Graz University of Technology

Masterarbeit

Automatische Schweißnahtgenerierung und Spannungsauswertung in der Schweißnaht bei FE-Plattenmodellen in Abaqus

eingereicht am

Institut für Baustatik
der
Technischen Universität Graz
im
Oktober 2015

Verfasser: Jürgen Krobath
Grazer Straße 44c
A-8045 Graz-Andritz

Betreuer: Univ.-Prof. Dr.-Ing. habil. Thomas-Peter Fries
Dipl.-Ing. Dr.techn Jürgen Zechner

Eidesstattliche Erklärung

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

Ort, Datum	Unterschrift

Danksagung

In erster Linie möchte ich mich bei meinem Betreuer Herrn Dipl.-Ing. Dr.techn. Jürgen Zechner für die tolle Unterstützung während der Masterarbeit recht herzlich bedanken. Ohne ihn wäre die Umsetzung der Arbeit in dieser Form nicht möglich gewesen.

Für die spannende Aufgabenstellung dieser Masterarbeit bedanke ich mich bei der Firma Sandvik/Leoben. Insbesondere bei Herrn Dipl.-Ing. Gottfried Poier, welcher mir bei vielen Fragen mit guten Ratschlägen weiterhelfen konnte.

Weiters möchte ich mich bei Herrn Univ.-Prof. Dr.-Ing. habil. Thomas-Peter Fries sowie dem gesamten Team des Baustatik Instituts, für die letzten Jahre in denen ich als Studienassistent tätig war, bedanken. Es war eine schöne Zeit mit einigen interessanten Tätigkeiten.

Großem Dank gilt auch meinen Freunden und Studienkollegen und vor allem der „Supergruppe“, die meine Studienzeit zu einem sehr schönen Teilabschnitt meines Lebens gemacht haben.

Zu guter Letzt möchte ich mich bei meiner Familie herzlichst bedanken, dass sie mir das Studium ermöglichten und mir immer mit tatkräftiger Unterstützung zur Seite standen, auch in jenen Zeiten in denen es nicht ganz nach Plan verlief.

Kurzfassung

In dieser Arbeit wurden, in Kooperation mit der Firma *Sandvik Mining Construction Materials Handling GmbH* & *Co KG* Programme zur automatischen Schweißnahtgenerierung und zur Spannungstransformation in den angrenzenden Elementen für FE-Plattenmodellen in Abaqus entwickelt. Die erstellten Skript-Dateien dienen als Grundlage für die Bemessung und Auslegung von Schweißnahtdetails in Hinsicht auf deren Ermüdungsfestigkeit.

Der entwickelte Algorithmus berechnet den Winkel zwischen den Normalvektoren zweier benachbarter Schalen-Elemente und vergleicht ihn mit einer, vom Benutzer eingegebenen Toleranz. Eine Schweißnaht wird erstellt wenn der berechnete Winkel größer bzw. gleich dieser Toleranz ist. Die potentiellen Schweißnähte werden im Modell durch Linienelemente visualisiert und zu Element-Sets zusammengefasst. Ein Element-Set stellt eine einzelne durchgehende Schweißnaht dar. Überflüssig generierte Linienelemente können vom Benutzer direkt in Abaqus markiert und durch das Ausführen eines Skripts auch wieder entfernt werden. Ebenso können mithilfe des selben Skripts ergänzende Linienelemente generiert werden, welche bei der Spannungstransformation mit berücksichtigt werden. Die Schweißnahtgenerierung erfolgt im Abaqus Präprozessor. Nach fertiggestellter Analyse werden im Abaqus Postprozessor die Spannungen in den betroffenen Schalen-Elementen in Richtung der Naht und in Richtung normal zur Naht transformiert.

In der schriftlichen Ausarbeitung werden Grundlagen, die für das Erstellen der Python-Skripten notwendig sind, zusammengefasst. Die einzelnen Algorithmen werden beschrieben und die Funktionalität des Programmcodes aufgezeigt.

Schlagwörter:

FEM, Abaqus, Schalen, Kantendetektion, Spannungstransformation, Python

Abstract

This thesis has been realized in cooperation with *Sandvik Mining Construction Materials Handling GmbH & Co KG*. The aim was to create a software for the automatic generation of welding seams on FE-plate models in Abaqus. Subsequently, the stresses of the surrounding plate elements are transformed with respect to the orientation of the welding seam. The created script files serve as a basis for the design of welds related to their structural durability.

The implemented algorithm determines the angle between the normal vectors of two adjacent elements and compares it with a given tolerance which is entered by the user. A weld is created if the angle is greater than or equal to this tolerance. The welds are visualized by line elements and assembled to element sets. An element set provides a single continuous weld. Superfluous line elements can be marked and subsequently removed by running a script in Abaqus again. By using the same script additional line elements can be created, which are taken into account by the stress transformation. The weld generation takes place in the Abaqus preprocessor. After analysis, the stresses in the affected shell elements are transformed in direction of the weld and normal to it.

The Topics which have been essential for the development of these scripts are summarized in the written report. Furthermore, the essential algorithms are discribed and the functionality of the program code is shown.

Keywords:

FEM, Abaqus, shells, edge detection, stress transformation, Python

Inhaltsverzeichnis

1 Einleitung

Mit der Entwicklung immer leistungsstärkerer Computer und dem Streben nach Wirtschaftlichkeit gewannen numerischen Berechnungsverfahren in den letzten Jahrzehnten immer mehr an Bedeutung. Das wohl am weitest verbreitete numerische Berechnungsverfahren zur Lösung von partiellen Differentialgleichungen ist die Methode der Finiten Elemente (FEM). Die FEM findet Anwendung in den unterschiedlichsten Disziplinen der Physik und des Ingenieurwesens. Durch die rasante Weiterentwicklung von Computerhardware und Software werden die Problemstellungen, die man im Stande ist zu lösen immer komplexer und die Anzahl der Freiheitsgrade bzw. der Elemente im Modell immer höher. Das Potential zur Automatisierung von bestimmten Abläufen im Bereich des Preprocessing, wie auch im Postprozessing und somit einer wirtschaftlicheren Bearbeitung diverser Problemstellungen ist speziell bei sehr großen Modellen nicht außer Acht zu lassen.

1.1 Motivation

Eine Disziplin in der große Modelle erstellt, analysiert und Ergebnisse ausgewertet und verarbeitet werden ist der Stahlbau. In Abbildung 1.1 ist ein Schaufelradbagger und dessen Dreheinrichtung als FEM-Modell dargestellt. Solche Stahlbauten werden dynamisch belastet und neben der normalen Spannungsberechnung auch immer einer Ermüdungsberechnung unterzogen. Im Hinblick auf eine möglichst lange Lebensdauer (Betriebsdauer) dieser Bauteile sind unter anderem die Ausführung der Schweißnahtdetails von großer Bedeutung. Die zulässigen Spannungen in den Schweißnähten sind unter anderem von der Lasteinwirkungsrichtung abhängig. So können in Schweißnahtrichtung höhere zulässige Spannungen angesetzt werden als normal dazu [1].

Abbildung 1.1: Beispiel eines Schaufelradbaggers und dessen Dreheinrichtung als FEM-Modell

Für die Bemessung und Auslegung von Schweißnähten ist es meist ausreichend, nur das idealisierte Plattenmodell ohne Schweißnähte zu modellieren. Es sind dabei die Spannungen in Richtung der Schweißnahtachse und orthogonal dazu sowie die zugehörige Schubspannung in den Stahlplatten für die Bemessung der Nähte erforderlich. In der FEM-Software Abaqus [2] sind die Hauptspannungsrichtungen von Schalenelementen durch die Projektion der globalen x-Achse auf das jeweilige Element bestimmt. Daher müssen diese Spannungen in Elemente die an einer Schweißnaht liegen transformiert werden. Eine manuelle Herangehensweise kann schon bei kleineren Modellen sehr zeitaufwändig sein. Es bietet sich daher an, den Schritt der Spannungstransformation in den betroffenen Elementen zu automatisieren um Zeit und damit Kosten einzusparen. Für die Automatisierung bestimmter wiederkehrender Abläufe bietet Abaqus Schnittstellen zu den Programmiersprachen C++ und

Python [3]. So können z.B. der Modellaufbau bzw. Teile davon aber auch Auswertung und Bearbeitung von Ergebnissen nach der Analyse durch das Ausführen eines Skripts automatisiert erfolgen.

1.2 Ziele und Aufbau der Arbeit

Ziel dieser Arbeit ist die Entwicklung eines Abaqus Benutzer-Programms zur automatischen Transformation der Hauptspannungsrichtungen in Richtung der jeweiligen Schweißnahtachse und normal zu dieser. Es sei erwähnt, dass sich die implementierten Algorithmen auf das Verwenden von Schalen-Elemente mit linearem und bilinearem Verschiebungsansatz beschränken. Die Transformation erfolgt nur in Elementen, die an eine Schweißnaht angrenzen.

Für das Markieren der Schweißnähte soll im Präprozessor eine Kantendetektion am Finite-Elemente-Netz des Modells sorgen. Es sollen jene Elementkanten und die dazugehörigen Elemente gefunden werden, an welchen in der Bauteilherstellung Schweißnähte aus konstruktiven Gründen gemacht werden müssen. Das Kriterium, ab welchem Winkel zwischen den Normalvektoren zweier benachbarter Schalenelemente eine Bauteilkante detektiert wird, soll vom Benutzer festgelegt und eingegeben werden können. Um eine visuelle Überprüfung der korrekten Lage der potentiellen Nähte zu gewährleisten werden an diesen Schalenelementkanten neue Linienelemente generiert. Diese Linienelemente werden in weiterer Folge in Gruppen, so genannte Element-Sets zusammengefasst. Jedes Element-Set repräsentiert eine einzelne Schweißnaht. Die Zusammenfassung einzelner Linienelemente zu Element-Sets soll über die Eingabe eines zweiten Parameters ermöglicht werden. Dieser Parameter legt fest, ab welchem Wert des Winkels zwischen den Richtungsvektoren zweier benachbarter neu generierter Linienelemente eine Teilung erfolgt. Ist die Kantendetektion und die automatische Generierung der Linienelemente abgeschlossen, kommt es zu einer visuellen Überprüfung. Es soll die Möglichkeit gegeben sein einzelne Linienelemente vom Modell zu entfernen bzw. hinzuzufügen. Hierfür wird ein Skript zur Verfügung gestellt, welches diese Operationen am Modell durchführt und die, für die Spannungstransformation benötigten Daten aktualisiert. Nach erfolgter FEM-Analyse des Modells, kommt es zur eigentlichen Spannungstransformation. Dazu soll ein weiteres Skript erstellt werden, welches die Spannungen in den betroffenen Schalenelementen in die Richtung der Schweißnahtachse und normal zur Achse transformiert. Diese transformierten Spannungen werden in Form von Contour-Plots dargestellt und dienen zur weiteren Bearbeitung bzw. Bemessung durch den Ingenieur.

Die Implementierung der Algorithmen erfolgt in Form von Python-Skripts, welche in Abaqus ausgeführt werden können. Die einzelnen Python-Skript-Dateien sollen der Firma *Sandvik Mining Construction Materials Handling GmbH & Co KG* [4] als Grundlage für die Bemessung und Auslegung von Schweißnahtdetails in Hinblick auf deren Ermüdungsfestigkeit dienen. Die Herstellungsprodukte von Sandvik reichen von Schaufelladern, Förderbänder und Transportern bis hin zu Bohrgeräten und Bohrköpfen.

In dieser schriftlichen Ausarbeitung erfolgt zunächst eine Zusammenfassung der, für die Bearbeitung der Problemstellung notwendigen theoretischen Grundlagen. In Kapitel 2 wird neben der Winkelberechnung zweier benachbarter Schalenelemente und der Spannungstransformation kurz auf die Methode der Finite Elemente und die Konventionen in Abaqus eingegangen. Im Kapitel 3 wird die Programmtechnische Umsetzung der einzelnen Algorithmen u.a. in Form von Ablaufdiagrammen näher erläutert und die Komplexität der Methode zum Auffinden von Nachbarelementen aufgezeigt. Im Kapitel 4 soll anhand einiger einfacher Beispiele die Leistungsfähigkeit und eventuelle Einschränkungen des Programmcodes geprüft werden. Abschließend erfolgt eine Zusammenfassung der Arbeit und ein Ausblick auf mögliche Weiterentwicklungen. Beigelegt befindet sich ein Datenträger mit allen erstellten Programmdaten.

2 Grundlagen

In diesem Kapitel sollen die Grundlagen in einem Umfang, der zum Verständnis dieser Arbeit notwendig ist, aufgezeigt werden. Im Kapitel 2.1 erfolgt eine kurz gehaltene Einführung in die Finite-Elemente-Methode. Die zu bearbeitenden Plattenmodelle bestehen vorwiegend aus Schalen-Elemente mit linearem Ansatz, daher wird hier nur auf diese spezielle Form der Diskretisierung etwas näher eingegangen. In Kapitel 2.2 wird die benötigte Berechnung des durchschnittlichen Winkels zwischen den Normalvektoren entlang der gemeinsamen Kante zweier benachbarter linearer Schalen-Elemente erläutert. In Kapitel 2.3 wird ein Einblick in die Besonderheiten des ebenen Spannungszustandes und der Transformation in ein gedrehtes Koordinatensystem gegeben. Da es sich bei dieser Arbeit um eine programmtechnische Umsetzung in Abaqus mit der Python-Skripting-Schnittstelle handelt, werden die notwendigen Konventionen der FEM-Software in Kapitel 2.4 zusammengefasst.

2.1 Finite-Elemente-Methode

2.1.1 Allgemeines

Naturwissenschaftliche Vorgänge lassen sich durch Differentialgleichungen bzw. Systemen von Differentialgleichungen beschreiben, wobei sich komplexere Vorgänge in den meisten Fällen kaum mehr analytisch geschlossen lösen lassen. Man ist daher bestrebt Lösungen zumindest näherungsweise zu berechnen. Für diese Annäherungen an die analytisch exakte Lösung wurden, vorwiegend im vergangenen Jahrhundert, numerische Berechnungsverfahren entwickelt. Diese Verfahren gewannen durch die Entwicklung von Computern immer mehr an Bedeutung und sind mittlerweile in den unterschiedlichsten Disziplinen der Physik und des Ingenieurwesens kaum mehr wegzudenken.

Das wohl bekannteste und wichtigste numerisches Berechnungsverfahren ist die FEM. Ursprünglich wurde diese Methode dazu entwickelt das mechanische Verhalten diskreter Systeme wie allgemeiner Tragwerke, Fachwerke, Flugkörper, Mehrkörpersysteme etc., zu beschreiben. Diese Systeme sind durch eine endliche Anzahl an Freiheitsgraden gekennzeichnet. Etwas später wurde das Anwendungsgebiet auf die Kontinuumsmachanik ausgedehnt. In der Kontinuumsmechanik werden Feldgrößen, also im Speziellen in der Festkörpermechanik (Elasto-, Plastomechanik) Verschiebungen, Verzerrungen und Spannungen in jedem Punkt des Kontinuums, gesucht. Solche kontinuierlichen Systeme besitzen im Unterschied zu diskreten Systemen eine unendliche Anzahl an Freiheitsgraden [5].

Ein Grundgedanke der Finite-Elemente-Methode ist das Unterteilen eines gegebenen Gebietes in eine endliche Anzahl an Teilgebieten, also das Überführen eines kontinuierlichen Systems in ein Diskretes. Dieser Vorgang wird im Allgemeinen als Diskretisierung bzw. Vernetzung bezeichnet. Je nach Art der Problemstellung finden bei der Diskretisierung unterschiedliche Elementformen ihre Anwendung. So kommen z.B. bei einer Modellgeometrie in 1D Linienelemente, in 2D Drei- und Viereckelemente und in 3D quader- und tetraederförmige Elemente zum Einsatz. Die einzelnen finiten Elemente sind dabei durch Knoten definiert. Abbildung 2.1 zeigt examplarisch für den Fall eines Gebietes in 2D die Diskretisierung mit Drei- und Viereckelementen.

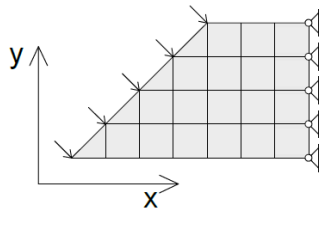

Abbildung 2.1: Geometrie in 2D diskretisiert mit Drei- und Viereckelementen

Des Weiteren wird bei den einzelnen Elementtypen zwischen linearen, quadratischen und Elementen höherer Ordnung unterschieden. Je höher die Ordnung, desto mehr Knoten sind im jeweiligen Element enthalten. Die Elementknoten sind erforderlich um die jeweiligen Ansatzfunktionen lokal an einem sogenannten Referenzelement zu definieren. Diese Ansatzfunktionen sind über Transformationsfunktionen mit dem Realelement verknüpft und sollen die gesuchten physikalische Größen, also z.B. die Verschiebungen, innerhalb des kontinuierlichen Elements approximieren. Es werden meist Polynome der jeweiligen Ordnung für die Ansatzfunktionen verwendet. Wird neben der physikalischen Größe auch die Geometrie des Modells mithilfe den selben Ansatzfunktion beschrieben, spricht man von isoparametrischen Elementen. Diese Elemente werden vor allem in der Analyse von Festkörpern eingesetzt. Eine exakte Erfassung der Modellgeometrie ist dabei aber nur in Ausnahmefällen möglich. Im Allgemeinen gilt, je feiner die Vernetzung und je höher der Ansatz, desto genauer die Beschreibung der Geometrie und die Annäherung an die gesuchte Lösung.

Mit dem Prinzip der virtuellen Arbeiten können dann die einzelnen Elementsteifigkeitsmatrizen $\mathbf{K^e}$ berechnet werden. Es ist dabei ein Gleichgewicht der Summen aus inneren und äußeren virtuellen Arbeiten gefordert. Gleichung (2.1) zeigt das Gleichungssystem für ein Element. Der Vektor $\mathbf{u^e}$ beinhalten die einzelnen Knotenfreiwerte und der Vektor $\mathbf{f^e}$ die auf das Element aufgebrachten Lasten.

$$\mathbf{K^e} \cdot \mathbf{u^e} = \mathbf{f^e} \tag{2.1}$$

Ein weiterer wichtiger Schritt ist die Assemblierung zu einem globalen Gleichungssystems. Unter Berücksichtigung der Lage der einzelnen Elemente, werden die Elementgleichungssysteme zu einem Gesamtgleichungssystem zusammengesetzt. Der Einbau der Verschiebungsrandbedingungen erfolgt durch das Streichen von Zeilen und den dazugehörigen Spalten in der Systemmatrix. Es folgt das zu lösenden Gesamtgleichungssystem

$$\mathbf{K} \cdot \mathbf{u} = \mathbf{f}. \tag{2.2}$$

In Gleichung (2.2) repräsentiert $\mathbf{K}$ die Gesamtsteifigkeitsmatrix, $\mathbf{u}$ die Knotenfreiwerte, welche es zu ermitteln gilt und $\mathbf{f}$ den Gesamtlastvektor dar. Beim Lösen dieses algebraischen Gleichungssystems kommen je nach Größe direkte oder iterative Verfahren zum Einsatz. In einer Nachlaufrechnung können dann die Verzerrungen und Spannungen aus den Knotenverschiebungen bzw. Knotenverdrehungen und dem zugrundeliegenden Materialgesetz berechnet werden.

2.1.2 Schalen-Elemente

Bei Schalen-Elementen handelt es sich um flächenförmige Elemente, welche im dreidimensionalen Raum angeordnet werden. Es kommt in der Modellbildung zur Vernachlässigung der Dicke des Bauteils. Diese Vereinfachung ist möglich, wenn es sich um dünnwandige Bauteile handelt, also deren Dicke im Verhältnis zu dessen Seitenlängen sehr klein ist. Somit wird meist nur die Mittelfläche, eventuell aber auch entweder die Unterseite oder Oberseite des jeweiligen Bauteils abgebildet.

Bei dünnwandigen Bauteilen sind grundsätzlich folgende Begriffe zu unterscheiden:

- **Plattenbauteile**: Plattenbauteile sind in deren Ausdehnung eben und sie sind dazu geeignet, Lasten quer zu dieser Ebene aufzunehmen. Zur Berechnung von Plattenbauteilen eignen sich Schalen-Elemente mit vernachlässigter Scheibenwirkung. Ein Knoten besitzt demnach 3 Freiheitsgrade davon einen Verschiebungsfreiheitsgrad normal zur Plattenebene und die 2 Rotationsfreiheitsgrade.

- **Scheibenbauteile**: Scheibenbauteile sind eben und dazu geeignet Lasten in Richtung der Ebene aufzunehmen. Ein Knoten eines Elements besitzt 2 Verschiebungsfreiheitsgrade, jeweils in x- und y-Richtung der Ebene.

- **Schalenbauteile**: Es handelt sich hier um dünnwandige Bauteile, welche sowohl quer als auch entlang dessen Ausdehnungsebene belastet werden können. Das Verhalten setzt sich daher aus Scheiben- und Plattentragwirkung zusammen. In der FEM-Simulation werden hierfür spezielle Schalen-Elemente verwendet. Diese Elemente sind für sich eben, wobei benachbarte Elemente im Raum anders ausgerichtet sein können. Ein Knoten besitzt 3 Verschiebungsfreiheitsgrade und 3 Rotationsfreiheitsgrade.
- **Membrane**: Bei Membrane handelt es sich um dünnwandige Bauteile, welche nur in dessen Ebene belastet werden können. Durch die Ausbildung einer „Kettenlinie“ kommt es zu einer räumlichen Tragwirkung. Im Zuge einer FEM-Simulation kommen spezielle Schalen-Elemente zum Einsatz [6].

Für die Diskretisierung der einzelnen flächenförmigen Bauteile wird auf Drei- bzw. Viereckelemente zurückgegriffen. In weiterer Folge wird auf Elementgeometrien mit linearen Ansatzfunktionen dieser Elementgeometrien näher eingegangen.

