.0
36            FIRSTRUN=1
37  C         PARAMETER FOR FIRST STEP
38            FIRSTSTEP=KSTEP
39  C         GET WORKING DIRECTORY
40            CALL GETOUTDIR(OUTDIR,LENOUTDIR)
41  C         GET JOBNAME
42            CALL GETJOBNAME(JOBNAME,LENJOBNAME)
43  C         OUTOUT TO FILE
44            WRITE(6,*) ''
45            WRITE(6,*) '****************************************'
46            WRITE(6,*) 'STARTED UMESHMOTION'
47            WRITE(6,*) 'FOR JOB: ',JOBNAME
48            WRITE(6,*) 'IN WORKING DIRECTORY: ',OUTDIR
49            WRITE(6,*) '****************************************'
50            WRITE(6,*) ''
51            FIRSTNODE=NODE
52            NODE_I=0
53  C            DETERMINE TOTAL NUMBER OF LINES
54               NP=ZERO
55               STAT_R=ZERO
56               OPEN(400,FILE=OUTDIR(1:LENOUTDIR)//'/'//'aleConstraintNodes.
                    txt',
57         1           STATUS='OLD',ACTION='READ',IOSTAT=STAT_O)
58                DO K1=1,MAXRECS
59                 READ(400,*,IOSTAT=STAT_R) CONSTRAINTNODESCOUNTER
60                    IF(STAT_R.NE.0) EXIT
61                    IF (K1.EQ.MAXRECS) THEN
62                       write(7,*) 'Error: Maximum number of records exceeded
                            ...'
63                       write(7,*) 'Exiting now...'
64                       CALL XIT
65                    ENDIF
66                     NP = NP + 1
67                END DO
68                REWIND(400)
69                 ALLOCATE(ALLCONSTRAINTNODES(NP))
70                 READ(400,*,IOSTAT=STAT_R) ALLCONSTRAINTNODES
71                  WRITE(6,*) 'ALLCONSTRAINTNODES: ', ALLCONSTRAINTNODES
72                 CLOSE(400)
73               NR=ZERO
74               STAT_R=ZERO
75               OPEN(300,FILE=OUTDIR(1:LENOUTDIR)//'/'//'elementConnectivity.
                    txt',
76         1           STATUS='OLD',ACTION='READ',IOSTAT=STAT_O)
77                DO K1=1,MAXRECS
78                 READ(300,*,IOSTAT=STAT_R) FIRST
79                    IF(STAT_R.NE.0) EXIT
```

```
80                    IF (K1.EQ.MAXRECS) THEN
81                       write(7,*) 'Error: Maximum number of records exceeded
                             ...'
82                       write(7,*) 'Exiting now...'
83                       CALL XIT
84                    ENDIF
85                     NR = NR + 1
86                 END DO
87                 REWIND(300)
88                  ALLOCATE(ALLELEM(9,NR))
89                  ALLOCATE(ALLELEM_TRANS(NR,9))
90                  READ(300,*,IOSTAT=STAT_R) ALLELEM
91                  REWIND(300)
92                  ALLELEM_TRANS=TRANSPOSE(ALLELEM)
93                  WRITE(6,*) 'ALLELEM_TRANS: ', ALLELEM_TRANS
94                  CLOSE(300)
95  C            READ ABLATION DATA
96                  STAT_R=ZERO
97                  OPEN(500,FILE=OUTDIR(1:LENOUTDIR)//'/'//'ablation.txt',
98                  STATUS='OLD',ACTION='READ',IOSTAT=STAT_O)
99                  DO K1=1,MAXRECS
100                   READ(500,*,IOSTAT=STAT_R) ABL
101                      IF(STAT_R.NE.0) EXIT
102                      IF (K1.EQ.MAXRECS) THEN
103                         write(7,*) 'Error: Maximum number of records exceeded
                               ...'
104                         write(7,*) 'Exiting now...'
105                         CALL XIT
106                      ENDIF
107                       NQ = NQ + 1
108                END DO
109                 REWIND(500)
110                 ALLOCATE(TOTAL_ABLATION(2,NQ))
111                 ALLOCATE(TOTAL_ABLATION_TRANS(NQ,2))
112                 READ(500,*,IOSTAT=STAT_R) TOTAL_ABLATION
113                 WRITE(7,*) 'READ TOTAL_ABLATION'
114                 TOTAL_ABLATION_TRANS = TRANSPOSE(TOTAL_ABLATION)
115                  DO K3=1,NQ
116                    WRITE(7,*) 'TOTAL_ABLATION: ', TOTAL_ABLATION_TRANS(K3
                           ,:)
117                  END DO
118                REWIND(500)
119                CLOSE(500)
120          ENDIF
121 C
122 C
123          IF (KMESHSWEEP.EQ.0) THEN
124 C          GET WEAR DEPTH
125            DO K1 = 1, NQ
126               IF(NODE.EQ.TOTAL_ABLATION_TRANS(K1,1)) THEN
127                  ABL_APPLIED = DTIME*TOTAL_ABLATION_TRANS(K1,2)
128               END IF
129            END DO
```

```
130           IF (NDIM.EQ.2) THEN
131              ULOCAL(2)=ULOCAL(2)-ABL_APPLIED
132           ELSEIF (NDIM.EQ.3) THEN
133   C       GET ELEMENTS CONNECTED TO NODE
134           NELEMS_CORRECT = 0
135                 DO K1=1,NR
136                   DO K2= 2, 9
137                      IF (NODE.EQ.ALLELEM_TRANS(K1,K2)) THEN
138                          NELEMS_CORRECT = NELEMS_CORRECT + 1
139                      ENDIF
140                   END DO
141                 END DO
142   C           ALLOCATE SIZE FOR NODES ARRAY
143                ALLOCATE(NODESELEM(NELEMS_CORRECT,8))
144                IF (NELEMS_CORRECT.EQ.2) THEN
145                   NELEMS_CORRECT2 = 0
146                   DO K1=1,NR
147                      DO K2= 2,9
148                         IF (NODE.EQ.ALLELEM_TRANS(K1,K2)) THEN
149                            NELEMS_CORRECT2 = NELEMS_CORRECT2 + 1
150                              DO K3 = 1, 8
151                                NODESELEM(NELEMS_CORRECT2, K3) = ALLELEM_TRANS
                                      (K1,K3+1)
152                              END DO
153                         ENDIF
154                      END DO
155                   END DO
156   C             FIND NODES COMMON TO BOTH ELEMENTS
157                    C2=ZERO
158                     DO K2=1,8
159                        DO K4=1,8
160                           IF(NODESELEM(1,K2).EQ.NODESELEM(2,K4))THEN
161                           C2=C2+1
162                           COMMON_NODES(C2)=NODESELEM(1,K2)
163                           END IF
164                        END DO
165                     END DO
166   C              FIND PERPENDICULAR NODE
167                    C3=ZERO
168                      DO K3=1,C2
169                       DO K4=1,NP
170                        IF(COMMON_NODES(K3).EQ.ALLCONSTRAINTNODES(K4).
171                            AND.COMMON_NODES(K3).NE.NODE) THEN
172                         C3=C3+1
173                         PERP_NODE=COMMON_NODES(K3)
174                        END IF
175                       END DO
176                      END DO
177   C              FIND SUBSURFACE NODES
178                       C8= ZERO
179                         DO K1=1,C2
180                           C7=ZERO
181                           DO K2=1,NP
```

```
182                         IF(COMMON_NODES(K1).NE.ALLCONSTRAINTNODES(K2))THEN
183                           C7=C7+1
184                            IF(C7.EQ.NP) THEN
185                              C8=C8+1
186                              SUB_NODES(C8)=COMMON_NODES(K1)
187                            END IF
188                         END IF
189                       END DO
190                     END DO
191 C             FIND NODES THAT ARE NOT SHARED BY JELEMLIST
192                   C5=ZERO
193                   DO K1=1,NELEMS_CORRECT
194                    DO K2=1,8
195                     C4=ZERO
196                     DO K3=1,C2
197                      IF(NODESELEM(K1,K2).NE.COMMON_NODES(K3)) THEN
198                       C4=C4+1
199                        IF(C4.EQ.4) THEN
200                         C5=C5+1
201                         NOTCOMMON_NODES(C5)=NODESELEM(K1,K2)
202                        END IF
203                      END IF
204                     END DO
205                    END DO
206                   END DO
207 C             FIND NOTCOMMON_NODES THAT BELONG TO ADAPTIVE MESH CONSTRAINT
      DOMAIN
208                    C6=ZERO
209                    DO K1=1,8
210                     DO K2=1,NP
211                       IF(NOTCOMMON_NODES(K1).EQ.ALLCONSTRAINTNODES(K2))
                             THEN
212                        C6=C6+1
213                        SURF_NODES(C6)=NOTCOMMON_NODES(K1)
214                       END IF
215                     END DO
216                    END DO
217 C             FIND VECTORS FROM NODE TO SURF_NODES
218 C               1. FIND COORDINATES OF SURFACE NODES
219                       LTRN=0
220                       DO K1=1,4
221                       CALL GETVRN(SURF_NODES(K1),'COORD',ARRAY,JRCD,
                            JGVBLOCK,LTRN)
222                         DO K2=1,3
223                           SURFNODE_COORDS(K1,K2) = ARRAY(K2)
224                         END DO
225                       END DO
226 C               2. FIND COORDINATES OF CURRENT NODE
227                       CALL GETVRN(NODE,'COORD',ARRAY,JRCD,JGVBLOCK,LTRN)
228                         DO K3=1,3
229                           NODE_COORDS(K3)=ARRAY(K3)
230                         END DO
231 C               3. FIND ALL SURFACE VECTORS
```

```
232                       DO K4=1,4
233                        DO K5=1,3
234                           V_SURFNODES(K4,K5)= SURFNODE_COORDS(K4,K5)-
                                 NODE_COORDS(K5)
235                        END DO
236                           MGNT_VSURFNODES(K4)=SQRT(V_SURFNODES(K4,1)**2+
237                           V_SURFNODES(K4,2)**2+V_SURFNODES(K4,3)**2)
238                       END DO
239 C                  3.1 FIND SHORTER VECTORS
240                        DO K1=1,3
241                         DO K2=K1+1,4
242                          IF(MGNT_VSURFNODES(K1).GT.MGNT_VSURFNODES(K2))
                                 THEN
243                           DO K3=1,3
244                            TEMP_V(K3)=V_SURFNODES(K1,K3)
245                            V_SURFNODES(K1,K3)=V_SURFNODES(K2,K3)
246                            V_SURFNODES(K2,K3)=TEMP_V(K3)
247                           END DO
248                            TEMP_M=MGNT_VSURFNODES(K1)
249                            MGNT_VSURFNODES(K1)=MGNT_VSURFNODES(K2)
250                            MGNT_VSURFNODES(K2)=TEMP_M
251                          END IF
252                         END DO
253                        END DO
254 C                 4. FIND UNIT VECTORS
255                       DO K1=1,4
256                         DO K2=1,3
257                          V_SURFNODES_UNIT(K1,K2)=V_SURFNODES(K1,K2)/
                                MGNT_VSURFNODES(K1)
258                         END DO
259                       END DO
260 C
261 C               FIND VECTOR FROM NODE TO PERP_NODE
262                    CALL GETVRN(PERP_NODE,'COORD',ARRAY,JRCD,JGVBLOCK,LTRN)
263                       DO K3=1,3
264                        V_PERPNODE(K3)= ARRAY(K3)-NODE_COORDS(K3)
265                       END DO
266                        MGNT_VPERPNODE=SQRT(V_PERPNODE(1)**2+V_PERPNODE(2)
                              **2+V_PERPNODE(3)**2)
267                       DO K4=1,3
268                        V_PERPNODE_UNIT(K4)=V_PERPNODE(K4)/MGNT_VPERPNODE
269                       END DO
270 C               FIND VECTORS FROM NODE TO SUB_NODE (TAKE ARBITRARY SUB_NODE)
271                    CALL GETVRN(SUB_NODES(1),'COORD',ARRAY,JRCD,JGVBLOCK,LTRN
                          )
272                       DO K3=1,3
273                        V_SUBNODE(K3)= ARRAY(K3)-NODE_COORDS(K3)
274                       END DO
275                        MGNT_VSUBNODE=SQRT(V_SUBNODE(1)**2+V_SUBNODE(2)**2+
                              V_SUBNODE(3)**2)
276                       DO K4=1,3
277                        V_SUBNODE_UNIT(K4)=V_SUBNODE(K4)/MGNT_VSUBNODE
278                       END DO
```