Ansatzfunktionen

Wie bereits erwähnt dienen die Ansatzfunktionen zur Approximation der Verschiebung, sowie zur Approximation der Modellgeometrie im jeweiligen Element. Da über jedes Element numerisch integriert werden muss, sind diese Ansatzfunktionen lokal für ein Referenzelement definiert und über spezielle Transformationsformeln mit dem eigentlichen Realelement im Modell verknüpft. Abbildung 2.2 und Abbildung 2.3 zeigen die Überführung der im Referenz-Koordinatensystems (ξ, η) definierten liniearen Dreieck- und bilinearen Viereckelemente in das globale Koordinatensystem (x, y, z). Es handelt sich dabei um eine Abbildung des zweidimensionalen Raums $\mathbb{R}^2$ in den dreidimensionalen Raum $\mathbb{R}^3$.

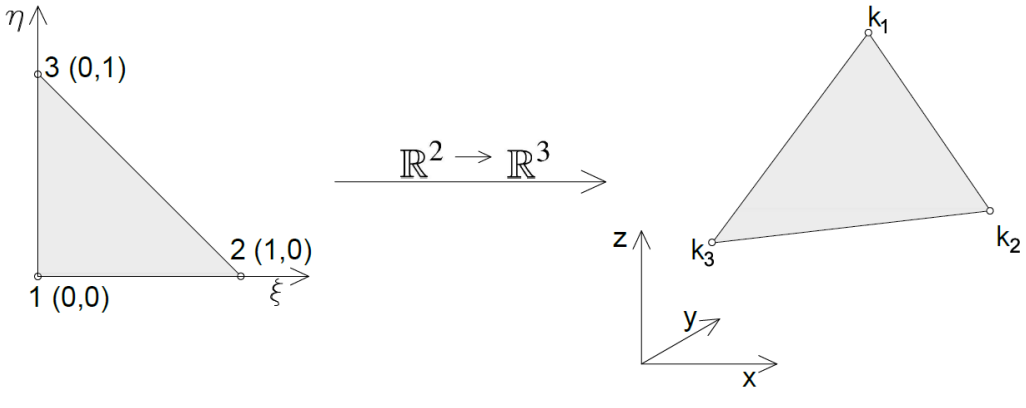

Abbildung 2.2: Referenzelement in $\mathbb{R}^2$ mit der Abbildung zum realen Element in $\mathbb{R}^3$

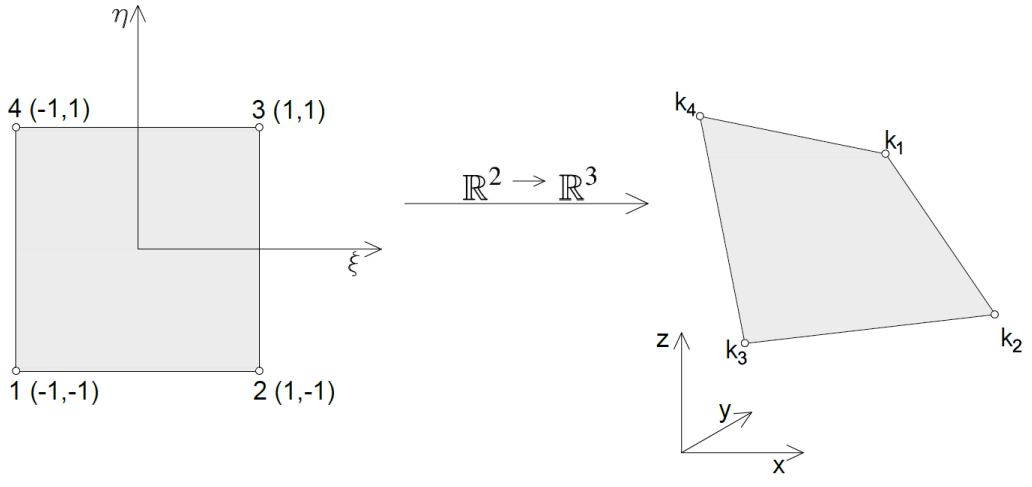

Abbildung 2.3: Referenzelement in $\mathbb{R}^2$ mit der Abbildung zum realen Element in $\mathbb{R}^3$

Für jeden Knoten des Referenzelements wird eine Ansatzfunktion definiert, welche an dem jeweiligen Knotenpunkt den Wert 1 und an den übrigen Knoten den Wert 0 annimmt. Die Ansatzfunktion eines linearen Dreieckelements bilden sich wie folgt:

$$\begin{aligned} N_1 &= 1 - \xi - \eta \\ N_2 &= \xi \\ N_3 &= \eta \end{aligned} \tag{2.3}$$

Die Abbildungen 2.4 bis 2.6 veranschaulichen die einzelnen Ansatzfunktionen für das lineare Dreieckelement.

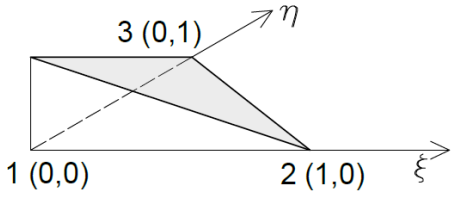

Abbildung 2.4: Ansatzfunktion N_1

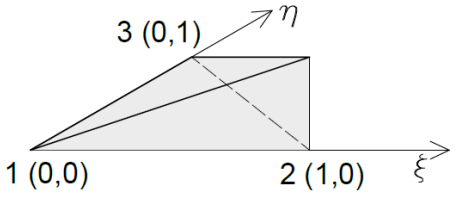

Abbildung 2.5: Ansatzfunktion N_2

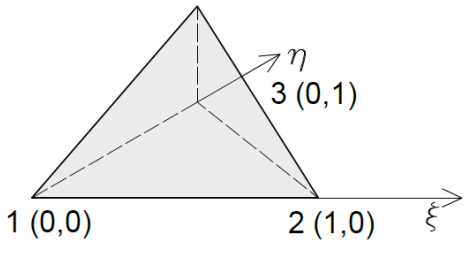

Abbildung 2.6: Ansatzfunktionen N_3

Die Ansatzfunktionen eines bilinearen Viereckelements sind

$$
\begin{aligned}
N_1 &= \frac{1}{4}(1-\xi)(1-\eta) \\
N_2 &= \frac{1}{4}(1+\xi)(1-\eta) \\
N_3 &= \frac{1}{4}(1+\xi)(1+\eta) \\
N_4 &= \frac{1}{4}(1-\xi)(1+\eta).
\end{aligned}
\tag{2.4}
$$

Die Abbildung 2.7 bis 2.10 zeigen die einzelnen Ansatzfunktionen für das bilineare Viereckelement.

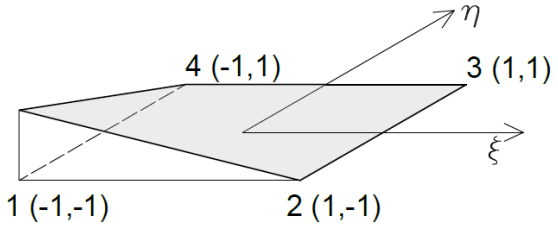

Abbildung 2.7: Ansatzfunktion N_1

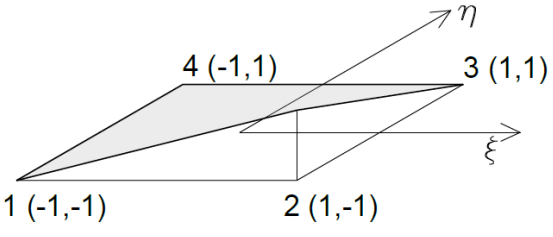

Abbildung 2.8: Ansatzfunktion N_2

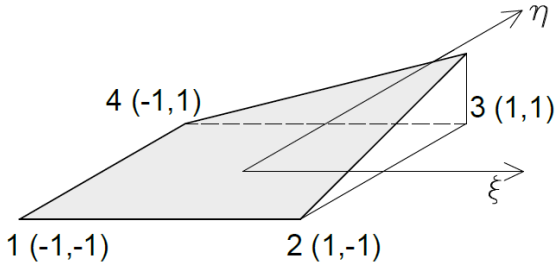

Abbildung 2.9: Ansatzfunktion N_3

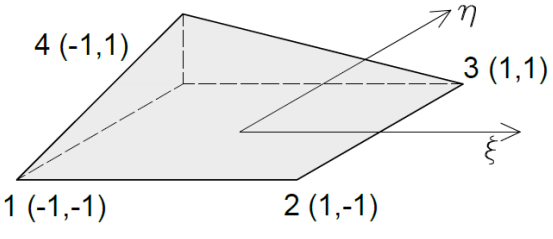

Abbildung 2.10: Ansatzfunktion N_4

2.2 Winkelberechnung

In diesem Abschnitt wird zunächst das Berechnen des Normalvektors in einem Punkt eines Schalen-Elements und die numerische Integration mittels Gauß-Quadratur erläutert. Dies sind die Grundlagen für die eigentliche Winkelberechnung, auf welche im Anschluss eingegangen wird. Die Winkelberechnung der Normalvektoren benachbarter Schalen-Elemente stellt die Grundlage für die Kantendetektion dar.

2.2.1 Berechnung des Normalvektors in einem Punkt eines Schalenelements

Den Zusammenhang zwischen den lokalen Koordinaten (ξ, η) des Referenzelements und den globalen Koordinaten (x,y,z) eines Realelements mit n Knoten erfolgt in Form der Gleichung (2.5). Diese Gleichung stellt die Koordinatentransformation dar.

$$
\mathbf{x}_{(\xi,\eta)} = \sum_{i=1}^{n} N_{i(\xi,\eta)} \cdot \mathbf{x}_i \tag{2.5}
$$

Für die Berechnung des Normalvektors ist es notwendig die Jacobi-Matrix zu ermitteln. Die Jacobi-Matrix enthält die Ableitungen der globalen Koordinaten aus Gleichung (2.5) nach die lokalen Koordinaten (ξ, η) und setzt sich wie folgt zusammen:

$$\mathbf{J} = \frac{\partial_{x_\mathrm{i}}}{\partial_{\xi_\mathrm{i}}} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{bmatrix} \tag{2.6}$$

Die beiden Spalten der Jacobi-Matrix stellen die Tangentenvektoren $\mathbf{t_1}$ und $\mathbf{t_2}$ am Realelement in Richtung der lokal definierten Koordinatenachsen ξ und η dar. Mit dem Kreuzprodukt der beiden Vektoren erhält man den normierten Normalvektor in einem Punkt eines Schalen-Elements.

$$||\mathbf{n}|| = ||\mathbf{t_1} \times \mathbf{t_2}|| = || \begin{bmatrix} \frac{\partial x}{\partial \xi} \\ \frac{\partial y}{\partial \xi} \\ \frac{\partial z}{\partial \xi} \end{bmatrix} \times \begin{bmatrix} \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \eta} \end{bmatrix} || \tag{2.7}$$

Beispiel 1: Berechnung des Normalvektors am linearen Dreieckelement

Ausgehend von den Gleichungen (2.5) bis (2.7) soll die Berechnung des Normalvektors eines Dreieckelements, wie in Abbildung 2.11 illustriert, gezeigt werden.

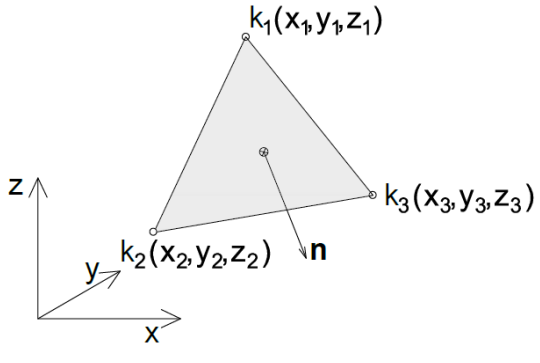

Abbildung 2.11: Lineares Dreieckelement in $\mathbb{R}^3$

Zunächst werden die einzelnen Formfunktionen des linearen Dreieckelements und die globalen Koordinaten der zugörigen Knoten in die Transformationsgleichung (2.5) eingesetzt.

$$\begin{aligned} x_{(\xi,\eta)} &= (1-\xi-\eta)x_1 + \xi x_2 + \eta x_2 \\ y_{(\xi,\eta)} &= (1-\xi-\eta)y_1 + \xi y_2 + \eta y_2 \\ z_{(\xi,\eta)} &= (1-\xi-\eta)z_1 + \xi z_2 + \eta z_2 \end{aligned} \tag{2.8}$$

Der nächste Schritt ist das Berechnen der Tangentenvektoren

$$\mathbf{t_1} = \begin{bmatrix} \frac{\partial x}{\partial \xi} \\ \frac{\partial y}{\partial \xi} \\ \frac{\partial z}{\partial \xi} \end{bmatrix} = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix} \tag{2.9}$$

und

$$\mathbf{t_2} = \begin{bmatrix} \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \eta} \end{bmatrix} = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{bmatrix}. \tag{2.10}$$

Abschließend wird aus den beiden Tangentenvektoren der Normalvektor

$$||\mathbf{n}|| = ||\mathbf{t_1} \times \mathbf{t_2}|| = || \begin{bmatrix} (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1) \\ (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_1) \\ (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) \end{bmatrix} || \tag{2.11}$$

berechnet.

Da der Normalvektor aus Gleichung (2.11) ausschließlich von den globalen Koordinaten der Elementknoten abhängt, ist dieser am linearen Dreieckelement stets konstant.

Beispiel 2: Berechnung des Normalvektors am Rand eines bilinearen Viereckelements

Anhand dieses Beispiels soll die Berechnung des Normalvektors an einem bestimmten Punkt am Rand eines bilinearen Viereckelements, welches in Abbildung 2.12 abgebildet ist, gezeigt werden.

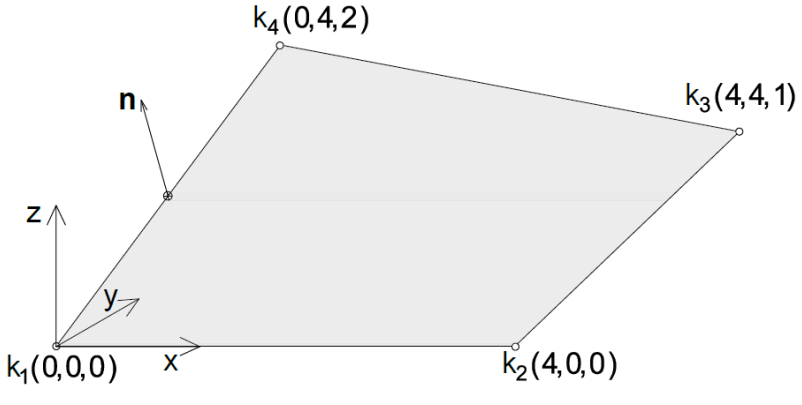

Abbildung 2.12: Bilineares Viereckelement in $\mathbb{R}^3$

Knoten Nr.	k_1	k_2	k_3	k_4
x	0	4	4	0
y	0	0	4	4
z	0	0	1	2

Tabelle 2.1: Knotenkoordinaten

Die Kante an welcher der Auswertungspunkt für den Normalvektor liegt, sei durch die Elementknoten k_1 und k_4 definiert. Aus den Element-Konnektivitäten [k_1,k_2,k_3,k_4] geht hervor, dass es sich dabei um die Kante mit den lokalen Knotennummern 1 und 4 am Referenzelement handelt für welche $\xi = -1$ und $-1 \leq \eta \leq +1$ gilt.

Im nachfolgenden Abschnitt wird das Berechnen des durchschnittlichen Winkels zweier Schalen-Elemente entlang der gemeinsamen Kante mithilfe der Gauß-Quadratur behandelt. Hierzu ist es notwendig die Normalvektoren an den Stützstellen im normierten Intervall $[-1, +1]$ entlang entsprechenden Kante zu berechnen. Im Zuge dieses Beispiels soll vereinfacht der Normalvektor in der Mitte der entsprechenden Kante im Punkt P_1 mit den globalen Koordinaten $(0.0, 2.0, 1.0)$ ermittelt werden. Somit sind die lokalen Koordinaten mit $\xi = -1$ und $\eta = 0$ festgelegt.

Zunächst werden die Formfunktionen für das bilineare Viereckelement und die Koordinaten der Elementknoten in die Transformationsgleichungen (2.5) eingesetzt.

$$\begin{aligned} x_{(\xi,\eta)} &= \frac{1}{4}[(1-\xi)(1-\eta)x_1 + (1+\xi)(1-\eta)x_2 + (1+\xi)(1+\eta)x_3 + (1-\xi)(1+\eta)x_4 \\ y_{(\xi,\eta)} &= \frac{1}{4}[(1-\xi)(1-\eta)y_1 + (1+\xi)(1-\eta)y_2 + (1+\xi)(1+\eta)y_3 + (1-\xi)(1+\eta)y_4 \\ z_{(\xi,\eta)} &= \frac{1}{4}[(1-\xi)(1-\eta)z_1 + (1+\xi)(1-\eta)z_2 + (1+\xi)(1+\eta)z_3 + (1-\xi)(1+\eta)z_4 \end{aligned} \tag{2.12}$$

Die beiden Tangentenvektoren ergeben sich zu

$$\mathbf{t_1} = \begin{bmatrix} \frac{\partial x}{\partial \xi} \\ \frac{\partial y}{\partial \xi} \\ \frac{\partial z}{\partial \xi} \end{bmatrix} = \begin{bmatrix} \frac{1}{4}[(\eta-1)x_1 + (1-\eta)x_2 + (\eta+1)x_3 - (\eta+1)x_4] \\ \frac{1}{4}[(\eta-1)y_1 + (1-\eta)y_2 + (\eta+1)y_3 - (\eta+1)y_4] \\ \frac{1}{4}[(\eta-1)z_1 + (1-\eta)z_2 + (\eta+1)z_3 - (\eta+1)z_4] \end{bmatrix} \tag{2.13}$$

und

$$\mathbf{t_2} = \begin{bmatrix} \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{1}{4}[(\xi-1)x_1 - (\xi+1)x_2 + (\xi+1)x_3 + (1-\xi)x_4] \\ \frac{1}{4}[(\xi-1)y_1 - (\xi+1)y_2 + (\xi+1)y_3 + (1-\xi)y_4] \\ \frac{1}{4}[(\xi-1)z_1 - (\xi+1)z_2 + (\xi+1)z_3 + (1-\xi)z_4] \end{bmatrix}. \tag{2.14}$$

Durch Einsetzen der globalen Koordinatenwerte für die jeweiligen Knoten aus Tabelle 2.1 und der lokalen Koordinaten $\xi = -1$ und $\eta = 0$ folgt:

$$\mathbf{t_1} = \begin{bmatrix} 2 \\ 0 \\ -\frac{1}{4} \end{bmatrix} \qquad \mathbf{t_2} = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix} \tag{2.15}$$

Bildet man das Kreuzprodukt der beiden Tangentenvektoren und normiert es anschließend erhält man den Normalvektor im Punkt P_1.

$$||\mathbf{n}|| = ||\mathbf{t_1} \times \mathbf{t_2}|| = \frac{1}{\sqrt{\frac{101}{4}}} \begin{bmatrix} \frac{1}{2} \\ -3 \\ 4 \end{bmatrix} \tag{2.16}$$

2.2.2 Integration mit Gauß-Quadratur

Um den durchschnittlichen Winkel zwischen den Normalvektoren zweier benachbarter Schalen-Elemente entlang der gemeinsamen Kante zu berechnen, wird im Zuge dieser Arbeit auf die Integration mit der Gauß-Quadratur zurückgegriffen. In diesem Abschnitt wird zunächst allgemein auf diese numerische Integrationsmethode eingegangen.

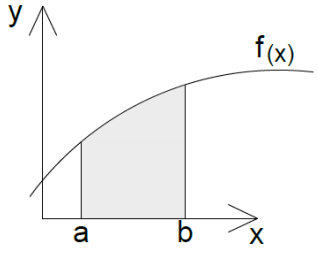

Abbildung 2.13: Funktion $f(x)$

Es soll die Funktion $f(x)$ im globalen Intervall $x \in [a, b]$ integriert werden.

$$I = \int_a^b f(x)dx \tag{2.17}$$

Bei der Gauß-Quadratur wird das gegebene Intervall $x \in [a, b]$ in das normierte Intervall $t \in [-1, +1]$, wie in Gleichungen (2.18) gezeigt, transformiert.

$$x = \frac{b-a}{2}t + \frac{a+b}{2} \tag{2.18}$$

Setzt man nun die Koordinatentransformation von Gleichung (2.18) in Gleichung (2.17) ein folgt:

$$I = \frac{b-a}{2} \int_{-1}^{1} f(\frac{b-a}{2}t + \frac{a+b}{2})dt \tag{2.19}$$

Bei numerischen Integrationsverfahren soll nur noch eine endliche Anzahl an Funktionswerten an bestimmten Stützstellen benötigt werden. Es kommt zum Übergang der analytischen Darstellung aus Gleichung (2.17) in die diskrete Darstellung der Gleichung (2.20).

$$I \approx I_n = \frac{b-a}{2} \sum_{i=1}^{n} w_i \cdot f(\frac{b-a}{2}t_i + \frac{a+b}{2}) = \frac{b-a}{2} \sum_{i=1}^{n} w_i \cdot f(x_i) \tag{2.20}$$

mit

$$x_i = \frac{b-a}{2}t_i + \frac{a+b}{2} \tag{2.21}$$

In Gleichung (2.20) sind $f(x_\mathrm{i})$ die Funktionswerte an den Stützstellen im globalen Intervall $x_\mathrm{i} \in [a, b]$ und w_i die dazugehörigen Gewichte. Es können mit n Stützstellen Polynome der Ordnung $m = 2n - 1$ exakt integriert werden. In Tabelle 2.2 sind die Stützstellen und dessen zugehörigen Gewichte exemplarisch für die Gauß-Ordnungen eins bis drei angeführt.