```
279 C             FIND NORMAL VECTORS WITH CROSS PRODUCT
280                   CROSS1_X=V_PERPNODE_UNIT(2)*V_SURFNODES_UNIT(1,3)-
281                            V_PERPNODE_UNIT(3)*V_SURFNODES_UNIT(1,2)
282                   CROSS1_Y=V_PERPNODE_UNIT(3)*V_SURFNODES_UNIT(1,1)-
283                            V_PERPNODE_UNIT(1)*V_SURFNODES_UNIT(1,3)
284                   CROSS1_Z=V_PERPNODE_UNIT(1)*V_SURFNODES_UNIT(1,2)-
285                            V_PERPNODE_UNIT(2)*V_SURFNODES_UNIT(1,1)
286                   NORM_EDGE_V(1,1)=CROSS1_X
287                   NORM_EDGE_V(1,2)=CROSS1_Y
288                   NORM_EDGE_V(1,3)=CROSS1_Z
289                   MGNT_NORMEDGEV(1)=SQRT(NORM_EDGE_V(1,1)**2+NORM_EDGE_V
                          (1,2)**2+NORM_EDGE_V(1,3)**2)
290 C
291                   CROSS2_X=V_PERPNODE_UNIT(2)*V_SURFNODES_UNIT(2,3)-
292                            V_PERPNODE_UNIT(3)*V_SURFNODES_UNIT(2,2)
293                   CROSS2_Y=V_PERPNODE_UNIT(3)*V_SURFNODES_UNIT(2,1)-
294                            V_PERPNODE_UNIT(1)*V_SURFNODES_UNIT(2,3)
295                   CROSS2_Z=V_PERPNODE_UNIT(1)*V_SURFNODES_UNIT(2,2)-
296                            V_PERPNODE_UNIT(2)*V_SURFNODES_UNIT(2,1)
297                   NORM_EDGE_V(2,1)=CROSS2_X
298                   NORM_EDGE_V(2,2)=CROSS2_Y
299                   NORM_EDGE_V(2,3)=CROSS2_Z
300                   MGNT_NORMEDGEV(2)=SQRT(NORM_EDGE_V(2,1)**2+
301                                          NORM_EDGE_V(2,2)**2+NORM_EDGE_V(2,3)
                                              **2)
302 C              FIND ANGLE BETWEEN NORM_EDGE AND V_SUBNODE_UNIT
303                    DO K1=1,2
304                       THETA(K1)=ACOS((NORM_EDGE_V(K1,1)*V_SUBNODE_UNIT(1)+
305                                       NORM_EDGE_V(K1,2)*V_SUBNODE_UNIT(2)+
306       1                               NORM_EDGE_V(K1,3)*V_SUBNODE_UNIT(3))/(
              MGNT_NORMEDGEV(K1)))
307                       IF(THETA(K1).GT.PI/2) THEN
308                         NORM_EDGE_V(K1,:)=-NORM_EDGE_V(K1,:)
309                       END IF
310                    END DO
311 C              VERIFY ANGLE
312                    DO K1=1,2
313                      THETA_VERIF(K1)=ACOS((NORM_EDGE_V(K1,1)*
                            V_SUBNODE_UNIT(1)+
314                                          NORM_EDGE_V(K1,2)*V_SUBNODE_UNIT(2)+
315       1                                  NORM_EDGE_V(K1,3)*V_SUBNODE_UNIT(3))
              /(MGNT_NORMEDGEV(K1)))
316                    END DO
317 C              FIND AVERAGE NORMAL
318                    DO K1=1,3
319                      SUM_NORMEDGE(K1)= NORM_EDGE_V(1,K1)+NORM_EDGE_V(2,K1)
320                    END DO
321                    MGNT_SUMNORMEDGE=SQRT(SUM_NORMEDGE(1)**2+SUM_NORMEDGE
                          (2)**2+SUM_NORMEDGE(3)**2)
322                    DO K2=1,3
323                      NORM_EDGE_AVG(K2)=(SUM_NORMEDGE(K2))/(
                            MGNT_SUMNORMEDGE)
324                    END DO
```

```
325                       MGNT_NORMEDGEV_AVG=SQRT(NORM_EDGE_AVG(1)**2+
326                                               NORM_EDGE_AVG(2)**2+NORM_EDGE_AVG
                                                    (3)**2)
327   C               CALULATE WEAR IN DIRECTION OF NORM_EDGE_AVG
328                       DO K1=1,3
329                          WEAR_GLOBAL(K1) = ABL_APPLIED*(NORM_EDGE_AVG(K1)/
                               MGNT_NORMEDGEV_AVG)
330                       END DO
331                       DO K1=1,3
332                         WEAR_LOCAL(K1)=ZERO
333                         DO K2=1,3
334                          WEAR_LOCAL(K1)= WEAR_LOCAL(K1)+ALOCAL(K2,K1)*
                               WEAR_GLOBAL(K2)
335                         END DO
336                       END DO
337                     DO K1=1,NDIM
338                       ULOCAL(K1)=ULOCAL(K1)+WEAR_LOCAL(K1)
339                     END DO
340   !                        ULOCAL(3) = ULOCAL(3) - ABL_APPLIED
341   C                 VERIFY ULOCAL
342                       DO K2=1,NDIM
343                         UGLOBAL(K2)=ZERO
344                        DO K3=1,NDIM
345                         UGLOBAL(K2)=UGLOBAL(K2)+ALOCAL(K2,K3)*ULOCAL(K3)
346                        END DO
347                       END DO
348                   ELSEIF(NELEMS_CORRECT.EQ.1) THEN
349                   NELEMS_CORRECT2 = 0
350                     DO K1=1,NR
351                       DO K2= 2,9
352                         IF (NODE.EQ.ALLELEM_TRANS(K1,K2)) THEN
353                           NELEMS_CORRECT2 = NELEMS_CORRECT2 + 1
354                             DO K3 = 1, 8
355                               NODESELEM(NELEMS_CORRECT2, K3) = ALLELEM_TRANS
                                    (K1,K3+1)
356                             END DO
357                         ENDIF
358                       END DO
359                     END DO
360   C                 1. FIND NODES IN ADAPTIVE MESH CONSTRAINT DOMAIN
361                      DO K1=1,NELEMS_CORRECT
362                       C1=ZERO
363                        DO K3=1,8
364                         DO K2=1,NP
365                           IF(NODESELEM(K1,K3).EQ.ALLCONSTRAINTNODES(K2).AND.
                                NODESELEM(K1,K3).NE.NODE) THEN
366                            C1=C1+1
367                            SURF_NODES_CORNER(C1)=NODESELEM(K1,K3)
368                           END IF
369                         END DO
370                        END DO
371                      END DO
372   C                 2.1 FIND COORDINATES OF SURF_NODES_CORNER
```

```
373                   DO K1=1,3
374                     CALL GETVRN(SURF_NODES_CORNER(K1),'COORD',ARRAY,JRCD,
                            JGVBLOCK,LTRN)
375                      DO K2=1,3
376                       COORDS_SURF_CORNER(K1,K2)=ARRAY(K2)
377                      END DO
378                   END DO
379 C              2.2 FIND COORDINATES OF CURRENT NODE
380                   CALL GETVRN(NODE,'COORD',ARRAY,JRCD,JGVBLOCK,LTRN)
381                    DO K3=1,3
382                     NODE_COORDS(K3)=ARRAY(K3)
383                    END DO
384 C              3. FIND VECTORS TO ALL SURF_NODES_CORNER
385                    DO K1=1,3
386                      DO K2=1,3
387                        V_SURFNODES_C(K1,K2)=COORDS_SURF_CORNER(K1,K2)-
                              NODE_COORDS(K2)
388                      END DO
389                      MGNT_VSURFNODES_C(K1)=SQRT(V_SURFNODES_C(K1,1)**2+
                           V_SURFNODES_C(K1,2)**2+
390                                                V_SURFNODES_C(K1,3)**2)
391                    END DO
392 C              4. FIND SHORTER VECTORS
393                       DO K1=1,2
394                        DO K2=K1+1,3
395                         IF(MGNT_VSURFNODES_C(K1).GT.MGNT_VSURFNODES_C(K2)
                              ) THEN
396                          DO K3=1,3
397                           TEMP_V_C(K3)=V_SURFNODES_C(K1,K3)
398                           V_SURFNODES_C(K1,K3)=V_SURFNODES_C(K2,K3)
399                           V_SURFNODES_C(K2,K3)=TEMP_V_C(K3)
400                          END DO
401                           TEMP_M_C=MGNT_VSURFNODES_C(K1)
402                           MGNT_VSURFNODES_C(K1)=MGNT_VSURFNODES_C(K2)
403                           MGNT_VSURFNODES_C(K2)=TEMP_M_C
404                         END IF
405                        END DO
406                       END DO
407 C               4.1 FIND UNIT VECTORS
408                     DO K1=1,3
409                       DO K2=1,3
410                        V_SURFNODES_UNIT_C(K1,K2)=V_SURFNODES_C(K1,K2)/
                             MGNT_VSURFNODES_C(K1)
411                       END DO
412                     END DO
413 C              5. FIND SUBSURFACE NODES
414                    C8= ZERO
415                     DO K1=1,NELEMS_CORRECT
416                      DO K2=1,8
417                       C7=ZERO
418                        DO K3=1,NP
419                         IF(NODESELEM(K1,K2).NE.ALLCONSTRAINTNODES(K3))THEN
420                           C7=C7+1
```

```
421                             IF(C7.EQ.NP) THEN
422                               C8=C8+1
423                               SUB_NODES_C(C8)=NODESELEM(K1,K2)
424                             END IF
425                           END IF
426                          END DO
427                        END DO
428                      END DO
429 C             6. FIND VECTOR TO ARBITRARY SUBSURFACE NODE
430                      CALL GETVRN(SUB_NODES_C(1),'COORD',ARRAY,JRCD,JGVBLOCK,
                            LTRN)
431                       DO K3=1,3
432                        V_SUBNODE_C(K3)= ARRAY(K3)-NODE_COORDS(K3)
433                       END DO
434                        MGNT_VSUBNODE_C=SQRT(V_SUBNODE_C(1)**2+V_SUBNODE_C
                              (2)**2+V_SUBNODE_C(3)**2)
435                       DO K4=1,3
436                        V_SUBNODE_UNIT_C(K4)=V_SUBNODE_C(K4)/MGNT_VSUBNODE_C
437                       END DO
438 C             7. FIND CROSS PRODUCT
439                     CROSS1_X_C=V_SURFNODES_UNIT_C(1,2)*V_SURFNODES_UNIT_C
                           (2,3)-V_SURFNODES_UNIT_C(1,3)*V_SURFNODES_UNIT_C(2,2)
440                     CROSS1_Y_C=V_SURFNODES_UNIT_C(1,3)*V_SURFNODES_UNIT_C
                           (2,1)-V_SURFNODES_UNIT_C(1,1)*V_SURFNODES_UNIT_C(2,3)
441                     CROSS1_Z_C=V_SURFNODES_UNIT_C(1,1)*V_SURFNODES_UNIT_C
                           (2,2)-V_SURFNODES_UNIT_C(1,2)*V_SURFNODES_UNIT_C(2,1)
442                     NORM_CORNER_V(1)=CROSS1_X_C
443                     NORM_CORNER_V(2)=CROSS1_Y_C
444                     NORM_CORNER_V(3)=CROSS1_Z_C
445                     MGNT_NORMCORNERV=SQRT(NORM_CORNER_V(1)**2+NORM_CORNER_V
                           (2)**2+NORM_CORNER_V(3)**2)
446 C             8. FIND ANGLE BETWEEN NORM_CORNER_V AND V_SUBNODE_UNIT_C
447                      THETA_C=ACOS((NORM_CORNER_V(1)*V_SUBNODE_UNIT_C(1)+
448                                    NORM_CORNER_V(2)*V_SUBNODE_UNIT_C(2)+
449       1                            NORM_CORNER_V(3)*V_SUBNODE_UNIT_C(3))/(
      MGNT_NORMCORNERV))
450                      IF(THETA_C.GT.PI/2) THEN
451                        NORM_CORNER_V(:)=-NORM_CORNER_V(:)
452                      END IF
453 C              VERIFY ANGLE
454                      THETA_VERIF_C=ACOS((NORM_CORNER_V(1)*
                           V_SUBNODE_UNIT_C(1)+NORM_CORNER_V(2)*
                           V_SUBNODE_UNIT_C(2)+
455       1                            NORM_CORNER_V(3)*V_SUBNODE_UNIT_C(3))/(
      MGNT_NORMCORNERV))
456 C             9. CALULATE WEAR IN DIRECTION OF NORM_CORNER_AVG
457                     DO K1=1,3
458                        WEAR_GLOBAL_C(K1) = ABL_APPLIED*(NORM_CORNER_V(K1)/
                           MGNT_NORMCORNERV)
459                     END DO
460                     DO K1=1,NDIM
461                       WEAR_LOCAL_C(K1)=ZERO
462                       DO K2=1,NDIM
```