Ordnung n	$\mathbf{x_i}$	$\mathbf{w_i}$
1	0	2
2	$-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$	1, 1
3	$-\sqrt{\frac{3}{5}}, 0, \sqrt{\frac{3}{5}}$	$\frac{5}{9}, \frac{8}{9}, \frac{5}{9}$

Tabelle 2.2: Gauß-Punkte und Gewichte

2.2.3 Berechnung des gemittelten Winkels zweier benachbarter Schalen-Elemente entlang der gemeinsamen Kante

In diesem Abschnitt soll die Berechnung des gemittelten Winkels zweier benachbarter Schalen-Elemente entlang der gemeinsamen Kante erläutert werden. Zu diesem Zweck illustriert Abbildung 2.14 zwei Viereckelemente deren gemeinsame Kante durch die Knoten k_1 und k_2 definiert ist.

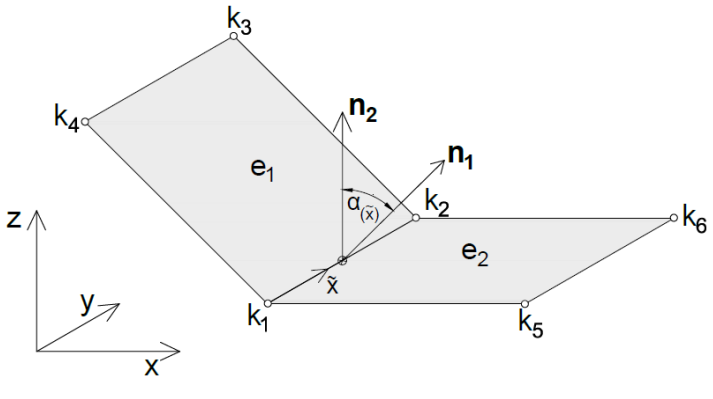

Abbildung 2.14: Zwei benachbarte Schalen-Elemente

Der mittlere Winkel der Normalvektoren der beiden Schalen-Elemente entlang der gemeinsamen Kante erechnet sich mit

$$\alpha_m = \frac{1}{\int_{k_1}^{k_2} d\tilde{x}} \int_{k_1}^{k_2} \alpha(\tilde{x}) d\tilde{x} \tag{2.22}$$

wobei

$$\alpha(\tilde{x}) = \arccos\left(\frac{n_1(\tilde{x}) \cdot n_2(\tilde{x})}{|n_1(\tilde{x})| \cdot |n_2(\tilde{x})|}\right). \tag{2.23}$$

Wendet man zur Berechnung von 2.22 die Gauß-Quadratur an müssen zunächst die Normalvektoren $\mathbf{n_i}$ beider Elemente an den definierten Stützstellen, wie im Abschnitt 2.2.1 gezeigt, ermittelt werden. Dabei ist zu beachten auf welcher Kante des Referenzelements sich die jeweiligen Stützstellen des entsprechenden Realelements befinden um die korrekten Werte der lokalen Koordinaten zu erhalten. Abbildung 2.15 zeigt exemplarisch für das Normalenvektorpaar in der Mitte der gemeinsamen Kante die jeweilige Lage am Referenzelement.

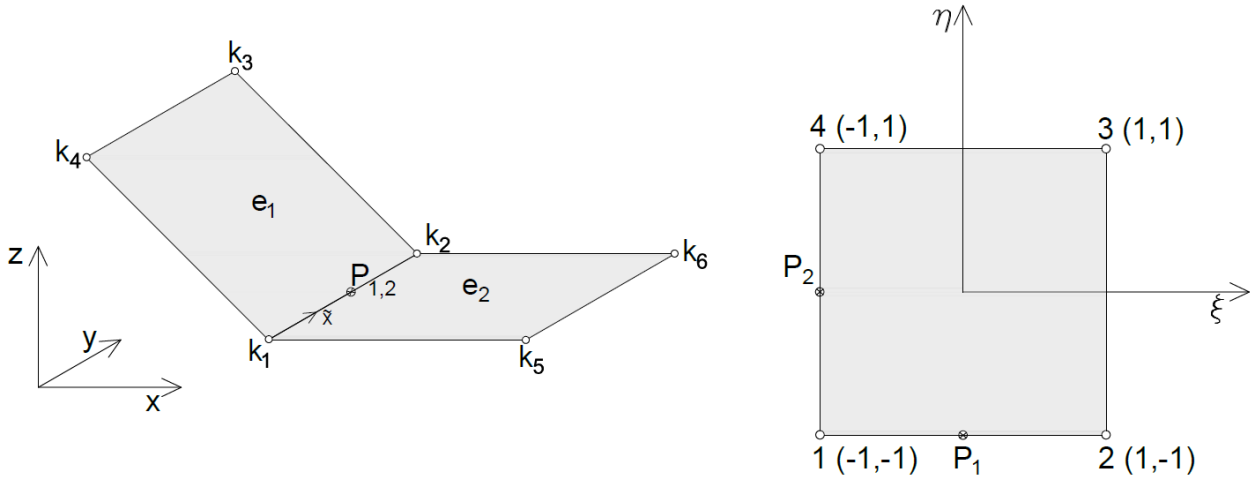

Abbildung 2.15: Zwei Schalen-Elemente in $\mathbb{R}^3$ und die jeweiligen Stützpunkten am Referenzelement

Wurden die Normalvektoren an den Stützstellen ermittelt, kann der Cosinus der Winkel α_i zwischen den Normalvektorpaaren mit

$$\cos\alpha_i = \frac{n_{i1} \cdot n_{i2}}{|n_{i1}| \cdot |n_{i2}|} \tag{2.24}$$

berechnet werden. Anschließend lassen sich die einzelnen Winkel α_i zwischen den Normalvektorpaaren an den Stützstellen berechnen.

$$\alpha_i = \arccos(\cos\alpha_i) \tag{2.25}$$

Der durchschnittliche Winkel errechnet sich nun für n Gauß-Punkte wie folgt:

$$\alpha_m = \frac{\frac{k_2-k_1}{2}\sum_{i=1}^{n} w_i \cdot \alpha_i}{k_2 - k_1} = \frac{1}{2}\sum_{i=1}^{n} w_i \cdot \alpha_i \tag{2.26}$$

2.3 Spannungen und deren Tranformation bei Schalen-Elementen

Wurde in der FEM-Analyse der Vektor **u**, welcher die Knotenfreiwerte (Verschiebungen und Verdrehungen) beinhaltet, berechnet, erfolgt in einer Nachlaufrechnung die Ermittlung des Verzerrungsvektors $\boldsymbol{\epsilon}$ mit der Beziehung

$$\boldsymbol{\epsilon} = \mathbf{B} \cdot \mathbf{u}\,. \tag{2.27}$$

Die Matrix **B** stellt dabei den diskreten Differentialoperator dar [7]. Anschließend folgt die Berechnung der Spannungen

$$\boldsymbol{\sigma} = \mathbf{D} \cdot \boldsymbol{\epsilon} \tag{2.28}$$

unter Berücksichtigung der entsprechenden Materialparameter, welche in der Matrix **D** enthalten sind. Bei Verwendung von Elementen mit linearem Ansatz ist das Spannungsfeld, im Gegensatz zum Verschiebungsfeld nicht stetig. Da üblicherweise die Spannungskomponenten an den Integrationspunkten des Elements berechnet und zu den Knoten extrapoliert werden, entstehen an den Elementübergängen Spannungssprünge.

2.3.1 Ebener Spannungszustand

Wenn Spannungen nur in zwei Raumrichtungen auftreten spricht man von einem ebenen Spannungszustand. Dieser ebene Spannungszustand wird vereinfacht bei dünnwandigen Bauteilen, welche in der FEM-Simulation mit Platten-, Scheiben- und Schalen-Elementen diskretisiert werden, angenommen. Es handelt sich dabei um Bauteile in denen die Spannungen in Dickenrichtung vernachlässigt werden können. Ist das Element in der xy-Ebene definiert, fallen die Spannungen in z-Richtung weg und der räumliche Spannungstensor

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{bmatrix} \tag{2.29}$$

vereinfacht sich zum ebenen Spannungstensor

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \tau_{xy} \\ \tau_{yx} & \sigma_{yy} \end{bmatrix}. \tag{2.30}$$

Abbildung 2.16 veranschaulicht den ebenen Spannungszustand an einem Flächenelement. Dabei zeigen positive Spannungen am positiven Schnittufer in die positive Richtung der jeweiligen Koordinatenachse und am negativen Schnittufer in die negative Richtung der Koordinatenachse. Der erste Index bezeichnet die Koordinatenachse, welche auf die Schnittkante normal steht und der zweite Index die Koordinatenrichtung in welche die Spannung wirkt [8].

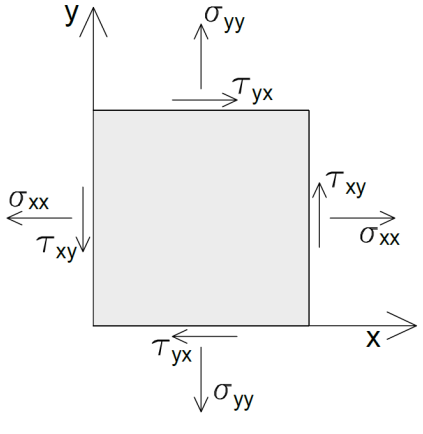

Abbildung 2.16: Flächenelement mit ebenen Spannungszustand

Wegen der Dualität der Schubspannungen $\tau_{\mathrm{xy}} = \tau_{\mathrm{yx}}$ ist es möglich den ebenen Spannungszustand mit den beiden Hauptspannungskomponenten σ_{xx} und σ_{yy} und der zugehörigen Schubspannung τ_{xy} vollständig zu beschreiben. Die Komponenten werden dann oft in einem Pseudovektor

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{bmatrix} \tag{2.31}$$

zusammengefasst.

2.3.2 Spannungstransformation

Der Vektor in Gleichung (2.31) beschreibt den ebenen Spannungszustand in einem Punkt für ein vorgegebenes kartesisches Koordinatensystem. Die Schnittkanten sind dabei parallel zu den Koordinatenachsen. Ist es erforderlich die Spannungskomponenten bezüglich eines gedrehten Koordinatensystems auszudrücken, benötigt man eine Transformation. Abbildung 2.17 zeigt eine Transformation vom xy-Koordinatensystem in das um den Winkel ϕ gedrehte $\xi\eta$-Koordinatensystem in der selben Ebene.

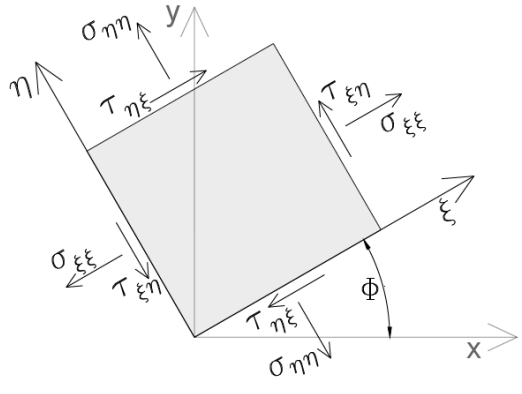

Abbildung 2.17: Flächenelement im gedrehten Koordinatensystem

Der Drehwinkel ϕ ist positiv wenn bezüglich des durch die Achsen x, y definierten Rechtssystems eine Drehung in die positve Drehrichtung erfolgt. Die Transformationsformeln für die Berechnung der Normalspannungen $\sigma_{\xi\xi}$ und $\sigma_{\eta\eta}$ und der dazugehörigen Schubspannung $\tau_{\xi\eta}$ sind

$$\begin{aligned} \sigma_{\xi\xi} &= \frac{1}{2}(\sigma_{xx} + \sigma_{yy}) + \frac{1}{2}(\sigma_{xx} - \sigma_{yy})\cos 2\phi + \tau_{xy}\sin 2\phi \\ \sigma_{\eta\eta} &= \frac{1}{2}(\sigma_{xx} + \sigma_{yy}) - \frac{1}{2}(\sigma_{xx} - \sigma_{yy})\cos 2\phi - \tau_{xy}\sin 2\phi \\ \tau_{\xi\eta} &= -\frac{1}{2}(\sigma_{xx} - \sigma_{yy})\sin 2\phi + \tau_{xy}\cos 2\phi. \end{aligned} \tag{2.32}$$

2.4 Das FEM-Softwarepaket Abaqus

Abaqus ist ein FEM-Berechnungsprogramm [2] und dient zum Lösen von linearen und nicht-linearen Anfangsrandwertaufgaben der Disziplinen Mechanik, Wärmeleitung, Strömungsmechanik und Elektormagnetismus. Das Softwarepaket besteht aus den Teilen Abaqus/CAE (Complete Abaqus Environment) für Pre- und Postprocessing und den Lösern:

- Abaqus/Standard: Gleichungslöser für lineare und nicht lineare statische und dynamische Probeleme
- Abaqus/Explizit: Gleichungslöser für z.B. Stoßprozesse oder komplexen Kontaktproblemen

- Abaqus/CFD: Gleichungslöser für inkompressible Strömungen

In Abaqus/CAE können Modelle erstellt oder importiert, Simulationen gestartet und Ergebnisse dargestellt und verarbeitet werden. Der Prä- und Postprozessor besitzt neben den Schnittstellen zu den Programmiersprachen Fortran und C++ auch eine Python-Skripting-Schnittstelle. So können unter anderem der Modellaufbau, oder Teile davon und die Ergebnisauswertung und Weiterverarbeitung automatisiert werden [9].

In den folgenden Abschnitten werden die Konventionen der Abaqus-Software, dessen Kenntnis für die Erstellung der einzelnen Python-Skript-Dateien notwendig sind, näher beschrieben.

2.4.1 Elementbibliothek

Die in Abaqus verfügbaren Elemente lassen sich anhand der 5 Aspekte

- Familie
- Knotenfreiwerte
- Anzahl der Knoten
- Formulierung
- Integration

charakterisieren [10]. Dem Namen eines Elements sind die einzelnen Charakteristiken zu entnehmen. Auf die, für diese Arbeit benötigten Elementtypen wird nun näher eingegangen.

Es sind dies die Schalen-Elemente S3/S3R und S4/S4R. Der erste Buchstabe beschreibt die Element-Familie. In diesem Fall also „S“ für „Shell“. Die Ziffern 3 bzw. 4 geben die Anzahl der Elementknoten und somit auch die verwendete Formulierung der Ansatzfunktionen wieder. So handelt es sich bei S3-Elementen um Dreieckelemente mit linearem und bei S4-Elementen um Viereckelemente mit bilinearem Ansatz. Der Zusatzbuchstabe „R“ gibt Auskunft über die Integration des jeweiligen Elements. Wird dieser angeführt wird dieses Element reduziert integriert, d.h. es besitzt in der Element-Ebene nur einen Integrationspunkt. Sonst wird das jeweilige Element mit ausreichender Quadratur integriert. Bei diesen beiden Elementklassen handelt es sich um Standard-Schalenelemente.

Die Richtung des Normalvektors am Flächenelement ist durch die Reihenfolge der Knoten bestimmt und folgt der „Korkenzieherregel“. Abbildung 2.18 veranschaulicht dies anhand eines Viereckelements.

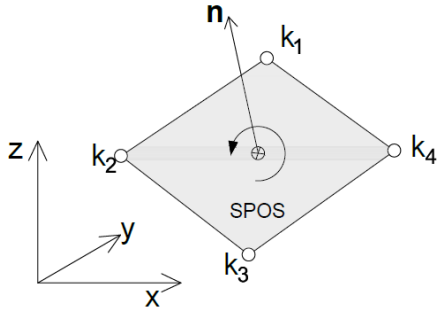

Abbildung 2.18: Normalendefinition eines Schalen-Elements

In Abaqus legt die Austrittsseite des Normalvektors die Oberseite des Schalen-Elements fest. Diese ist unter anderem für den Zugriff auf die entsprechenden Spannungskomponenten an der Unter- bzw. Oberseite des jeweiligen Schalen-Elements im Postprocessing von Bedeutung.

2.4.2 Zugriff auf einzelnen Objekte mithilfe eines Python-Skripts

Die Python-Schnittstelle bietet den Zugriff auf bestimmte Objekte der Abaqus-Datenbank. Dieser Abschnitt soll einen groben Überblick darüber geben.

Preprocessing

Abbildung 2.19 illustriert die Pfade zu den Objekten im Preprocessing, auf welche in den Skripts die meisten bzw. wichtigsten Zugriffe erfolgen [11].

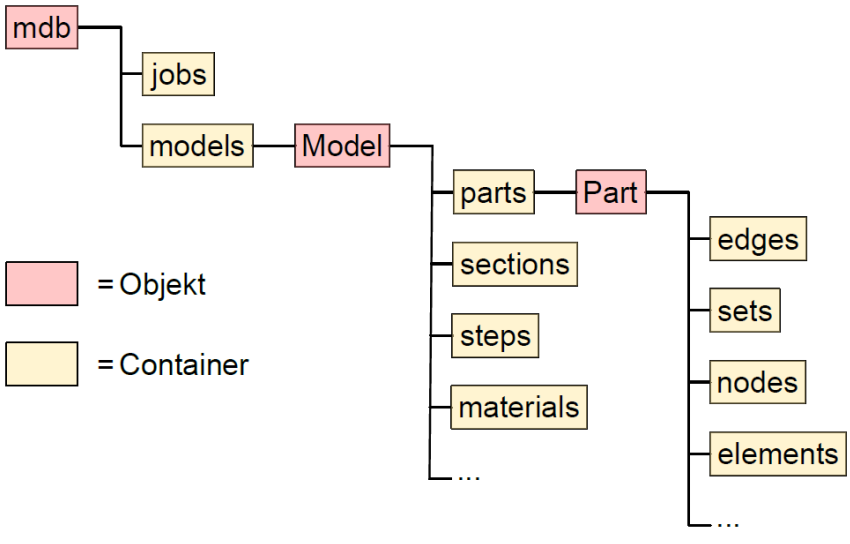

Abbildung 2.19: Pfade zu den Objekten im Präprozessing

Exemplarisch sieht der Zugriff auf die Konnektivitäten des ersten Elements im *„elements“* Container z.B. wie folgt aus:

$$connectivities1 = mdb.models['Model\text{-}Name'].parts['Part\text{-}Name'].elements[0].connectivities$$

Der Container *„models“* beinhalten sämtliche Modelle, die in der Abaqus/CAE Umgebung geöffnet sind. Ein Modell kann aus mehreren Part-Objekten bestehen, welche die einzelnen Bauteile des jeweiligen Modells darstellen. In einem Part-Objekt sind wieder mehrere Container enthalten, u.a. jener in dem sich sämtliche Element-Objekte befinden.

Postprocessing

Abbildung 2.20 zeigt den Pfad zu *„fieldOutput“* in der Output Database im Postprocessing [11]. *„fieldOutput“* beinhaltet die gesamte Ergebnissdatenbank für einen Frame eines bestimmten Steps. Ein Frame stellt einen Iterationsschritt eines Steps dar. Bei einem Step handelt es sich um einen Berechnungsschritt einer Analyse.

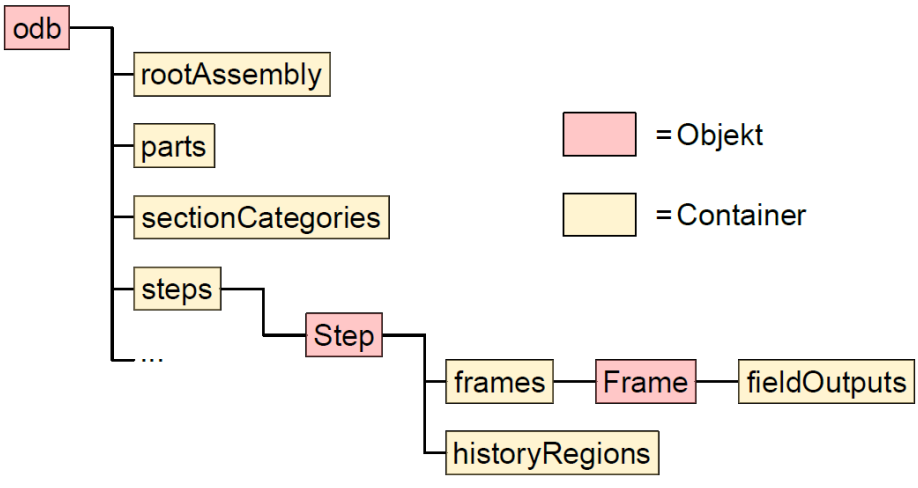

Abbildung 2.20: Pfade zu den Objekten im Präprozessing

Exemplarisch für den letzten Frame eines Steps funktioniert der Zugriff auf *fieldOutput* wie folgt:

$$fieldOutput = odb['ODB\text{-}Name'].steps['Step\text{-}Name'].frames[-1].fieldOutput$$

Die Ergebnisse sind an den Integrationspunkten sowie am Mittelpunkt und an den Knoten des Elements, jeweils an den einzelnen Section-Points, verfügbar. Diese Section-Points stellen die Integrationspunkte in Dickenrichtung des Schalen-Elements dar. Für die Integration in Dickenrichtung des Schalen-Elements ist in Abaqus die Simpson-Integration mit 5 Stützstellen als „default“ eingestellt. Im Zuge dieser Arbeit sind die Ergebnisse an den beiden äußersten Section-Points 1 und 5 von Bedeutung. Section Point 1 befindet sich an der Unterseite (negativen Seite), in Abaqus als SNEG bezeichnet und Section Point 5 an der Oberseite (positiven Seite), in Abaqus als SPOS bezeichnet.