```
463                       WEAR_LOCAL_C(K1)= WEAR_LOCAL_C(K1)+ALOCAL(K2,K1)*
                              WEAR_GLOBAL_C(K2)
464                     END DO
465                   END DO
466                   DO K1=1,NDIM
467                     ULOCAL(K1)=ULOCAL(K1)+WEAR_LOCAL_C(K1)
468                   END DO
469 C               VERIFY ULOCAL
470                   DO K2=1,NDIM
471                     UGLOBAL(K2)=ZERO
472                     DO K3=1,NDIM
473                       UGLOBAL(K2)=UGLOBAL(K2)+ALOCAL(K2,K3)*ULOCAL(K3)
474                     END DO
475                   END DO
476             ELSEIF(NELEMS_CORRECT.NE.1.AND.NELEMS_CORRECT.NE.2) THEN
477                 ULOCAL(3)=ULOCAL(3)-ABL_APPLIED
478 C               VERIFY ULOCAL
479                   DO K2=1,NDIM
480                     UGLOBAL(K2)=ZERO
481                     DO K3=1,NDIM
482                       UGLOBAL(K2)=UGLOBAL(K2)+ALOCAL(K2,K3)*ULOCAL(K3)
483                     END DO
484                   END DO
485             ENDIF
486 C           OUTPUT
487              WRITE(6,*) '----------------------------------------'
488              WRITE(6,*) 'ULOCAL CHANGED'
489              WRITE(6,*) 'KINC: ', KINC
490              WRITE(6,*) 'KMESHSWEEP: ', KMESHSWEEP
491              WRITE(6,*) 'NODE: ', NODE
492              WRITE(6,*) 'DTIME: ', DTIME
493              WRITE(6,*) 'ABL_APPLIED: ', ABL_APPLIED
494              WRITE(6,*) 'ULOCAL(3): ', ULOCAL(3)
495             ENDIF
496           DEALLOCATE(NODESELEM)
497       ENDIF
498 C
499       RETURN
500       END SUBROUTINE UMESHMOTION
```

# Appendix C

# UMESHMOTION Subroutine 2

## C.1 UMESHMOTION Subroutine 3: Node shift direction follows outer surface

```
1       INCLUDE 'umeshmotion_module.f'
2       INCLUDE 'umeshmotion_functions.f'
3 C
4       SUBROUTINE UMESHMOTION(UREF,ULOCAL,NODE,NNDOF,LNODETYPE,ALOCAL,
5      1 NDIM,TIME,DTIME,PNEWDT,KSTEP,KINC,KMESHSWEEP,JMATYP,JGVBLOCK,
6      2 LSMOOTH)
7 C
8       USE MODULE_UMESHMOTION ! LOAD MODULE
9       IMPLICIT NONE
10 C
11 C     DECLARATION OF VARIABLES
12 C
13      INTEGER,PARAMETER::NPRECD=2 ! PARAMETER FOR DOUBLE PRECISSION
14 C
15      INTEGER NODE,NNDOF,LNODETYPE,NDIM,KSTEP,KINC,KMESHSWEEP,LSMOOTH,
16     1 FINDINDEX,FIRSTRUN,KINC_MEM, ZERO, K1, K2
17 C
18      REAL*8 UREF,DTIME,PNEWDT
19 C
20      REAL*8 ULOCAL(NDIM),ALOCAL(NDIM,*),TIME(2),JMATYP(*),JGVBLOCK(*),
21     1 ARRAY(15)
22 C
23 C     INITIAL VALUES FOR FIRST SUBROUTINE RUN
24      DATA FIRSTRUN /-1/
25      DATA KINC_MEM /0/
26 C
27 C     ****************************************************************
28 C     * MAIN SECTION                                                 *
29 C     ****************************************************************
```

```
30  C
31  C      ONLY FOR FIRST SUBROUTINE RUN
32         IF (FIRSTRUN.EQ.-1) THEN
33            WRITE(6,*) 'FIRSTRUN'
34            WRITE(6,*) 'KINC:', KINC
35            ZERO = 0.0
36            FIRSTRUN=1
37  C         PARAMETER FOR FIRST STEP
38            FIRSTSTEP=KSTEP
39  C         GET WORKING DIRECTORY
40            CALL GETOUTDIR(OUTDIR,LENOUTDIR)
41  C         GET JOBNAME
42            CALL GETJOBNAME(JOBNAME,LENJOBNAME)
43  C         OUTOUT TO FILE
44            WRITE(6,*) ''
45            WRITE(6,*) '****************************************'
46            WRITE(6,*) 'STARTED UMESHMOTION'
47            WRITE(6,*) 'FOR JOB: ',JOBNAME
48            WRITE(6,*) 'IN WORKING DIRECTORY: ',OUTDIR
49            WRITE(6,*) '****************************************'
50            WRITE(6,*) ''
51            FIRSTNODE=NODE
52            NODE_I=0
53  C            DETERMINE TOTAL NUMBER OF LINES
54               NP=ZERO
55               STAT_R=ZERO
56               OPEN(400,FILE=OUTDIR(1:LENOUTDIR)//'/'//'aleConstraintNodes.
                    txt',
57        1            STATUS='OLD',ACTION='READ',IOSTAT=STAT_O)
58                DO K1=1,MAXRECS
59                 READ(400,*,IOSTAT=STAT_R) CONSTRAINTNODESCOUNTER
60                    IF(STAT_R.NE.0) EXIT
61                    IF (K1.EQ.MAXRECS) THEN
62                       write(7,*) 'Error: Maximum number of records exceeded
                            ...'
63                       write(7,*) 'Exiting now...'
64                       CALL XIT
65                    ENDIF
66                     NP = NP + 1
67                END DO
68                REWIND(400)
69                 ALLOCATE(ALLCONSTRAINTNODES(NP))
70                 READ(400,*,IOSTAT=STAT_R) ALLCONSTRAINTNODES
71                  WRITE(6,*) 'ALLCONSTRAINTNODES: ', ALLCONSTRAINTNODES
72                 CLOSE(400)
73               NR=ZERO
74               STAT_R=ZERO
75               OPEN(300,FILE=OUTDIR(1:LENOUTDIR)//'/'//'elementConnectivity.
                    txt',
76        1            STATUS='OLD',ACTION='READ',IOSTAT=STAT_O)
77                DO K1=1,MAXRECS
78                 READ(300,*,IOSTAT=STAT_R) FIRST
79                    IF(STAT_R.NE.0) EXIT
```

```
80                    IF (K1.EQ.MAXRECS) THEN
81                       write(7,*) 'Error: Maximum number of records exceeded
                             ...'
82                       write(7,*) 'Exiting now...'
83                       CALL XIT
84                    ENDIF
85                     NR = NR + 1
86                END DO
87                REWIND(300)
88                 ALLOCATE(ALLELEM(9,NR))
89                 ALLOCATE(ALLELEM_TRANS(NR,9))
90                 READ(300,*,IOSTAT=STAT_R) ALLELEM
91                 REWIND(300)
92                 ALLELEM_TRANS=TRANSPOSE(ALLELEM)
93                 WRITE(6,*) 'ALLELEM_TRANS: ', ALLELEM_TRANS
94                 CLOSE(300)
95  C           READ ABLATION DATA
96                STAT_R=ZERO
97                OPEN(500,FILE=OUTDIR(1:LENOUTDIR)//'/'//'ablation.txt',
98                    STATUS='OLD',ACTION='READ',IOSTAT=STAT_O)
99                DO K1=1,MAXRECS
100                 READ(500,*,IOSTAT=STAT_R) ABL
101                    IF(STAT_R.NE.0) EXIT
102                    IF (K1.EQ.MAXRECS) THEN
103                       write(7,*) 'Error: Maximum number of records exceeded
                             ...'
104                       write(7,*) 'Exiting now...'
105                       CALL XIT
106                    ENDIF
107                     NQ = NQ + 1
108               END DO
109                REWIND(500)
110                 ALLOCATE(TOTAL_ABLATION(2,NQ))
111                 ALLOCATE(TOTAL_ABLATION_TRANS(NQ,2))
112                 READ(500,*,IOSTAT=STAT_R) TOTAL_ABLATION
113                 WRITE(7,*) 'READ TOTAL_ABLATION'
114                 TOTAL_ABLATION_TRANS = TRANSPOSE(TOTAL_ABLATION)
115                  DO K3=1,NQ
116                    WRITE(7,*) 'TOTAL_ABLATION: ', TOTAL_ABLATION_TRANS(K3
                          ,:)
117                  END DO
118               REWIND(500)
119               CLOSE(500)
120       ENDIF
121
122
123       IF (KMESHSWEEP.EQ.0) THEN
124 C       GET WEAR DEPTH
125         DO K1 = 1, NQ
126            IF(NODE.EQ.TOTAL_ABLATION_TRANS(K1,1)) THEN
127               ABL_APPLIED = DTIME*TOTAL_ABLATION_TRANS(K1,2)
128            END IF
129         END DO
```

```
130           IF (NDIM.EQ.2) THEN
131              ULOCAL(2)=ULOCAL(2)-ABL_APPLIED
132           ELSEIF (NDIM.EQ.3) THEN
133   C       GET ELEMENTS CONNECTED TO NODE
134           NELEMS_CORRECT = 0
135                 DO K1=1,NR
136                   DO K2= 2, 9
137                      IF (NODE.EQ.ALLELEM_TRANS(K1,K2)) THEN
138                          NELEMS_CORRECT = NELEMS_CORRECT + 1
139                      ENDIF
140                   END DO
141                 END DO
142                 WRITE(6,*) 'NODE: ', NODE
143                 WRITE(6,*) 'NELEMS_CORRECT: ', NELEMS_CORRECT
144   C           ALLOCATE SIZE FOR NODES ARRAY
145                ALLOCATE(NODESELEM(NELEMS_CORRECT,8))
146                 NELEMS_CORRECT2 = 0
147                 DO K1=1,NR
148                    DO K2= 2,9
149                       IF (NODE.EQ.ALLELEM_TRANS(K1,K2)) THEN
150                          NELEMS_CORRECT2 = NELEMS_CORRECT2 + 1
151                            DO K3 = 1, 8
152                              NODESELEM(NELEMS_CORRECT2, K3) = ALLELEM_TRANS
                                     (K1,K3+1)
153                            END DO
154                       ENDIF
155                    END DO
156                 END DO
157                   WRITE(6,*) 'NODESDELEM ASSIGNED'
158            IF (NELEMS_CORRECT.EQ.2) THEN
159   C         EDGE NODE
160   C          FIND NODES COMMON TO BOTH ELEMENTS
161                 C2=ZERO
162                    DO K2=1,8
163                        DO K4=1,8
164                           IF(NODESELEM(1,K2).EQ.NODESELEM(2,K4))THEN
165                           C2=C2+1
166                           COMMON_NODES(C2)=NODESELEM(1,K2)
167                           END IF
168                        END DO
169                    END DO
170                    WRITE(6,*) 'COMMON_NODES', COMMON_NODES(:)
171   C           FIND NODES THAT DO NOT BELONG TO ADPATIVE CONSTRAINT REGION
172                    C8= ZERO
173                      DO K1=1,C2
174                        C7=ZERO
175                        DO K2=1,NP
176                          IF(COMMON_NODES(K1).NE.ALLCONSTRAINTNODES(K2))THEN
177                            C7=C7+1
178                             IF(C7.EQ.NP) THEN
179                               C8=C8+1
180                               SUB_NODES(C8)=COMMON_NODES(K1)
181                             END IF
```