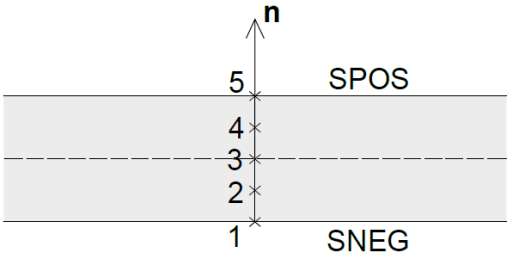

Abbildung 2.21: Section points (Simpson-Integration)

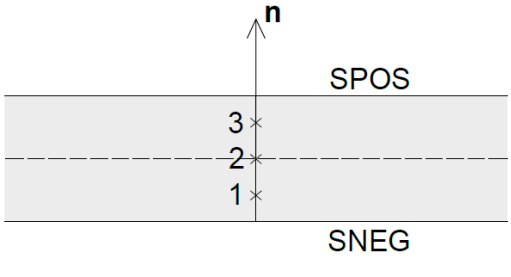

Abbildung 2.22: Section points (Gauß-Integration))

2.4.3 Richtung der Spannungskomponenten

Die Spannung S_{11} wirkt in Richtung der 1-Achse des lokalen Koordinatensystems des jeweiligen Schalen-Elements. Die Spannung S_{22} wirkt in Richtung der 2-Achse des lokalen Koordinatensystems. S_{12} ist die dazugehörige Schubspannung. Wegen der Dualität der Schubspannungen gilt $S_{12} = S_{21}$.

Bei der lokalen 1-Achse handelt es sich um die Projektion der globalen x-Achse auf das jeweilige Element wenn der Winkel zwischen x-Achse und der Flächennormalen größer als 1 [°] ist. Ist der Winkel kleiner als 1 [°] wird für die 1-Achse die Projektion der globalen z-Achse herangezogen. Die lokale 2-Achse steht normal auf die 1-Achse und deren Richtung ist so definiert, dass die beiden Achsen zusammen mit der Elementnormalen ein gültiges Rechtssystem bilden (siehe Abbildung 2.23).

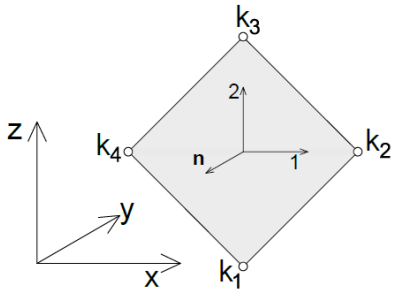

Abbildung 2.23: Definition des lokalen Koordinatensystems

3 Programmtechnische Umsetzung & Algorithmen

3.1 Generelle Aufbau

In weiterer Folge werden der Aufbau und die Abläufe der einzelnen Skript-Dateien welche zur Anwendung kommen näher beschrieben. Grundsätzlich besteht die gesamte programmtechnische Umsetzung aus den drei Hauptdateien *„WeldGeneratorMain.py“*, *„UpdateMain.py“* und *„StressTransformMain.py“*. Diese Dateien werden mit Fortlauf der Modellbearbeitung in der Abaqus-Umgebung ausgeführt. Die drei Dateien *„WeldGenerator.py“*, *„Update.py“* und *„StressTransform“* beinhalten den dazugehörigen Algorithmen für die eigentliche Bearbeitung des Modells. Methoden, die für den Ablauf des Programms benötigt werden, sind in eigenen Dateien mit den Namen *„methodsA.py“*, *„methodsB.py“* und *„methodsC.py“* zusammengefasst. In Abbildung 3.1 ist der generelle Ablauf der einzelnen Teile in einem Flussdiagramm aufskizziert.

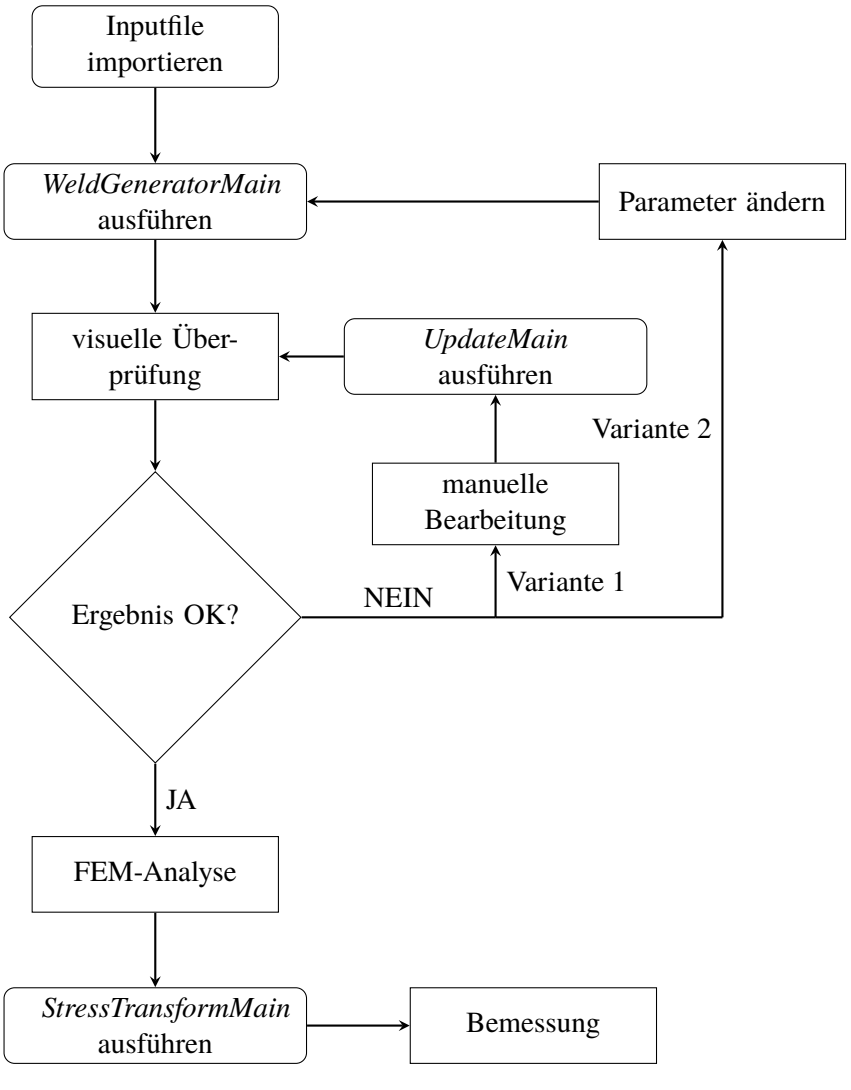

Abbildung 3.1: Generelle Ablauf der Analyse eines Modells

Zuerst wird ein geeignetes Input-File in die Abaqus-Umgebung importiert. Danach kann die Skript-Datei *„WeldGeneratorMain.py“* ausgeführt werden. Auf die nötigen Eingabeparameter wird im nachfolgenden Abschnitt genauer eingegangen. Wurde das Skript *„WeldGeneratorMain.py“* erfolgreich beendet und somit neue

Linienelemente an den entsprechenden Bauteilkanten generiert, wird eine Ausgabe-Datei mit der notwendigen Information für die Spannungstransformation im Postprozessing erstellt und in das Arbeitsverzeichnis von Abaqus gespeichert.

Im Anschluss erfolgt die visuelle Überprüfung durch den Anwender. Ist das Ergebnis der Scheißnahtgenerierung nicht zufriedenstellend, stehen zwei Möglichkeiten zur Verfügung um fortzufahren. Eine Variante ist die manuelle Nachbearbeitung mit anschließendem Ausführen der Skript-Datei *„UpdateMain.py“*. Die zweite Variante ist das Anpassen der Eingabeparameter und dem erneuten Ausführen von *„WeldGeneratorMain.py“*.

Ist das Ergebnis in Ordnung, kann die FEM-Analyse durchgefürt werden. Diese erfolgt ohne Berücksichtigung der zuvor generierten Linienelemente. Im Postprozessingbereich von Abaqus wird, nach fertiggestellter Simulation, das Skript *„StressTransformMain.py“* ausgefürt. Diese Skript-Datei sorgt nun für die Spannungstransformation in jenen Elementen, welche an einer zuvor detektierten Kante und somit an einer potentiellen Schweißnaht liegen. Die Transformierten Spannungen können dann in Contour-Plots dargestellt werden und dienen in weiterer Folge zur Bemessung der Schweißnahtdetails.

3.2 Teil A - „WeldGeneratorMain.py“ und „WeldGenerator.py“

Mit dem Ausführen der Skript-Datei *„WeldGeneratorMain.py“* erfolgt die Generierung neuer Linienelemente, welche zur Repräsentation von Schweißnähten dienen sollen. Als Grundlage dafür muss ein bereits vernetztes Plattenmodell in Abaqus geöffnet sein.

Es erscheint eine Dialog-Box, in der zwei Toleranzen abgefragt werden und vom Benutzer eingegeben werden sollten. Der erste Wert, in weiterer Folge mit $\Delta\alpha_1$ bezeichnet, bezieht sich auf den Winkel zwischen den Normalvektoren zweier benachbarter Schalen-Elemente. Ist der berechnete Winkel zwischen den Normalvektoren größer bzw. gleich $\Delta\alpha_1$, so wird der gemeinsame Rand der beiden benachbarter Elemente als Kante markiert. Der zweite Wert, im Folgenden mit $\Delta\alpha_2$ bezeichnet, ist eine Toleranz die festlegt, ab welchem Winkel zweier aufeinanderfolgender Elementkanten ein Eckknoten erstellt werden soll. Dieses Kriterium ist für die Einteilung und dem Zusammenfassen von einzelnen Linienelementen in Gruppen von Bedeutung.

Anschließend erfolgt eine Iteration über alle Schalen-Elementkanten des aktuell geöffneten Modells. Die Indizes der einzelnen Elemente, welche an einer gemeinsamen Kante liegen, werden, sofern es sich um Elemente mit linearem Ansatz handelt und die Anzahl der Elemente pro Kante größer 1 ist, in eine Liste gespeichert. Die Speicherung der Knotenindizes der gemeinsamen Elementkante erfolgt auf gleicher Weise in eine separate Liste. Kommen im Modell quadratische Elemente vor, wird eine Warnung angezeigt und das Skript beendet. Ansonsten wird die Skript-Datei *„WeldGenerator.py“* ausgeführt. Abbildung 3.2 veranschaulicht den Ablauf bis zum Ausfüren dieses Unterprogramms.

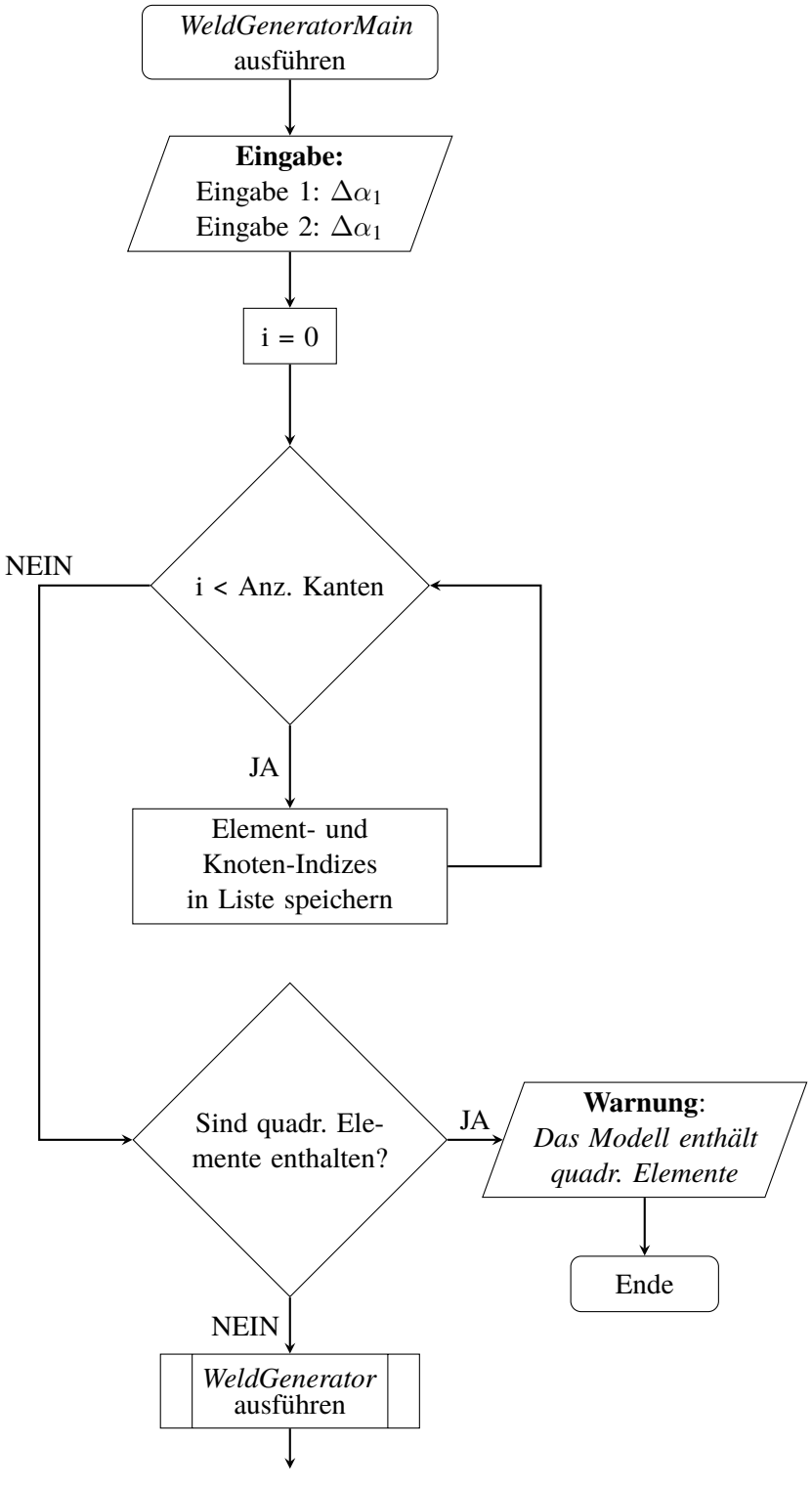

Abbildung 3.2: Ablauf von *„WeldGeneratorMain.py“*

Im Skript *„WeldGenerator.py“* erfolgt die eigentliche Kantendetektion. Es wird über die Liste, welche die Indizes der benachbarten Elemente beinhaltet, iteriert. Liegen genau 2 Elemente an einer gemeinsamen Kante, erfolgt die Berechnung des Winkels zwischen den Normalvektoren gemäß Gleichung 2.26 aus Kapitel 2.2.3. Ist der Wert des berechneten Winkels kleiner als der vom Benutzer eingegebenen Wert für $\Delta\alpha_1$, werden die Elemente und die gemeinsame Kante nicht mehr weiter berücksichtigt.

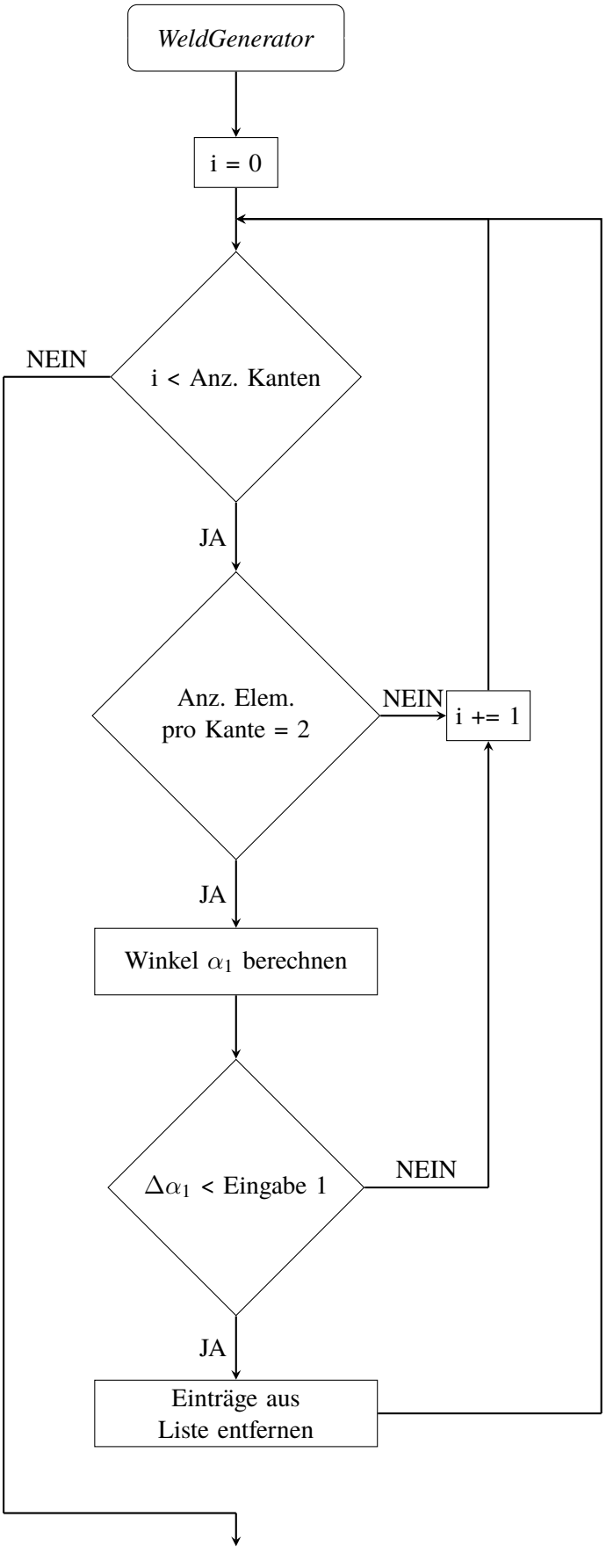

Abbildung 3.3: Ablauf der Kantendetektion

Wurde vom Benutzer die Möglichkeit gewählt Viereckelemente, welche an Kreuzungspunkten liegen, in zwei Dreieckelementen zu teilen wird dies nach Beendigung der Kantendetektion ausgeführt. Im Anschluss erfolgt die Sortierung der an einer Kante liegenden Knotenpaare. Zunächst werden die Start- und Kreuzungsknoten gesucht und ein passendes Knotenpaar an die erste Stelle der Liste verschoben. Danach werden die einzelnen Einträge der Liste so sortiert, dass im Modell benachbarte Knotenpaare sich auch in der Liste nebeneinander befinden. Ist kein unmittelbar benachbartes Knotenpaar mehr vorhanden, wird ein Knotenpaar, welches entweder einen Start- oder Kreuzungsknoten enthält, an die nächste Stelle verschoben und von hier aus weiter sortiert. Der Ablauf ist in Abbildung 3.4 zusammengefasst.

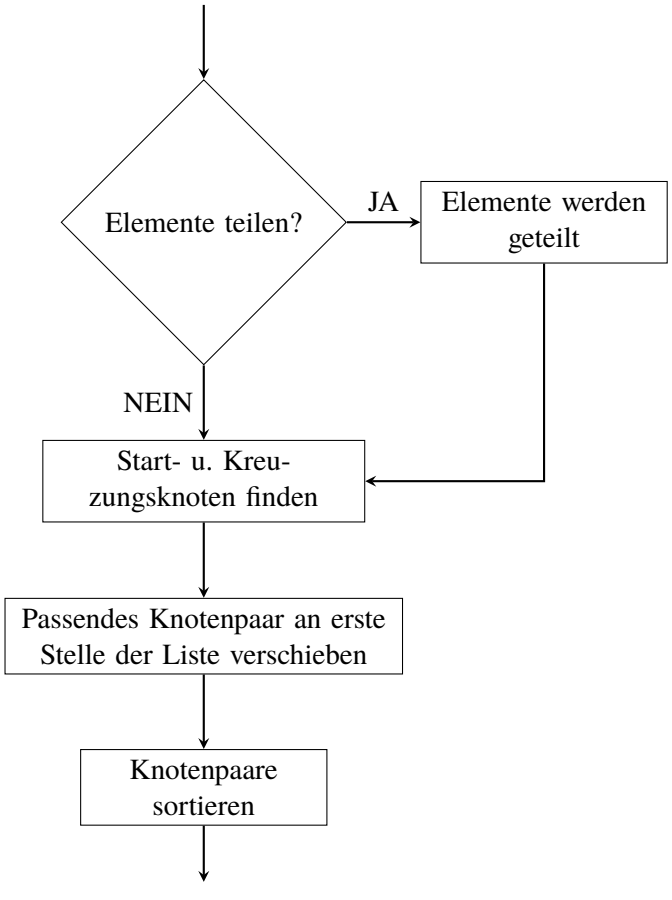

Abbildung 3.4: Ablauf der Sortierung der Knotenpaare

Danach kommt es zur Eckknotendetektion. Hier wird über die Knotenpaare, welche sich an einer Kante befinden iteriert. Das Kriterium für die Detektion eines Eckknotens ist der Wert der vom Benutzer eingegebenen Toleranz $\Delta\alpha_2$. Ist der Wert des Winkels α_2 zwischen den Richtungungsvektoren zweier aufeinanderfolgender Knotenpaare größer als für $\Delta\alpha_2$ handelt es sich beim gemeinsamen Knoten der beiden Knotenpaare um einen Eckknoten. Anschließend erfolgt die Einteilung der Knotenpaare in Gruppen, sodass es sich beim ersten und letzten Knoten jeder Gruppe entweder um einen Start-, Kreuzungs- oder Eckknoten handelt. Diese Gruppeneinteilung ist die Grundlage für die Generierung der Linienelemente, welche im Anschluss erfolgt.