```
182                         END IF
183                       END DO
184                     END DO
185 C                     WRITE(6,*) 'SUB_NODES', SUB_NODES(:)
186 C                     WRITE(6,*) 'C8: ', C8
187 C           FORM VECTORS TO ALL SUBSURFACE NODES
188 C                1. FIND COORDINATES OF SUBSURFACE NODES
189                       LTRN=0
190                       DO K1=1,2
191                       CALL GETVRN(SUB_NODES(K1),'COORD',ARRAY,JRCD,JGVBLOCK
                              ,LTRN)
192                         DO K2=1,3
193                           SUBNODE_COORDS(K1,K2) = ARRAY(K2)
194                         END DO
195                       END DO
196                       DO K1 = 1,2
197 C                         WRITE(6,*) 'SUBNODE_COORDS', SUBNODE_COORDS(K1,:)
198                       ENDDO
199 C                  2. FIND COORDINATES OF CURRENT NODE
200                       CALL GETVRN(NODE,'COORD',ARRAY,JRCD,JGVBLOCK,LTRN)
201                         DO K3=1,3
202                           NODE_COORDS(K3)=ARRAY(K3)
203                         END DO
204 C                        WRITE(6,*) 'NODE_COORDS', NODE_COORDS(:)
205 C                  3. FIND ALL SUBURFACE VECTORS
206                       DO K4=1,2
207                        DO K5=1,3
208                           V_SUBNODES(K4,K5)= SUBNODE_COORDS(K4,K5)-
                                 NODE_COORDS(K5)
209                        END DO
210                           MGNT_VSUBNODES(K4)=SQRT(V_SUBNODES(K4,1)**2+
211                           V_SUBNODES(K4,2)**2+V_SUBNODES(K4,3)**2)
212                       END DO
213 C                   3.1 FIND SHORTER VECTOR
214                       IF(MGNT_VSUBNODES(1).GT.MGNT_VSUBNODES(2)) THEN
215                          DO K1 = 1,3
216                           SHORT_VECTOR_EDGE(K1) = V_SUBNODES(2,K1)
217                          END DO
218                       ELSE
219                          DO K1 = 1,3
220                           SHORT_VECTOR_EDGE(K1) = V_SUBNODES(1,K1)
221                          END DO
222                       ENDIF
223 C                      WRITE(6,*) 'SHORT_VECTOR_EDGE: ', SHORT_VECTOR_EDGE
    (:)
224 C            FIND UNIT VECTOR
225                    MGNT_VSHORTVECTOREDGE=SQRT(SHORT_VECTOR_EDGE(1)**2+
226                                  SHORT_VECTOR_EDGE(2)**2+SHORT_VECTOR_EDGE
                                      (3)**2)
227                       DO K4=1,3
228                        SHORT_VECTOR_EDGE_UNIT(K4)=SHORT_VECTOR_EDGE(K4)/
                             MGNT_VSHORTVECTOREDGE
229                       END DO
```

```
230 C                    WRITE(6,*) 'SHORT_VECTOR_EDGE_UNIT: ',
    SHORT_VECTOR_EDGE_UNIT(:)
231 C          CALULATE WEAR IN DIRECTION OF MGNT_VSHORTVECTOREDGE
232                   DO K1=1,3
233                      WEAR_GLOBAL(K1) = ABL_APPLIED*SHORT_VECTOR_EDGE_UNIT
                            (K1)
234                   END DO
235 C                   WRITE(6,*) 'WEAR_GLOBAL: ', WEAR_GLOBAL(:)
236                   DO K1=1,3
237                     WEAR_LOCAL(K1)=ZERO
238                     DO K2=1,3
239                      WEAR_LOCAL(K1)= WEAR_LOCAL(K1)+ALOCAL(K2,K1)*
                            WEAR_GLOBAL(K2)
240                     END DO
241                   END DO
242 C                  WRITE(6,*) 'WEAR_LOCAL: ', WEAR_LOCAL(:)
243                DO K1=1,NDIM
244                   ULOCAL(K1)=ULOCAL(K1)+WEAR_LOCAL(K1)
245                END DO
246 C               WRITE(6,*) 'ULOCAL: ', ULOCAL(:)
247           ELSEIF (NELEMS_CORRECT.EQ.1) THEN
248 C         CORNER NODE
249 C         FIND NODES THAT DO NOT BELONG TO ADPATIVE CONSTRAINT REGION
250             C2 = ZERO
251             DO K1 = 1,8
252               C1 = ZERO
253               DO K2 = 1,NP
254                  IF (NODESELEM(1,K1).NE.ALLCONSTRAINTNODES(K2)) THEN
255                     C1 = C1 + 1
256                  ENDIF
257               END DO
258               IF (C1.EQ.NP) THEN
259                  C2 = C2 + 1
260                  SUB_NODES_C(C2) = NODESELEM(1,K1)
261               ENDIF
262             END DO
263             WRITE(6,*) 'C2: ', C2
264             WRITE(6,*) 'SUB_NODES_C: ', SUB_NODES_C(:)
265 C        FIND ALL VECTORS TO SUB_NODES_C
266 C            1. FIND COORDINATES OF SUBSURFACE NODES
267                   LTRN=0
268                   DO K1=1,4
269                   CALL GETVRN(SUB_NODES_C(K1),'COORD',ARRAY,JRCD,
                          JGVBLOCK,LTRN)
270                     DO K2=1,3
271                       SUBNODE_COORDS_C(K1,K2) = ARRAY(K2)
272                     END DO
273                   END DO
274                   DO K1 = 1,4
275                      WRITE(6,*) 'SUBNODE_COORDS_C: ', SUBNODE_COORDS_C(
                           K1,:)
276                   END DO
277 C              2. FIND COORDINATES OF CURRENT NODE
```