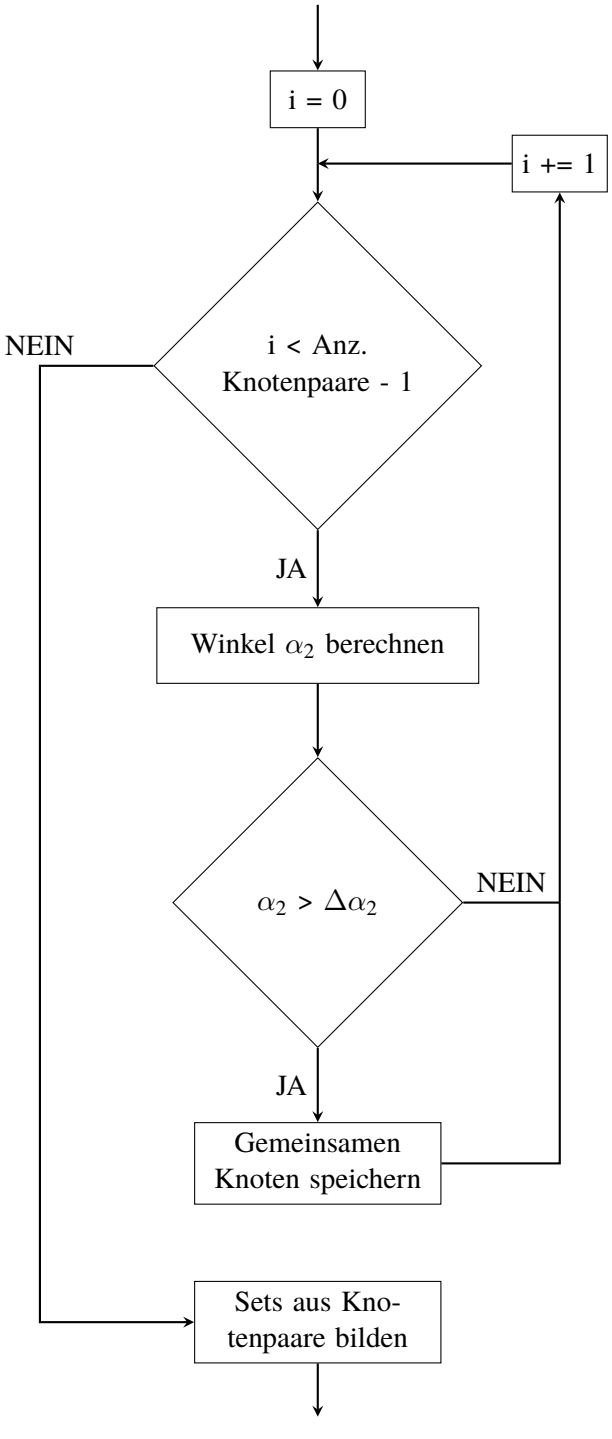

Abbildung 3.5: Ablauf Einteilung in Knotengruppen

Die Generierung der neuen Elemente erfolgt für ein neues Part-Objekt, welches unmittelbar davor erzeugt wird. Bei einem Part-Objekt handelt es sich um ein einzelnes Bauteil im Gesamtmodell. Die neu generierten Elemente werden, entsprechend der Einteilung der Knotenpaare, in Abaqus zu Element-Sets zusammengefasst und sind nach Beendigung dieses Skripts im Deklaratsionsbaum der Abaqus-Umgebung unter dem Namen *„WELD_SET + fortl. Nr.“* gelistet. Weiters werden sämtliche neu generierten Linienelemente zu einem Gesamtset zusammengefasst und das Basis-Part-Objekt mit dem neu erstellten Part-Objekt, welches die Linienelemente enthält, verbunden. In Abbildung 3.6 ist der Ablauf bis zum Zusammenfügen der beiden Part-Objekte illustriert.

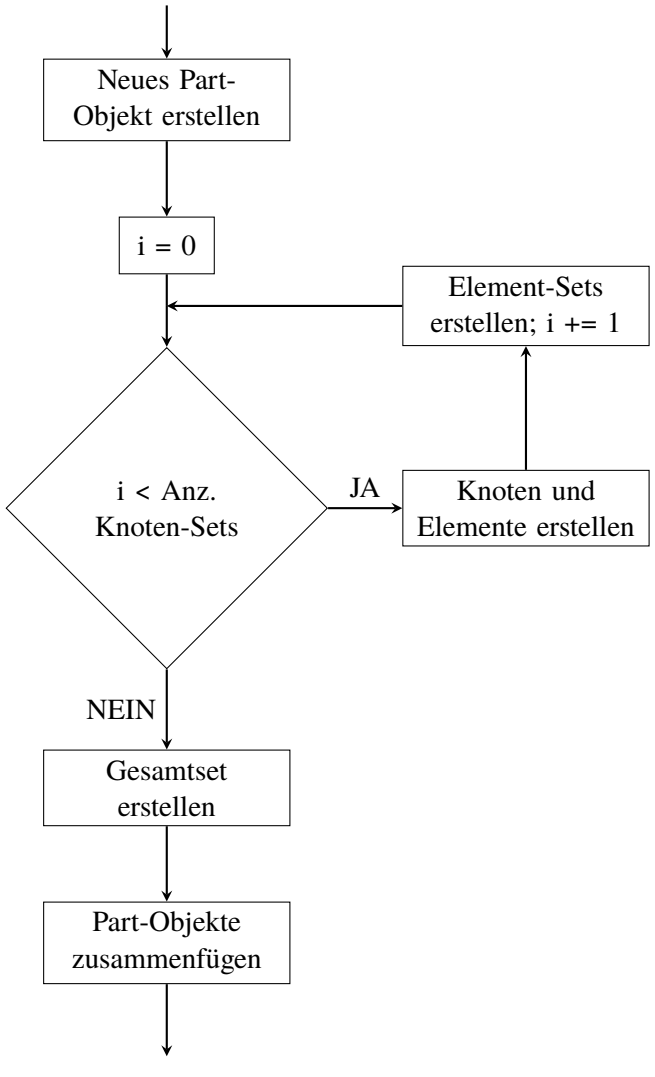

Abbildung 3.6: Ablauf Generierung der Linienelemente

Abschließend erscheint eine Dialog-Box in der vom Anwender ein Name für die Ausgabe-Datei verlangt wird. Der vorgeschlagene Name, der sich aus dem Modellnamen + *„_WELD“* zusammensetzt, sollte idealerweise übernommen werden. Nach erfolgter Eingabe werden die Indizes der Kantenknoten sowie die Indizes der umliegende Elemente in die Ausgabe-Datei geschrieben und diese im Arbeitsverzeichnis von Abaqus gespeichert. Wird die Eingabe abgebrochen, kommt es zur Beendigung des Skripts und es wird keine Ausgabe-Datei erstellt.

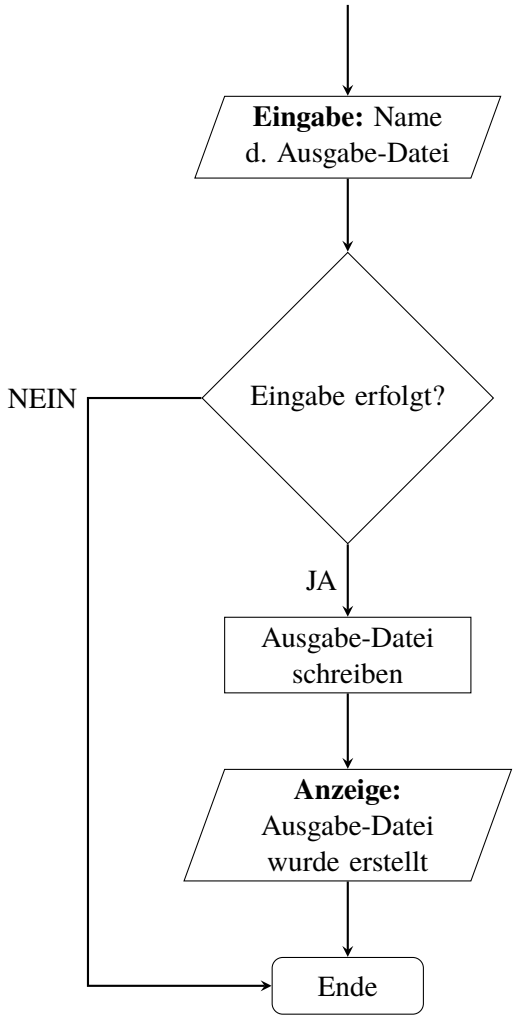

Abbildung 3.7: Ablauf Schreiben der Ausgabe-Datei

3.3 Teil B - „UpdateMain.py“ und „Update.py“

Aufgrund der Diskretisierung mit linearen Elementen stehen die Normalvektoren zweier benachbarter Schalen-Elemente bei runden Geometrien zwangsläufig mit einem gewissen Winkel zueinander. Ist dieser Winkel größer als $\Delta\alpha_1$ wird an diesen Stellen fälschlicherweise eine Kante detektiert und Linienelemente generiert. Es gibt einerseits die Möglichkeit, den Eingabeparamter $\Delta\alpha_1$ anzupassen und die Skript-Datei erneut auszuführen, oder man entfernt die überflüssigen Linienelemente händisch. Hierzu wird in Abaqus ein neues Element-Set mit dem Namen *„DEL-“ + fortl. Nr.* erstellt und die zu löschenden Elemente selektiert.

Es können auch Linienelemente händisch hinzugefügt werden. Dies wäre der Fall, wenn ein Stumpfstoß zweier Plattenbauteile notwendig ist. Die Normalvektoren der beiden benachbarten Schalen-Elemente würden demnach parallel zueinander verlaufen und daher keine Kanten detektiert werden. Werden vom Benutzer transformierte Spannungen in diesen Bereichen gewünscht, gibt es die Möglichkeit die Knoten entlang eines solchen Stumpfstoßes in einen Knoten-Set mit dem Namen *„ADD-“ + fortl. Nr.* zusammenzufassen.

Durch das Ausführen des Skripts *„UpdateMain.py“* wird das Modell und die Ausgabe-Datei, welche die notwendigen Informationen für die Spannungstransformation im Postprozessing enthält, angepasst. Zunächst erscheint eine Dialog-Box in der vom Anwender der Name der Ausgabe-Datei eingegeben werden sollte. Vorgeschlagen wird der Modellnamen + „_$WELD$“. Die Datei mit den notwendigen Informationen wird eingelesen und das Skript *„Update.py“* ausgeführt.

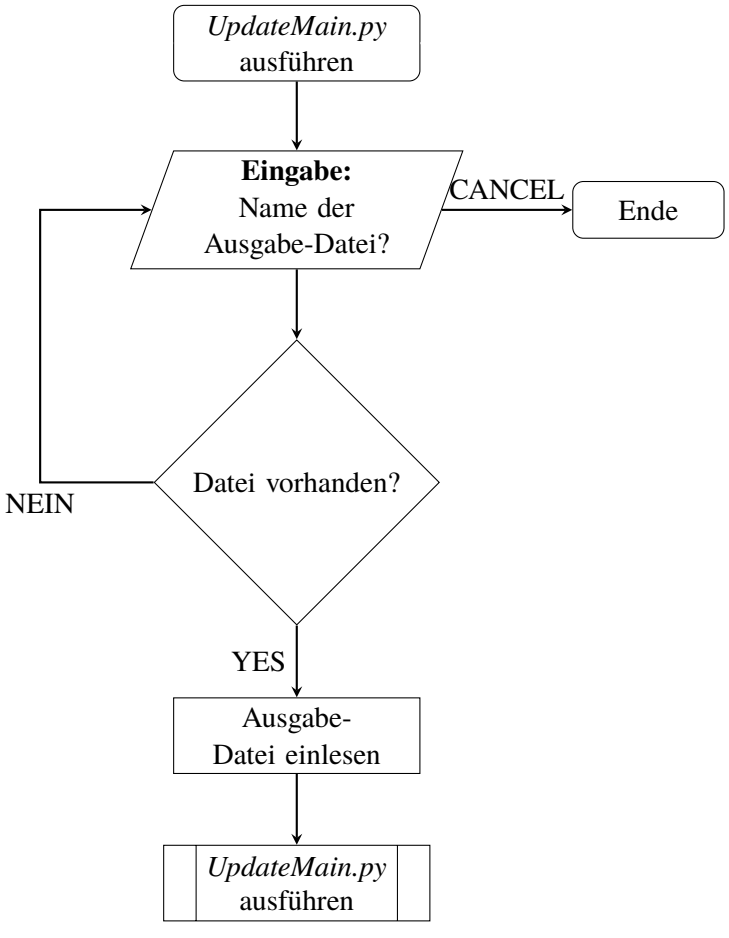

Abbildung 3.8: Ablauf von „*UpdateMain.py*“

Kommt es zum Ausführen der Skript-Datei „*Update.py*“ werden zunächst sämtliche Namen (Schlüssel) der bei der manuell erstellten Sets in eine Liste gespeichert. Im Anschluss wird über diese Listeneinträge iteriert und so auf die entsprechenden Sets im Modell zugegriffen. Handelt es sich um ein „*ADD-*“ Knoten-Set, werden zwischen den enthaltenen Knoten neue Linienelemente generiert und zu neuen Element-Sets zusammengefasst. Die zusätzlichen Knoten- und Elementindizes werden gespeichert.

Bei einem „*DEL-*“-Elementset werden die entsprechenden Linienelemente aus dem Modell gelöscht und die Information der Ausgabe-Datei angepasst.

Wurden alle „*ADD-*“- und „*DEL-*“-Sets abgearbeitet, wird das Gesamtset mit allen Linienelementen aktualisiert. Abbildung 3.9 zeigt den Ablauf der Skript-Datei „*Update.py*“ bis zum Aktualisieren des Gesamtsets.

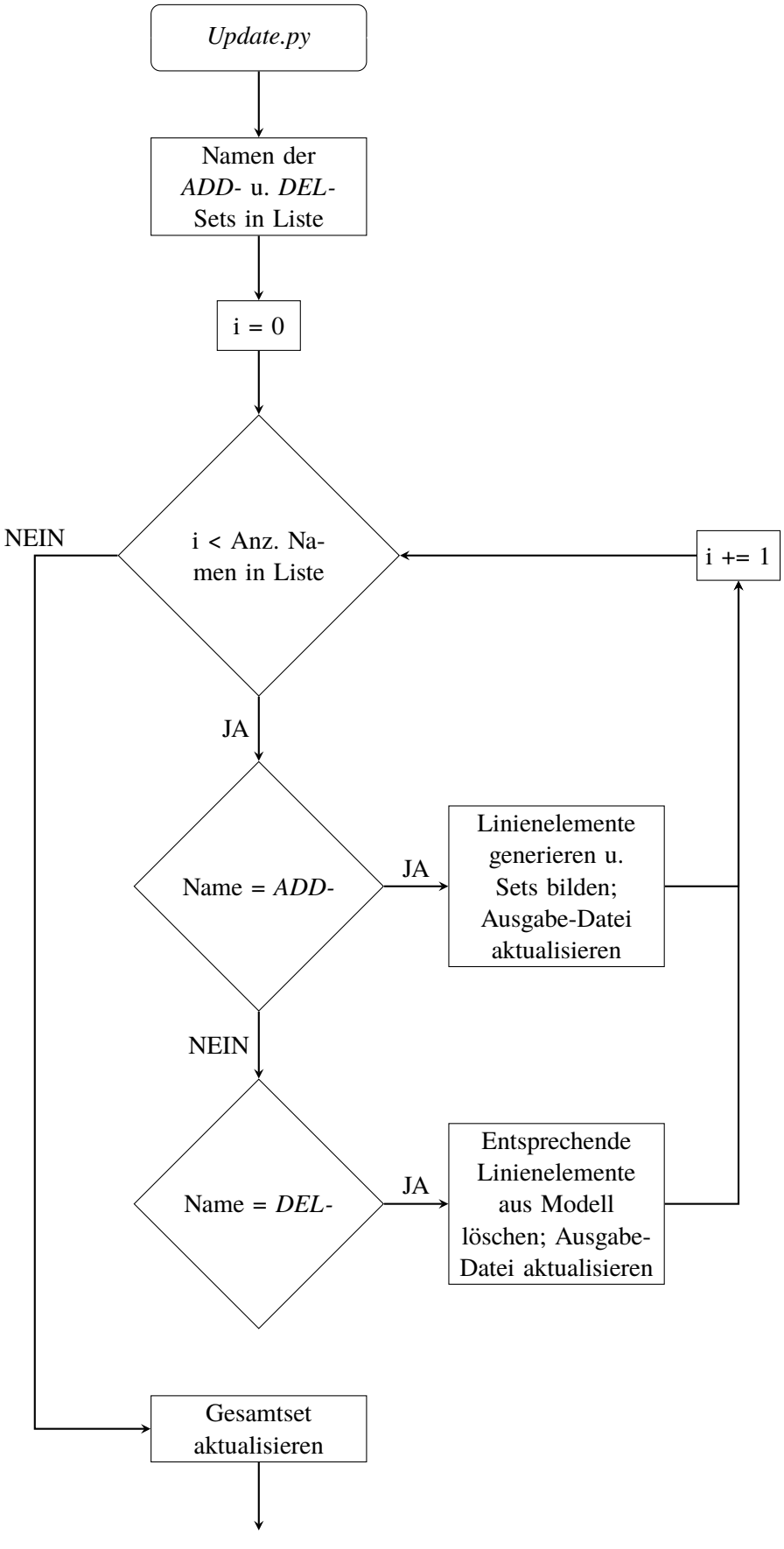

Abbildung 3.9: Ablauf der Bearbeitung des Modells

Im Anschluss wird ein neues Input-File des Modells mit allen Linienelementzügen, welche die potentiellen Schweißnäte repräsentieren, erstellt. Es erscheint nochmals eine Dialog-Box in der vom Anwender der Name der Ausgabe-Datei eingegeben werden sollte. Vorgeschlagen wird erneut der Modellnamen + *„_WELD“*. Die Datei wird mit dem entsprechenden Namen in das Arbeitsverzeichnis von Abaqus gespeichert und das Skript beendet.

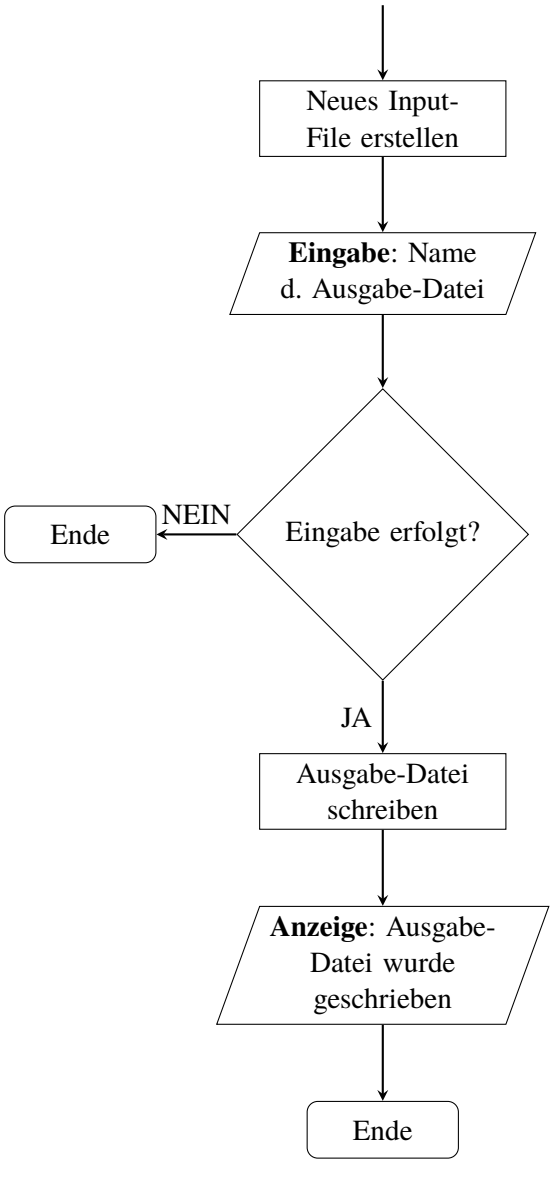

Abbildung 3.10: Ablauf Schreiben der Ausgabe-Datei

3.4 Teil C - „StressTransformMain.py“ und „StressTransform.py“

Im Anschluss an die Analyse erfolgt die Spannungstransformation jener Elemente, welche sich an den identifizierten Scheißnaht-Linienelementen befinden. Ist das gewünschte analysierte Modell (ODB-Datei) im Postprozessing-Bereich der Abaqus-Umgebung geöffnet, kann das Skript *„StressTransformMain.py“* ausgeführt werden. Zunächst erscheint eine Dialog-Box in der vom Anwender der Name der passenden Ausgabe-Datei eingegeben werden soll. Vorgeschlagen wird der Modellname + *„_Weld“*, so wie es bei der Abspeicherung der Datei im Präprozessing ebenfalls vorgeschlagen wird. Nach der Eingabe wird geprüft, ob die entsprechende Datei im aktuellen Arbeitsverzeichnis vorhanden ist. Wurde die Datei nicht gefunden, erscheint die Dialog-Box erneut. Durch Drücken der Schaltfläche *„CANCEL“* kann das Skript beendet werden. Wurde die Datei gefunden, kommt es zum Ausführen das Skripts *„StressTransform.py“* in welcher die Spannungstransformation folgt.

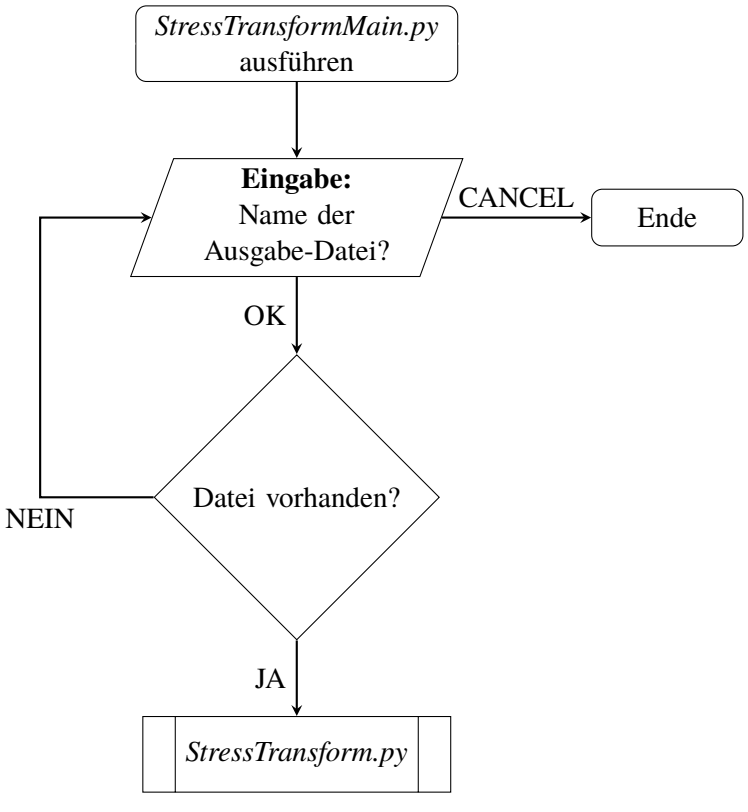

Abbildung 3.11: Ablauf von *„ StressTransformMain.py“*

Zu Beginn wird die passende Ausgabe-Datei eingelesen. Es folgt das Herausfiltern jener Elemente, welche sich an Kreuzungsknoten der im Preprocessing generierten Linienelementzügen befinden. Diese Elemente liegen an zwei potentiellen Schweißnähten und müssen daher bei der Spannungstransformation gesondert behandelt werden.

Der nächste Schritt ist das Erstellen von Abaqus-Element-Sets. Ein Set besteht dabei aus den Schalen-Elementen, welche an einer zusammenhängenden Kante liegen. Dies bietet den Vorteil, dass nach Fertigstellung der Transformation, einzelne Element-Sets je nach Belieben ein- und ausgeblendet werden können.

Weiters wird eine Liste mit den Werten der Transformationswinkel zwischen den lokalen 2-Achsen der Schalen-Elemente und den entsprechenden Elementkanten, an denen ein Linienelement generiert wurde, aufgestellt.

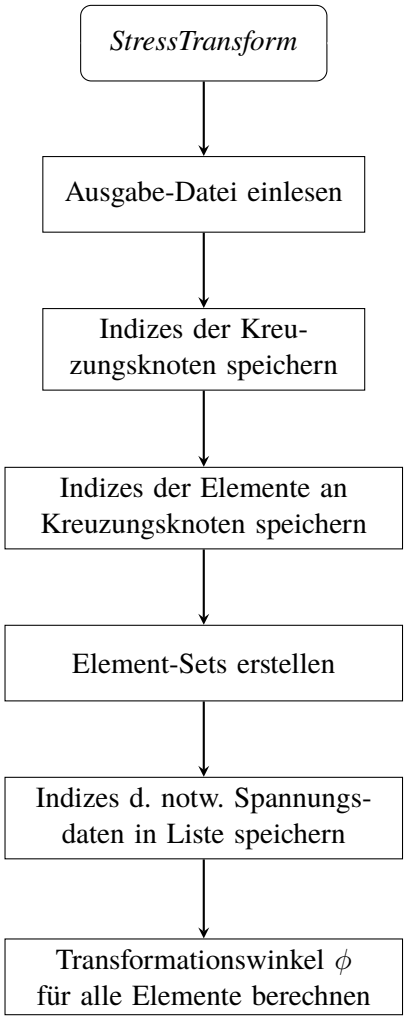

Abbildung 3.12: Ablauf von „*StressTransform.py*“ bis zur Berechnung des Transfomationswinkels

Anschließend folgen zwei Iterationsebenen. Zunächst wird über die einzelnen Berechnungsschritte (Steps) iteriert. Es werden die Spannungsdaten der betroffenen Elemente gelesen. Danach erfolgt die Iteration über diese Elemente. Dabei treten 2 Fälle auf, welche in weiterer Folge genauer erläutert werden.

Fall 1: Element liegt nicht an einem Kreuzungsknoten

Die Spannungskomponenten der beiden Knoten, welche an der potentiellen Schweißnaht liegen werden in Richtung der Schweißnahtachse und in die Richtung normal dazu transformiert. Die absolut größeren Werte der jeweiligen transformierten Spannungskomponenten werden für alle Knoten des Elements zwischengespeichert. Die einzelnen transformierten Spannungskomponenten sind aus diesem Grund über das Element konstant.

Beispiel zu Fall 1

Abbildung 3.13 zeigt zwei benachbarte Schalen-Elemente. Entlang der gemeinsamen Kante wurde eine Schweißnaht detektiert. Nun werden die Spannungskomponenten, welche im lokalen 1,2-Koordinatensystem gegeben sind, in die Richtung der Nahtachse und in die Richtung normal dazu transformiert und wie zuvor erwähnt den entsprechenden Knoten zugewiesen. In Tabelle 3.1 sind die Spannungskomponenten an den Knoten exemplarisch für das Element 2 aufgelistet.

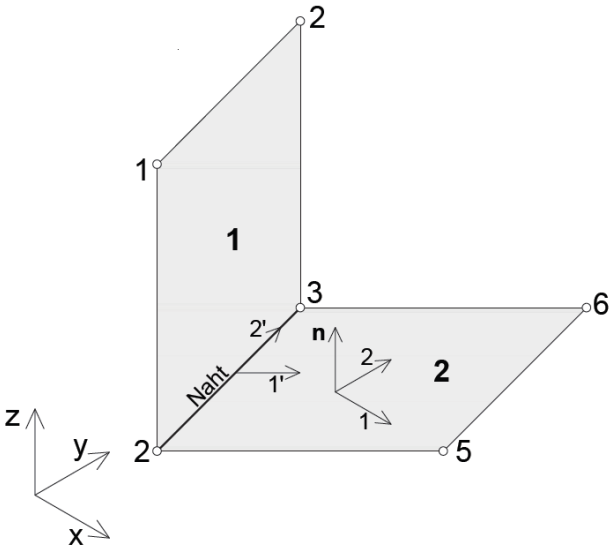

Abbildung 3.13: Zwei benachbarte Schalen-Elemente

Knoten	Komponenten		
	S11	S22	S12
2	**5262.15**	**-1246.13**	**2194.44**
5	4789.73	-1718.55	1336.09
6	4585.61	-1922.66	1706.96
3	**5058.04**	**-1450.24**	**2565.30**

Tabelle 3.1: Spannungskomponenten an den Elementknoten des Elements 2

Die transformierten Spannungen für die relevanten Knoten 2 und 3 und die absolut größeren Komponenten, welche in weiterer Folge allen Knoten des Elements 2 zugewiesen werden, kann man der Tabelle 3.2 entnehmen.

Knoten	Komponenten		
	S11*	S22*	S12*
2	-186.42	4202.45	3254.14
3	-761.40	4369.20	3254.14
max[|S_2*|,|S_3*|]	**-761.40**	**4369.20**	**3254.14**

Tabelle 3.2: Transformierte Spannungskomponenten an den Knoten 2 und 3 des Elements 2

Fall 2: Element liegt an einem Kreuzungsknoten

Elemente an Kreuzungsknoten liegen potentiell an zwei Schweißnähten. Die Zuweisung der Spannungskomponeten für den Kreuzungsknoten ist nicht eindeutig und muss daher gesondert behandelt werden. Während der Iteration kommen diese Elemente zweimal vor. Beim ersten mal werden die absolut größeren Werte der transformierten Spannungskomponenten aus den beiden an der Schweißnaht liegenden Knoten für den inneren Knoten des Knotenpaares zwischengespeichert. Bei wiederholtem Auftreten werden wieder die Spannungen der beiden Schweißnahtknoten transformiert und für den inneren Knoten des Knotenpaares der Schweißnaht die absolut größeren Spannungskomponenten zwischengespeichert.

Der Kreuzungsknoten erhählt im Anschluss die absolut größeren und der Innenknoten, welcher nicht an einer Schweißnaht liegt, die absolut kleineren Spannungskomponeten der beiden zwischengespeicherten Spannungszuständen.

Beispiel zu Fall 2

Abbildung 3.14 zeigt drei, an dem Kreuzungsknoten mit dem Index 3 liegende, Schalen-Elemente. Exemplarisch soll hier für Element 2 die Zuweisung der transformierten Spannungen an die entsprechenden Knoten gezeigt werden. Die Spannungskomponenten der Elementknoten sind in Tabelle 3.3 zusammengefasst.

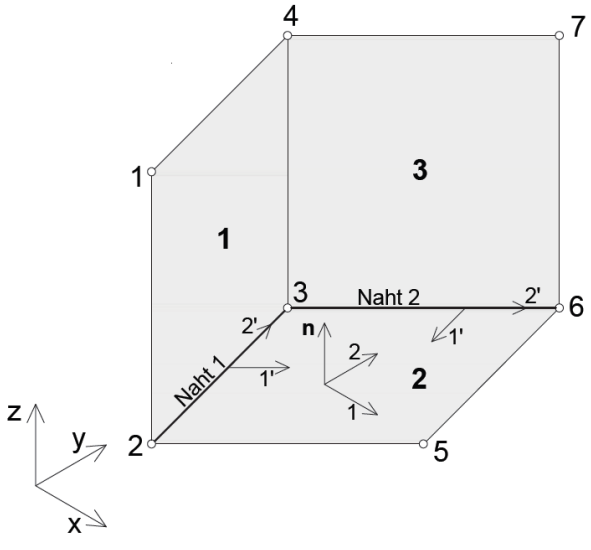

Abbildung 3.14: Benachbarte Elemente an einem Kreuzungsknoten

Knoten	Komponenten		
	S11	S22	S12
2	**-316.50**	**-837.49**	**505.70**
5	-333.98	-854.97	537.47
6	**-576.37**	**-1097.36**	**97.07**
3	**-558.89**	**-1079.88**	**65.30**

Tabelle 3.3: Spannungskomponenten an den Elementknoten des Elements 2

Die transformierten Spannungen der Knoten 2 und 3 an der Naht 1 sind der Tabelle 3.4 zu entnehmen. Die absolut größeren Komponenten werden dem Knoten 2 zugewiesen.

Knoten (Naht 1)	Komponenten		
	S11*	S22*	S12*
2	-71.30	-1082.69	-260.50
3	-754.09	-884.68	-260.50
max[|S_2*|,|S_3*|]	**-754.09**	**-1082.69**	**-260.50**

Tabelle 3.4: Transformierte Spannungskomponenten an den Elementknoten des Elements 2

In Tabelle 3.5 sind die transformierten Spannungen der an der Naht 2 liegenden Knoten 3 und 6 angeführt. Die absolut größeren Komponenten werden dem Knoten 6 zugewiesen.

Knoten (Naht 2)	Komponenten		
	S11*	S22*	S12*
3	-884.68	-754.09	260.50
6	-933.94	-739.80	260.50
max[|S_3*|,|S_6*|]	**-933.94**	**-754.09**	**260.50**

Tabelle 3.5: Transformierte Spannungskomponenten an den Elementknoten des Elements 2

Dem Kreuzungsknoten, in diesem Beispiel der Knoten mit dem Index 3, werden die absolut größeren Spannungskomponenten aus den beiden Nähten zugewiesen. Der Innenknoten, im Beispiel der Knoten mit dem Index 5, erhält die absolut kleineren Spannungskomponenten aus den beiden Nähten.

Tabelle 3.6 enthält die endgültig zugewiesenen transformierten Spannungen an den Knoten des Elements 2.

Knoten	Komponenten		
	S11*	S22*	S12*
2	-754.09	-1082.69	260.50
5	-754.09	-754.09	260.50
6	-933.94	-754.09	260.50
3	-933.94	-1082.69	260.50

Tabelle 3.6: Endgültig zugewiesenen transformierten Spannungen des Elements 2

Der Ablauf der gesamten Spannungstransformation für sämtliche Berechnungsschritte ist in Abbildung 3.15 dargestellt.

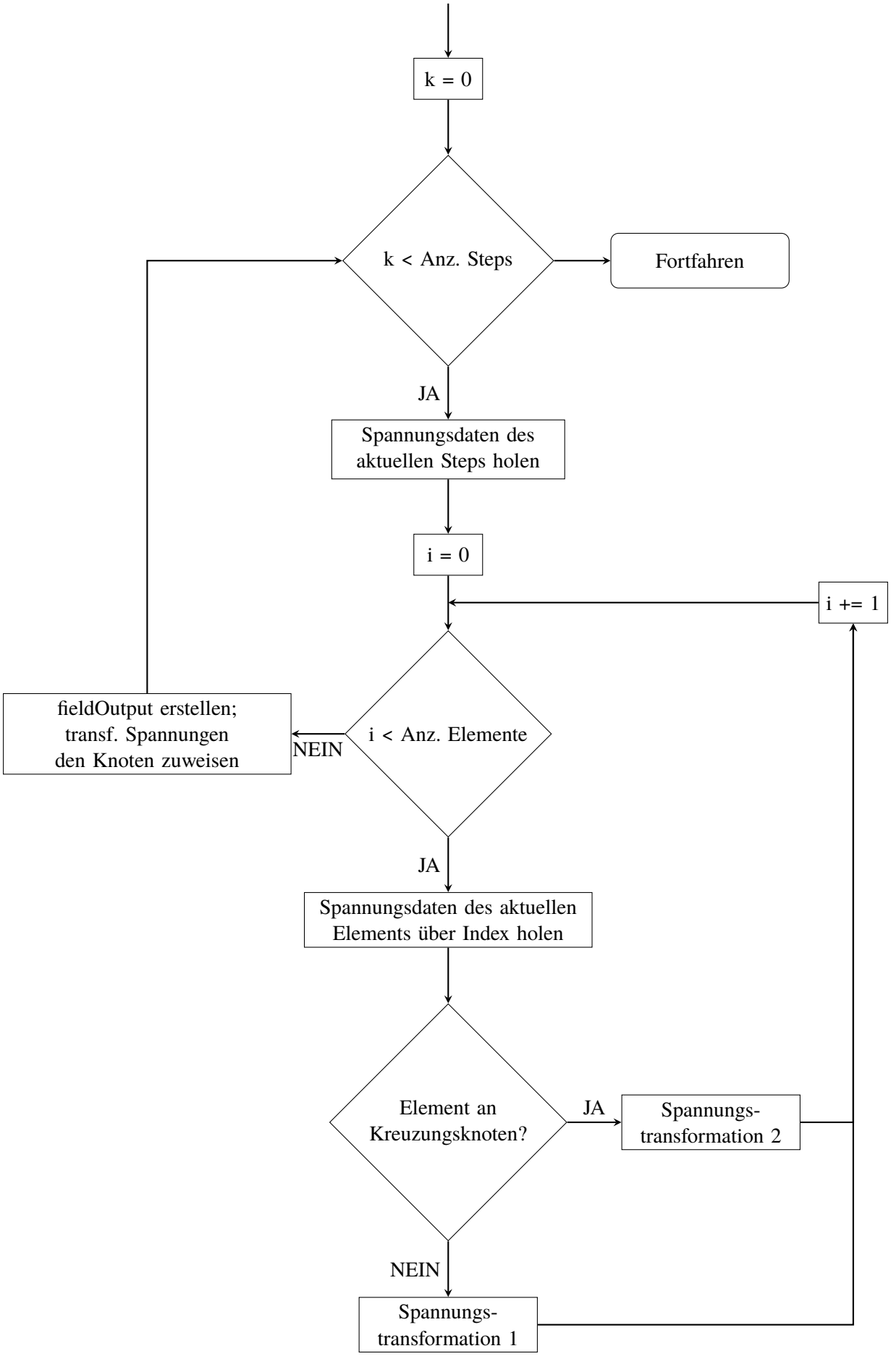

Abbildung 3.15: Ablauf der Spannungstransformation

Nach erfolgreicher Spannungstransformation wird die bearbeitete Datenbank in einer ODB-Datei gespeichert. Abschließend erscheint die Meldung, dass die Spannungstransformation erfolgreich war. Mit Drücken der Schaltfläche „OK“ wird das Skript beendet.

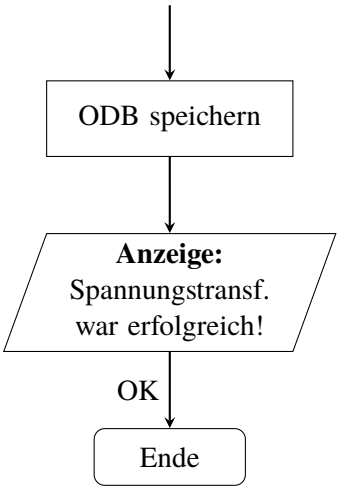

Abbildung 3.16: Ablauf vom Speichern der ODB-Datei bis zur Beendigung des Skripts

3.5 Auffinden von Nachbarelemente - Eine Gegenüberstellung unterschiedlicher Methoden

Das Skript *„WeldGenerator.py“* generiert in den FE-Plattenmodellen Linienelemente an jenen Kanten an denen in der Produktion der Bauteile Schweißnähte aus konstruktiven Gründen angeordnet werden müssen. Zu diesem Zweck ist es zunächst notwendig Nachbarelemente, also Elemente welche sich eine gemeinsame Elementkante teilen, aufzufinden. Im Zuge dieser Arbeit wurden mehrere Varianten implementiert, welche in den folgenden Abschnitten erkärt sind. Ziel der einzelnen Methoden ist die Erstellung einer Liste mit den Indizes der Knotenpaare, welche die Elementkanten definieren und einer zugeordneten Liste der Indizes der an dieser Kante liegenden Elemente.

Methode 1:

Es wird zunächst über die Konnektivitäten aller Schalen-Elemente iteriert und Knotenpaare gebildet. Ein Knotenpaar besteht aus den Indizes der Knoten einer Elementkante. Die Knotenpaare und deren Element-Indizes werden in zwei separate Listen gespeichert.

Anschließend wird jedes Knotenpaar mit jedem anderen in der zuvor erstellten Liste verglichen. Kommt es zur Übereinstimmung zweier Knotenpaar-Einträge, werden die dazugehöhrigen Element-Indizes zusammengefasst. Dies sind nun die Indizes jener Elemente, welche sich im Modell ein gemeinsames Knotenpaar bzw. eine gemeinsame Elementkante teilen. Es handelt sich bei der vorgestellten Methode um eine naive und sehr zeitintensive Hernagehensweise an die Problemstellung. Die erwartete Komplexität ist quadratisch mit der Anzahl der Elemente n, also $\mathcal{O}(n^2)$.

Methode 2:

Ausgehend vom Mittelpunkt jedes Elements wird eine Bounding-Box mit Hilfe einer Abaqus Funktion [3] aufgespannt. Die Kantenlänge der Bounding-Box ist mit der 4-fachen Länge der längsten Elementkante des aktuellen Elements definiert. Es werden für jeden Iterationsschritt nur jene Elemente als potentielle Nachbarn berücksichtigt, welche sich in der jeweiligen Bounding-Box befinden.

Innerhalb einer Bounding-Box erfolgt das Vergleichen der Knotenpaare und Zusammenfassen der Element-Indizes in der selben Form wie bei Methode 1. Zusätzlich müssen jene Einträge entfernt werden, welche aufgrund der Überschneidungen der einzelnen Bounding-Boxen öfter als einmal enthalten sind. Die erwartete asymptotische Komplexität ist mehr als $\mathcal{O}(nlogn)$.

Anschließend wird über die Knotenpaar-Summen iteriert und jene Einträge der Element-Indizes zusammengefasst, welche die identen Summenwerte aufweisen.

Methode 3:

Abaqus stellt eine Funktion zur Verfügung [3], in der alle Element-Kanten-Objekte des Modells abgerufen werden können. Mithilfe des Objekts kann neben den Knoten u.a. auch auf jene Elemente zugegriffen werden, welche an dieser Kante liegen. Im Anschluss an den Funktionsaufruf wird über diese Objekte iteriert und die Indizes der Elemente, welche an dieser gemeinsamen Kante liegen zusammengefasst gespeichert. Die Indizes der beiden Knoten der Kante werden wiederum in der gleichen Reihenfolge in eine separate Liste gespeichert. Der Aufwand für diese Methode ist $\mathcal{O}(n)$ und daher optimal.

Gegenüberstellung der einzelnen Methoden

Die Zeitaufwendungen der einzelnen Methoden für das Auffinden der benachbarten Elemente sind in Tabelle 3.7 sowie in Abbildung 3.17 gegenübergestellt. Methode 1 bestitzt eine quadratische Komplexität. Dies führt bereits bei mittelgroßen Modellen mit ca. 10.000 Elementen zu einem enormen Zeitaufwand und ist daher für eine sehr große Anzahl an Elementen ungeeignet.

Bei der Methoden 2 ist der zeitliche Aufwand bereits deutlich reduziert. Für Modelle mit einer Elementanzahl von 25.000 und mehr ist diese Methode in dieser Form jedoch ebenfalls unbefriedigend.

Methode 3 besitzt eine asymptotisch lineare Komplexität und ist daher auch bei einer extrem großen Elementanzahl sehr effizient. Diese Variante stellt die im Programmcode verwendete Methode für das Finden von Nachbarelementen dar.

Getestet wurden die Algorithmen auf einem TOSHIBA Satellite Laptop mit einem Intel(R) Core(TM) i7-4700HQ CPU mit 2.4 GHz.