```
278                       CALL GETVRN(NODE,'COORD',ARRAY,JRCD,JGVBLOCK,LTRN)
279                         DO K3=1,3
280                           NODE_COORDS(K3)=ARRAY(K3)
281                         END DO
282                         WRITE(6,*) 'NODE_COORDS: ', NODE_COORDS(:)
283 C               3. FIND ALL SUBURFACE VECTORS
284                       DO K4=1,4
285                        DO K5=1,3
286                           V_SUBNODES_C(K4,K5)= SUBNODE_COORDS_C(K4,K5)-
                                NODE_COORDS(K5)
287                        END DO
288                           MGNT_VSUBNODES_C(K4)=SQRT(V_SUBNODES_C(K4,1)**2+
289                           V_SUBNODES_C(K4,2)**2+V_SUBNODES_C(K4,3)**2)
290                       END DO
291                       DO K1=1,4
292                          WRITE(6,*) 'V_SUBNODES_C: ', V_SUBNODES_C(K1,:)
293                       END DO
294                       WRITE(6,*) 'MGNT_VSUBNODES_C: ', MGNT_VSUBNODES_C(:)
295 C               4. FIND SHORTEST SUBSURFACE VECTOR
296                       MINSUBVECTORC = MIN(MGNT_VSUBNODES_C(1),
                             MGNT_VSUBNODES_C(2),MGNT_VSUBNODES_C(3),
                             MGNT_VSUBNODES_C(4))
297                       WRITE(6,*) 'MINSUBVECTORC: ', MINSUBVECTORC
298                       DO K4=1,4
299                         IF (MGNT_VSUBNODES_C(K4).EQ.MINSUBVECTORC) THEN
300                          DO K5 = 1,3
301                           SHORT_VECTOR_CORNER(K5) = V_SUBNODES_C(K4,K5)
302                          END DO
303                         END IF
304                       END DO
305                        WRITE(6,*) 'SHORT_VECTOR_CORNER: ',
                              SHORT_VECTOR_CORNER(:)
306 C              FIND UNIT VECTOR
307                      MGNT_VSHORTVECTORCORNER=SQRT(SHORT_VECTOR_CORNER(1)**2+
308                      SHORT_VECTOR_CORNER(2)**2+SHORT_VECTOR_CORNER(3)**2)
309                        DO K4=1,3
310                         SHORT_VECTOR_CORNER_UNIT(K4)=SHORT_VECTOR_CORNER(K4)
                              /MGNT_VSHORTVECTORCORNER
311                       END DO
312                        WRITE(6,*) 'SHORT_VECTOR_CORNER_UNIT: ',
                              SHORT_VECTOR_CORNER_UNIT(:)
313 C           CALULATE WEAR IN DIRECTION OF MGNT_VSHORTVECTOREDGE
314                      DO K1=1,3
315                         WEAR_GLOBAL(K1) = ABL_APPLIED*
                              SHORT_VECTOR_CORNER_UNIT(K1)
316                      END DO
317                      WRITE(6,*) 'WEAR_GLOBAL: ', WEAR_GLOBAL(:)
318                      DO K1=1,3
319                        WEAR_LOCAL(K1)=ZERO
320                        DO K2=1,3
321                         WEAR_LOCAL(K1)= WEAR_LOCAL(K1)+ALOCAL(K2,K1)*
                              WEAR_GLOBAL(K2)
322                        END DO
```

```
323                     END DO
324                     WRITE(6,*) 'WEAR_LOCAL: ', WEAR_GLOBAL(:)
325                   DO K1=1,NDIM
326                     ULOCAL(K1)=ULOCAL(K1)+WEAR_LOCAL(K1)
327                   END DO
328                   WRITE(6,*) 'ULOCAL: ', ULOCAL(:)
329               ELSEIF(NELEMS_CORRECT.NE.1.AND.NELEMS_CORRECT.NE.2) THEN
330                   ULOCAL(3)=ULOCAL(3)-ABL_APPLIED
331 C                 VERIFY ULOCAL
332                     DO K2=1,NDIM
333                       UGLOBAL(K2)=ZERO
334                      DO K3=1,NDIM
335                       UGLOBAL(K2)=UGLOBAL(K2)+ALOCAL(K2,K3)*ULOCAL(K3)
336                      END DO
337                     END DO
338 C               ENDIF CORNER/EDGE/SURFACE
339 C               OUTPUT
340                  WRITE(6,*) '----------------------------------------'
341                  WRITE(6,*) 'ULOCAL CHANGED'
342                  WRITE(6,*) 'KINC: ', KINC
343                  WRITE(6,*) 'KMESHSWEEP: ', KMESHSWEEP
344                  WRITE(6,*) 'NODE: ', NODE
345                  WRITE(6,*) 'DTIME: ', DTIME
346                  WRITE(6,*) 'ABL_APPLIED: ', ABL_APPLIED
347               ENDIF
348 C              ENDIF NDIM=3
349             ENDIF
350             DEALLOCATE(NODESELEM)
351 C            ENDIF KMESHSWEEP = 0
352           ENDIF
353 C
354         RETURN
355         END SUBROUTINE UMESHMOTION
```