Elementanzahl n	Benötigte Zeit t [s]		
	Methode 1	Methode 2	Methode 3
1792	16.2	14.8	1.8
6300	148.1	72.3	6.2
11200	417.2	216.6	10.8
20412	1621.8	558.1	19.9
30954	-	1153.2	28.0

Tabelle 3.7: Gegenüberstellung der einzelnen Methoden

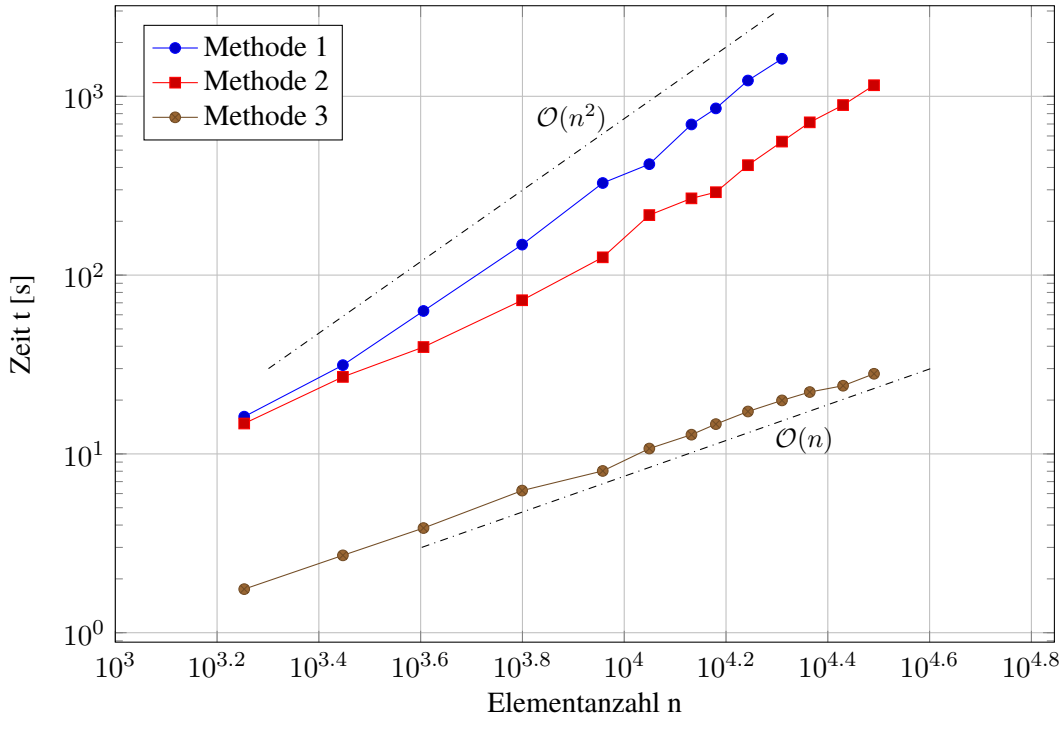

Abbildung 3.17: Gegenüberstellung der einzelnen Methoden (log-log)

4 Verifizierung des Programmcodes

4.1 Allgemeines

In diesem Kapitel soll anhand einiger Beispiele die Verifizierung des Codes erfolgen und dessen Leistungsfähigkeit und eventuelle Einschränkungen aufgezeigt werden. Bei den Beispielen 1 bis 5 handelt es sich um sehr einfach gehaltene Modelle mit einer relativ geringen Anzahl an Schalen-Elementen. Diese kleinen Modelle dienen nur zur Überprüfung der korrekten Arbeitsweise einzelner Funktionen der programmtechnischen Umsetzung und haben keinen praktischen Hintergrund. In Beispiel 6 wird anschließend die programmtechnische Umsetzung an einem realen Bauteil getestet. Wie in Kapitel 1 erwähnt, werden nur Schalen-Elemente mit linearen Ansätzen verwendet.

4.2 Beispiel 1: Detektion von Elementkanten

Das Test-Modell besteht aus 4 einzelnen Platten-Bauteilen, wobei die Grundplatte parallel zur xy-Ebene liegt und die übrigen 3 Bauteile mit unterschiedlichen Winkeln anschließen. Es soll gezeigt werden, dass ab einem bestimmten Winkel zwischen den Normalvektoren zweier benachbarter Schalen-Elemente eine Kante als Schweißnaht detektiert und an diesen Stellen neue Lienienelementzüge generiert werden.

Abbildung 4.1 zeigt das Test-Modell. Das linke Plattenbauteil schließt in einem Winkel von 10.5 [°] an die Grundplatte an, das rechte Plattenbauteil in einem Winkel von 9.5 [°] und die dritte Platte ist zur Grundplattenachse orthogonal angeschlossen.

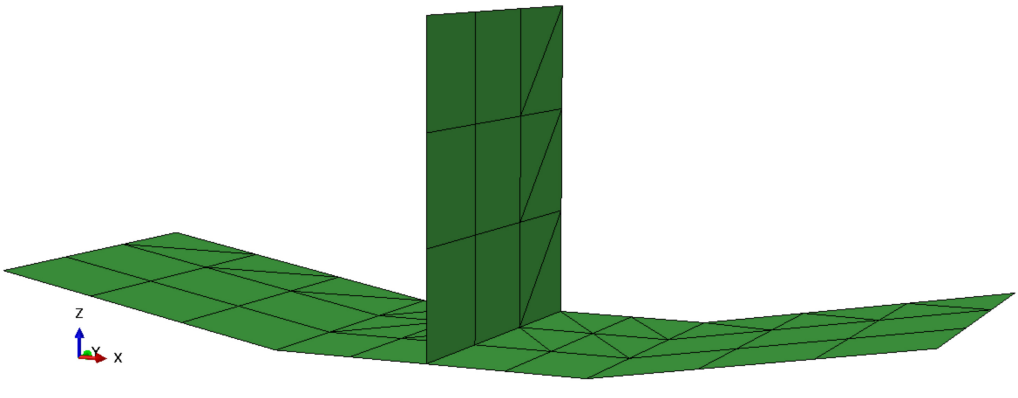

Abbildung 4.1: Grundmodell

Getestet wird an einem hybriden Netz, bestehend aus S3- und S4-Elementen. Dabei wird getestet, dass jede mögliche Kombination von Nachbarschaftsbeziehungen (S3-S3, S3-S4, S4-S4) vom Algorithmus abgedeckt ist.

Beim Ausführen des Skripts *„WeldGeneratorMain.py“* und Eingabe des Wertes 10 [°] für $\Delta\alpha_1$ werden an den Grenzen der Bauteile, welche mit einem Winkel größer-gleich 10 [°] zueinander liegen, neue Linienelemente generiert. Abbildung 4.2 zeigt die Eingabemaske für beide Toleranzparameter $\Delta\alpha_1$ und $\Delta\alpha_2$ und in Abbildung 4.3 ist das Modell nach Ausführen des entsprechenden Skripts zu sehen. Die neu generierten Linien-Elemente sind rot hervorgehoben.

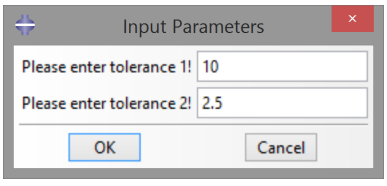

Abbildung 4.2: Eingabeparameter für Beispiel 2

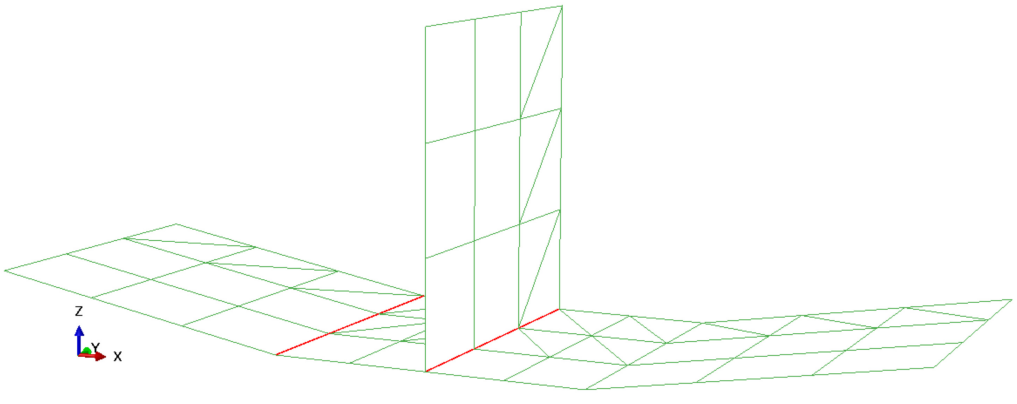

Abbildung 4.3: Modell mit den neu generierten Linienelemente (rot)

4.3 Beispiel 2: Generierung von geschlossenen Linienzüge

Mit diesem Test-Beispiel soll gezeigt werden, dass bei Eingabe eines bestimmten Werts für $\Delta\alpha_2$, geschlossene Linienelementzüge generiert werden können. Also, dass einzelne Linienelemente zu Element-Sets zusammengefasst werden. Dieses Zusammenfassen von einzelnen Elementen in Element-Gruppen ermöglicht die Darstellung und Modifikation von durchgehenden Schweißnähten.

Das Test-Modell besteht aus zwei Bauteilen wie in Abbildung 4.4 ersichtlich. Es handelt sich hierbei um eine Platte an welcher eine zylindrische Mantelfläche angeschlossen ist. Die Diskretisierung erfolgt dabei mit linearen Viereckelementen. Lineare Elemente haben unter anderem den Nachteil, dass sie bei der Vernetzung von gerundeten Geometrien einen großen Diskretisierungsfehler erzeugen und benachbarte Elemente immer einen gewissen Winkel zueinander aufweisen.

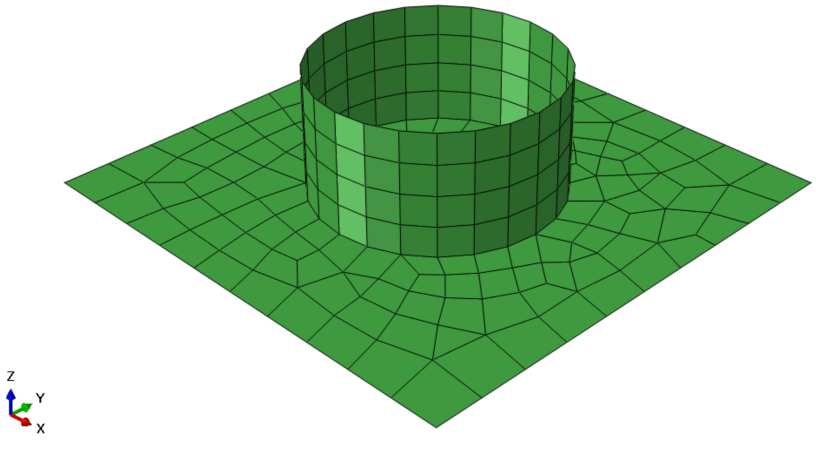

Abbildung 4.4: Platte mit zylindrischer Mantelfläche

In diesem Beispiel wird die Anzahl der Elemente in Umfangsrichtung der zylindrischen Mantelfläche so gewählt, dass die Richtungsvektoren zweier benachbarter neu zu generierende Linienelemente einen Winkel von 15 [°] einnehmen.

Wird das Skript *„WeldGeneratorMain.py“* ausgeführt und für $\Delta\alpha_2$ ein Wert von 16 [°] eingegeben, erfolgt die Generierung eines einzigen Linienzuges entlang des Anschlusses. In Abbildung 4.5 ist das entsprechende Ergebnis zu sehen. Der Linienelementzug ist rot hervorgehoben.

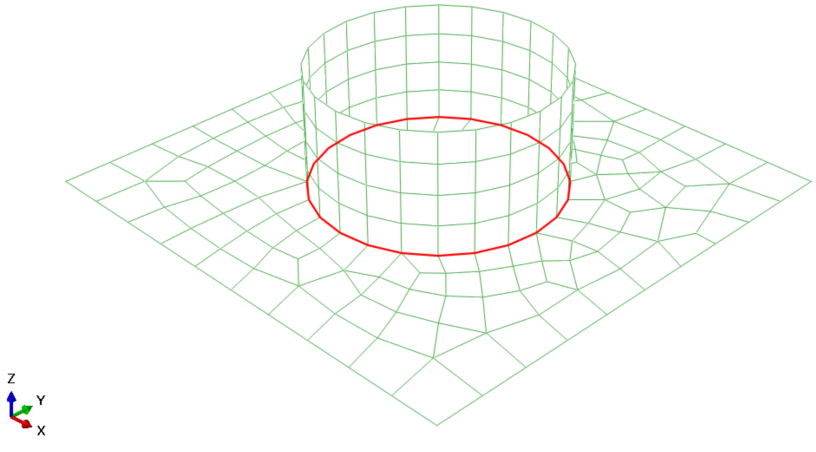

Abbildung 4.5: Geschlossener Linienelementzug entlang des Anschlusses

4.4 Beispiel 3: Teilen von S4-Elementen

In Kapitel 3.4 wurde gezeigt wie die Zuweisung der transformierten Spannungen eines S4-Elements, welches sich an einem Kreuzungsknoten befindet, erfolgt. Ein Kreuzungsknoten liegt an zwei potentiellen Schweißnähten. Die eindeutige Zuordnung zu einer Naht ist daher nicht möglich. Um dies zu vermeiden, können S4-Elemente an Kreuzungsknoten in zwei S3-Elemente geteilt werden.

Diese Funktionalität wird anhand des in Abbildung 4.6 gezeigten Modells getestet. Die Elemente mit den Indizes 1,13 und 9 liegen an einem Kreuzungsknoten und damnach jeweils an 2 Nähten, welche rot hervorgehoben sind.

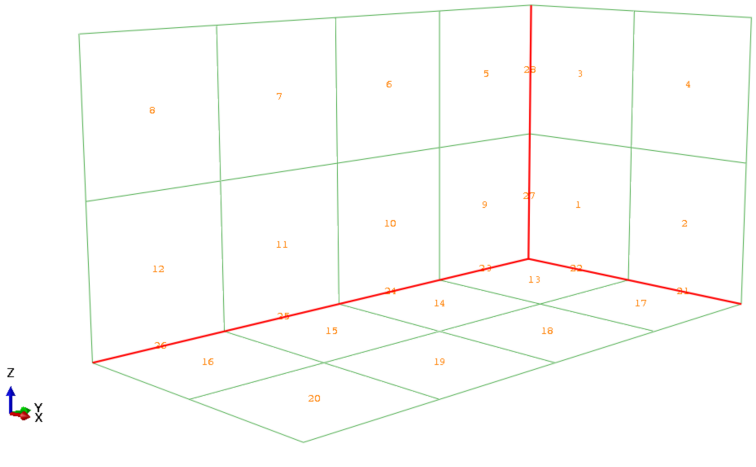

Abbildung 4.6: Grundmodell des Beispiels 3

Der Benutzer kann via Dialogbox, dargestellt in Abbildung 4.7, die Elementteilung veranlassen. Danach stehen zwei S3-Elemente für die beiden Schweißnähte zur Darstellung der im Postprozessing transformierten Spannungen zur Verfügung.

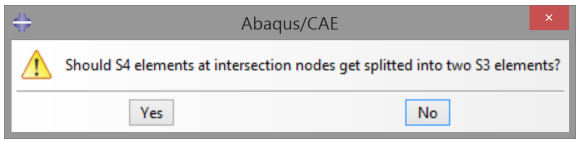

Abbildung 4.7: Abfrage ob Elemente geteilt werden sollen

Abbildung 4.8 illustriert das, nach Ausführen des Skripts, bearbeitete Modell. Der Verlauf der potentiellen Schweißnähte ist rot hervorgehoben.

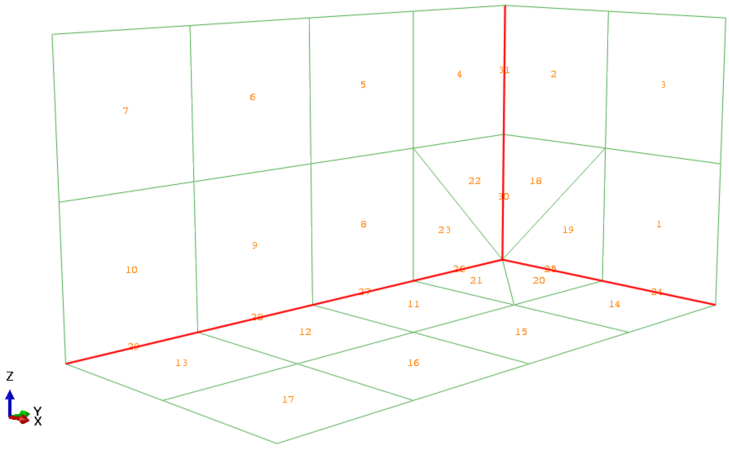

Abbildung 4.8: Geteilte Elemente an Kreuzungsknoten

4.5 Beispiel 4: Manuelle Modifikation der generierten Linienzüge

In diesem Beispiel soll die korrekte Arbeitsweise des Skripts „Update.py“, welches für das Entfernen und Hinzufügen von Linienelementen zuständig ist, untersucht werden.

Abbildung 4.9 illustriert einen Teil einer zylindrische Mantelfläche diskretisiert mit bilinearen S4-Elementen. Dabei ergibt sich ein großer Diskretisierungfehler für die Geometrie. Somit liegen die benachbarten Elemente in Umfangsrichtung, in diesem Fall mit einem Winkel von 15 [°], zueinander.

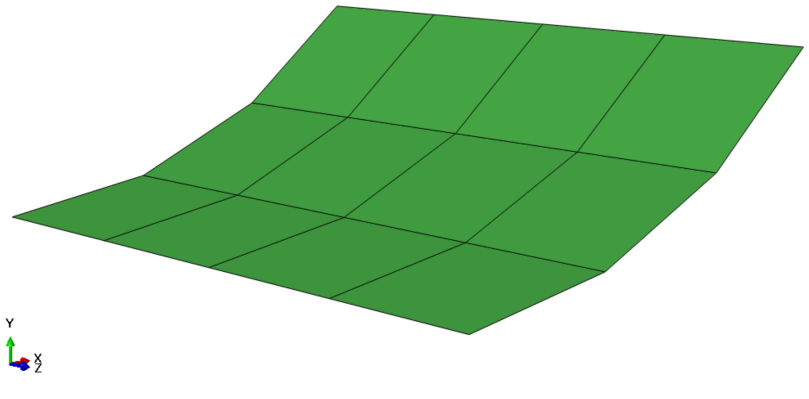

Abbildung 4.9: Teil einer zylindrischen Mantelfäche

Wird vom Anwender beim Ausführen des Skripts *„WeldGeneratorMain.py“* ein Wert kleiner-gleich 15 [°] für $\Delta\alpha_1$ gewählt, werden an den entsprechenden Bereichen, so wie es in Abbildung 4.10 dargestellt wird, überflüssige Linienelemente generiert. Diese sind in Abbildung 4.10 rot hervorgehoben.

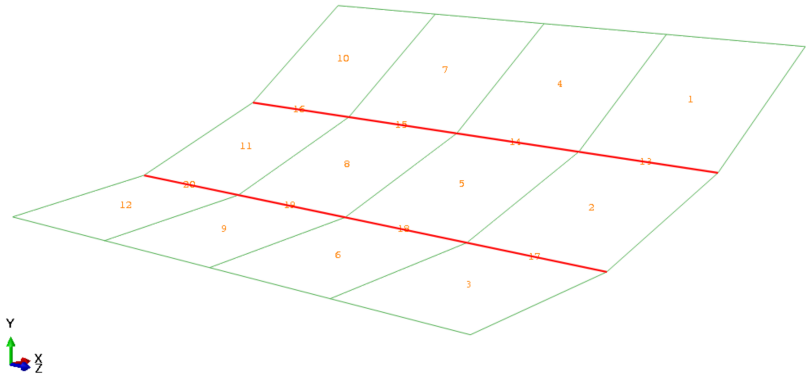

Abbildung 4.10: Modell mit den überflüssigen Linienelementen

Wie in Kapitel 3 beschrieben, hat der Anwender nun zwei Möglichkeiten. Einerseits kann der Eingabeparamter dementsprechend angepasst und das Skript erneut ausgeführt werden, anderseits können die überflüssigen Elemente manuell in Abaqus-Element-Sets zusammengefasst und durch das Ausführen des Skripts *„UpdateMain.py“* entfernt werden.

Im Zuge dieses Test-Beispiels, soll in Umfangsrichtung entlang der Bauteilachse zusätzliche Linienelemente generiert werden. Dies wäre z.B. der Fall, wenn es sich in der Bauteilherstellung in diesem Bereich um einen Stumpfstoß handeln würde. Solche Stumpfstöße werden vom Skript *„WeldGeneratorMain.py“* nicht erfasst. Werden vom Anwender transformierte Spannungen entlang solcher Stöße gewünscht, müssen die entsprechenden Knoten vor Ausführen von *„UpdateMain.py“* manuell in Abaqus-Knoten-Sets zusammengefasst werden.

In Abbildung 4.11 ist das Modell, nach manuellem Erstellen der entsprechenden Sets und Ausführen von *„UpdateMain.py“*, zu sehen. Der hinzugefügte Linienelementzug, welcher die Schweißnaht darstellt, ist dabei hervorgehoben.

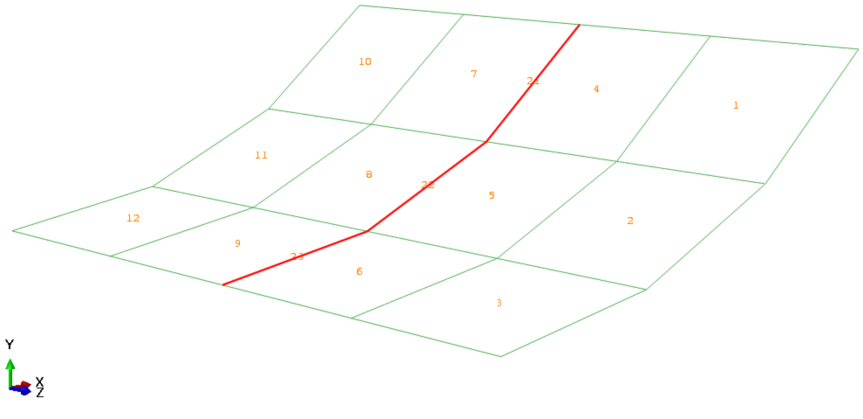

Abbildung 4.11: Modell mit den manuell hinzugefügten Linienelementen

4.6 Beispiel 5: Invarianz der transformierten Spannungen

Im Zuge dieses Tests wird nachgewiesen, dass sich die transformierten Spannungen, welche der Schweißnahtbemessung zugrunde gelegt werden, invariant bezüglich einer Veränderung der Lage des Modells im globalen Koordinatensystem verhalten.

In Abbildung 4.12 ist ein, an der linken Seite eingespannte, T-Träger mit einer am Gurt beanspruchten Flächenlast dargestellt. Die Längsachse des Trägers verläuft parallel zur globalen x-Achse. Abbildung 4.13 zeigt das selbe Modell, jedoch verläuft die Längsachse nicht parallel zur x-Achse.

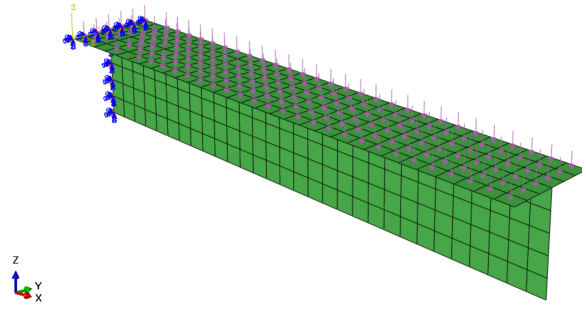

Abbildung 4.12: T-Träger mit Längsachse in x-Richtung

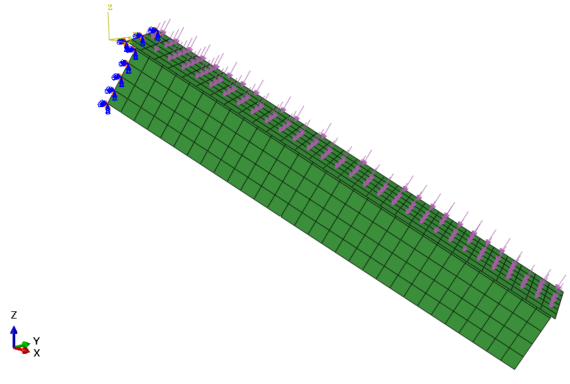

Abbildung 4.13: T-Träger gedreht

Die Spannungskomponeten sind, wie bereits im Abschnitt 2.4.3 gezeigt, im lokalen Koordinatensystem des Schalen-Elements definiert. Es ergeben sich für beide Modelle unterschiedliche Spannungen, was exemplarisch für die Spannungskomponeten S11 in den Abbildungen 4.14 und 4.15 illustriert ist.

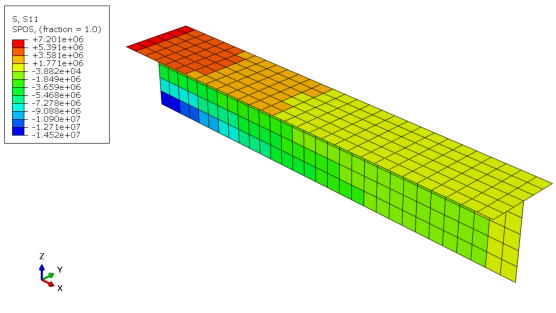

Abbildung 4.14: Spannungen S11

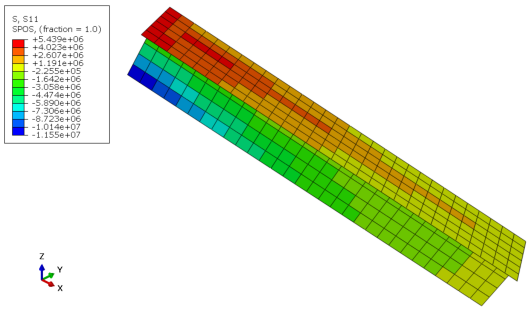

Abbildung 4.15: Spannungen S11

Führt der Anwender das Skript *„StressTransformMain.py“* aus, werden die Spannungen in die Richtung der Nahtachse und in die Richtung normal dazu transformiert.

In den Abbildungen 4.16 und 4.17 ist für die Spannungen in Richtung der Schweißnahtachse (S22*) zu erkennen, dass die transformierten Spannungen der beiden Modelle identen Werte ergeben und somit unabhängig von der Lage des Modells im globalen Koordinatensystem sind.

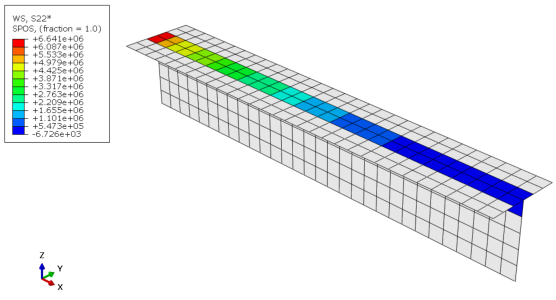

Abbildung 4.16: Spannungen S22*

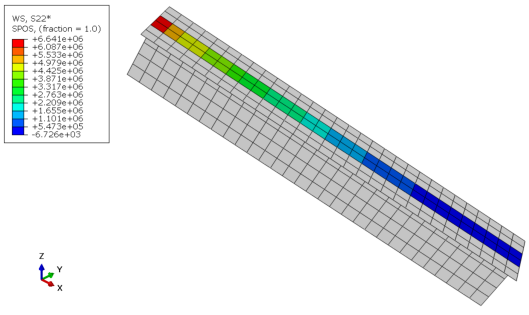

Abbildung 4.17: Spannungen S22*

4.7 Beispiel 6: Realer Anwendungsfall

Nun soll die Funktionalität der programmtechnischen Umsetzung anhand des Modells eines realen Bauteils untersucht werden. Hierzu ist das FEM-Modell einer Fahrwerkschwinge eines Schiffsbeladers in Abbildung 4.18 dargestellt.

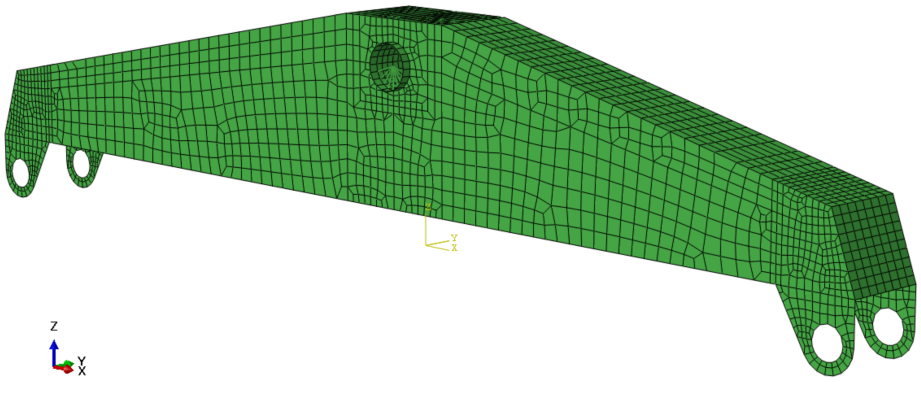

Abbildung 4.18: FEM-Modell einer Fahrwerkschwinge eines Schiffsbeladers

Der Parameter $\Delta\alpha_1$ wird mit 20 [°] so gewählt, dass an den Rundungen des Modells keine überflüssigen Linienelemente generiert werden. Für $\Delta\alpha_2$ werden die vorgeschlagenen 2.5 [°], so wie es in Abbildung 4.19 gezeigt ist, übernommen.

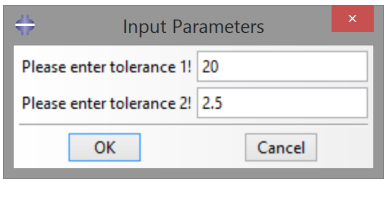

Abbildung 4.19: Inputparameter

Die Möglichkeit zur Teilung von Elementen, welche sich an Kreuzungsknoten befinden, wird nicht gewählt und somit ergibt sich das Modell inklusive der neu generierten Linienelemente (rot) wie in Abbildung 4.20 dargestellt.

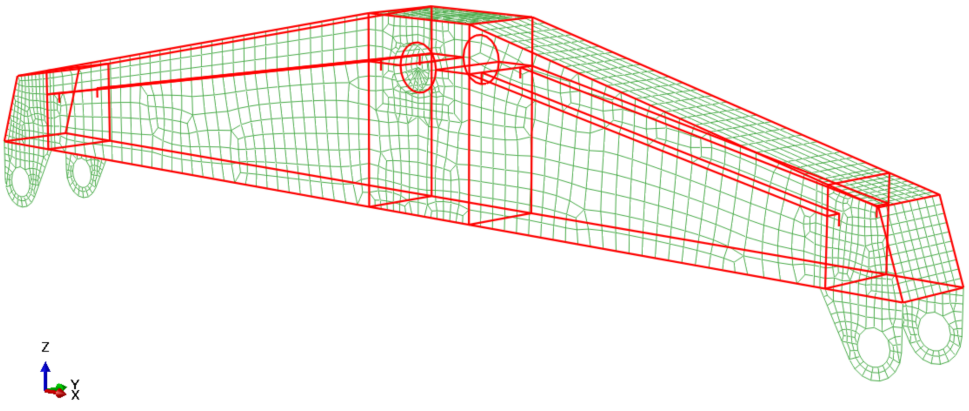

Abbildung 4.20: Modell mit neu generierten Linienelementen

Die transformierten Spannungen nach Ausführen des Skripts *„StressTransformMain.py“* sind in den Abbildungen 4.21 bis 4.23 illustriert. Bei S11* handelt es sich um Spannungen normal zur Schweißnahtachse, bei S22* um Spannungen in Richtung der Schweißnahtachse und bei S12* um die zugehörige Schubspannungen.

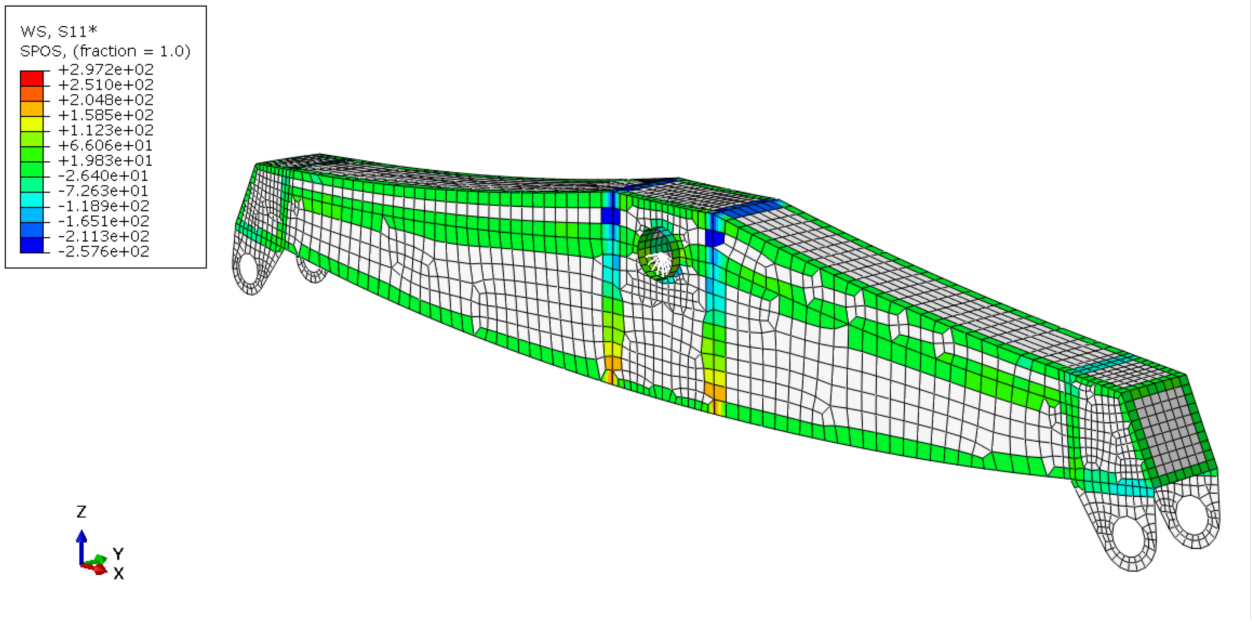

Abbildung 4.21: Spannungen in Richtung normal zur Schweißnahtachse (S11*)

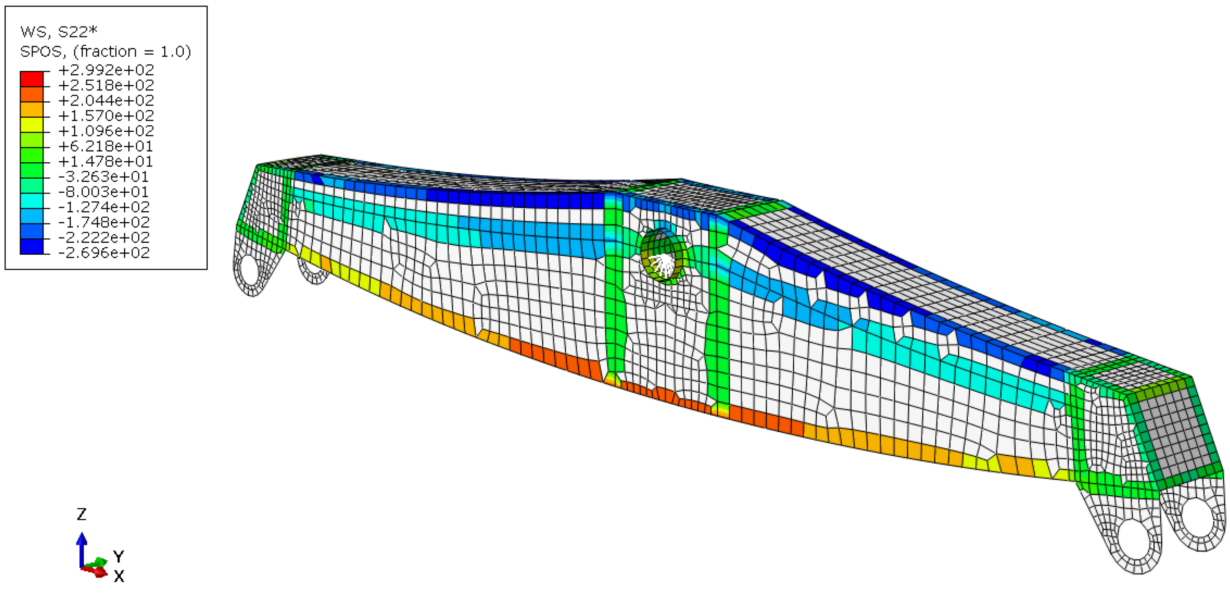

Abbildung 4.22: Spannungen in Richtung der Schweißnahtachse (S22*)

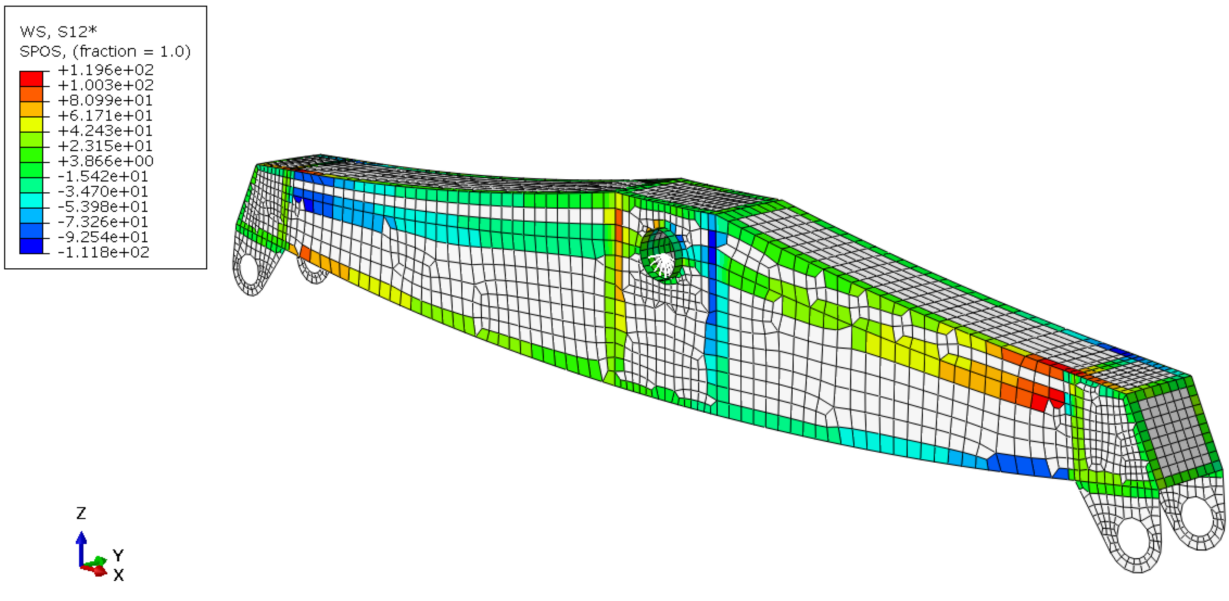

Abbildung 4.23: Schubspannungen (S12*)

5 Zusammenfassung und Ausblick

Ziel dieser Arbeit war das Erstellen von, in Abaqus ausführbaren Python-Skripts zur automatischen Schweißnahtgenerierung auf Basis von FE-Plattenmodellen mit anschließender Spannungstransformation in den angrenzenden Schalen-Elementen. Diese Arbeit wurde in Kooperation mit der Firma *Sandvik Mining Construction Materials Handling GmbH & Co KG* [4] erstellt. Die Python-Skripts sollen als Grundlage für die Bemessung und Auslegung von Schweißnahtdetails in Hinblick auf deren Ermüdungsfestigkeit dienen.

Kriterium für das Markieren einer Schweißnaht ist der Winkel zwischen den Normalvektoren zweier benachbarter Schalen-Elemente. Ein gewisser Toleranzwert kann dabei vom Anwender festgelegt und eingegeben werden. Ist der berechnete Winkel zwischen den Normalvektoren größer bzw. gleich diesem gewählten Toleranzwerts, wird an der gemeinsamen Kante der benachbarten Schalen-Elementen eine Schweißnaht detektiert. Die einzelnen Schweißnähte werden im Modell als Linienelemente dargestellt und zu Element-Sets zusammengefasst.

Vor der Winkelberechnung ist es notwendig eine Liste von Elementkanten und deren angrenzenden Elemente aufzustellen. Hierfür wurden unterschiedliche Algorithmen implementiert, getestet und deren Effizienz aufgezeigt. Es zeigte sich, dass an dieser Stelle das Potential an einzusparendem Zeitaufwand bei der Ausführung des Programms am höchsten ist.

Die Funktionalität der Implementierung wurde an Hand von mehreren Beispielen, welche bestimmte Sonderfälle abdecken, getestet. Dabei stellte sich heraus, dass aufgrund der Diskretisierung mit linearen Elementen an gerundeten Bauteilgeometrien je nach Wahl des Toleranzparameters überflüssige Schweißnähte generiert werden. Diesem Problem kann nur durch manuelles Anpassen der Toleranz entgegnet werden.

Es sei erwähnt, dass Stumpfstöße zweier Plattenbauteile vom Algorithmus der Kantendetektion nicht erfasst werden. Aus diesem Grund wurde ein zusätzliches Python-Skript entwickelt, welches durch manuelles Hinzufügen von Knoten-Sets in Abaqus die umliegenden Elemente in der Spannungstransformation mit berücksichtigt.

Die programmtechnische Umsetzung zur automatisierten Schweißnahtgenerierung und anschließender Transformation der Spannungen in den umliegenden Elementen besitzt durchaus Potential zur Optimierung bzw. Erweiterung und Weiterentwicklung einiger Funktionen.

So beschränkt sich der derzeitige Algorithmus auf Schalen-Elemente mit linearem Ansatz. Eine Erweiterung des Codes auf quadratische Elemente wäre sinnvoll und wünschenswert. Bei der Verwendung von quadratischen Elementen reduziert sich der Approximationsfehler für die Geometrie signifikant. Somit würde sich die Anzahl der falsch markierten Kanten auch bei kleineren Winkeltoleranzen verringern.

Weiters könnte das automatische Erfassen von Stumpfstößen zweier Plattenbauteile realisiert werden, sodass die manuelle Nachbearbeitung nicht mehr notwendig wäre. Der Algorithmus müsste so erweitert werden, dass unterschiedliche Bauteildicken bzw. die Grenzen der, vom Ingenieur definierten Element-Sets erfasst werden.

Die ermittelten transformierten Spannungen dienen als Grundlage für die Berechnung der Ermüdungsfestigkeit von Schweißnahtdetails. Eine mögliche Weiterentwicklung des aktuellen Programms wäre die Implementierung dieser Berechnung, sodass der gesamte Ablauf der Bemessung vollkommen automatisiert erfolgen könnte.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] ÖNORM EN 1993. *Eurocode 3: Bemessung und Konstruktion von Stahlbauten*. Austrian Standards Institute, 2012.

[2] URL `http://www.3ds.com/products-services/simulia/products/abaqus/`. Online Ressource, Abruf: 27.10.2015.

[3] URL `http://things.maths.cam.ac.uk/computing/software/abaqus_docs/docs/v6.12/books/ker/default.htm`. Online Ressource, Abruf: 27.10.2015.

[4] URL `http://mining.sandvik.com/en/contact/contact-information?Country=AT`. Online Ressource, Abruf: 28.10.2015.

[5] Josef Betten. *Finite Elemente für Ingenieure 1*. Springer-Verlag, 1997.

[6] URL `http://www.esocaet.com/wikiplus/index.php/Schalen-Element`. Online Ressource, Abruf: 01.10.2015.

[7] Lutz Nasdala. *FEM-Formelsammlung Statik und Dynamik*. Vieweg+Teubner Verlag, 2012.

[8] Dietmar Gross. *Technische Mechanik 2*. Springer-Verlag, 2009.

[9] URL `https://wiki.ubuntuusers.de/Abaqus`. Online Ressource, Abruf: 05.10.2015.

[10] Dessault Systemes. *Abaqus Analysis User's Manual*. Dessault Systemes, 2012.

[11] Dessault Systemes. *Abaqus Scripting User's Manual*. Dessault Systemes, 2012.

Anhang - Datenträger mit dem Programmcode

Am Datenträger sind sämtliche Python-Skript-Dateien für die Schweißnahtdetektion und Spannungstransformation enthalten. Im speziellen sind das die Dateien:

- *„WeldGeneratorMain.py“*
- *„WeldGenerator.py“*
- *„UpdateMain.py“*
- *„Update.py“*
- *„StressTransformMain.py“*
- *„StressTransform.py“*
- *„methodsA.py“*
- *„methodsB.py“*
- *„methodsC.py“*